MySQL HeatWave User Guide

Abstract

This document describes how to use MySQL HeatWave. It covers how to load data from the DB System and Object Storage, accelerate query processing, optimize workloads, and use the machine learning and generative AI features of MySQL HeatWave.

For information about creating and managing a MySQL HeatWave Cluster on Oracle Cloud Infrastructure (OCI), see *MySQL HeatWave on OCI Service Guide*.

For information about creating and managing a MySQL HeatWave Cluster on Amazon Web Services (AWS), see *MySQL HeatWave on AWS Service Guide*.

For information about creating and managing a MySQL HeatWave Cluster on Oracle Database Service for Azure (ODSA), see *MySQL HeatWave for Azure Service Guide*.

For MySQL Server documentation, refer to the *MySQL Reference Manual*.

For information about the latest MySQL HeatWave features and updates, refer to the MySQL HeatWave Release Notes.

For legal information, see the Legal Notices.

For help with using MySQL, please visit the MySQL Forums, where you can discuss your issues with other MySQL users.

Licensing information. This product may include third-party software, used under license. See MySQL HeatWave License Information User Manual for licensing information, including licensing information relating to third-party software that may be included in this release.

Document generated on: 2026-09-21 (revision: 84928)

Table of Contents

1 Overview 1
1.1 About MySQL HeatWave 1
1.2 MySQL HeatWave Features 3
1.3 About Accessing, Managing, and Connecting to MySQL HeatWave 5
1.4 Supported Cloud Platforms 6
1.5 Integration with Oracle Cloud Services 7
2 Get Started 9
2.1 Review Requirements 9
2.2 Create a MySQL HeatWave DB System 9
2.3 Provision a MySQL HeatWave Cluster 10
2.4 Get a User Account and Required Privileges 10
2.5 Connect to the MySQL HeatWave DB System 11
3 Quickstarts 13
3.1 Quickstart: tpch Analytics 13
3.2 Quickstart: AirportDB Analytics 21
3.3 Quickstart: Create and Use a Machine Learning Model 27
3.4 Quickstart: Set Up a GenAI-Powered Help Chat 34
4 Load and Manage Data 37
4.1 Load Data into DB System 38
4.1.1 Import Data 38
4.1.2 Bulk Ingest Data 38
4.2 Load Data from DB System into MySQL HeatWave Cluster 44
4.2.1 Supported Data Types for MySQL HeatWave 45
4.2.2 Prepare to Load Data 46
4.2.3 Enable or Disable Data Compression 49
4.2.4 About Auto Parallel Load 49
4.2.5 Load Data Using Auto Parallel Load 51
4.2.6 Load Data Manually 54
4.2.7 About Change Propagation 57
4.2.8 Reload Tables 59
4.3 Load Data from Object Storage into MySQL HeatWave Cluster 59
4.3.1 About MySQL HeatWave Lakehouse 59
4.3.2 Additional MySQL HeatWave Lakehouse Requirements 62
4.3.3 MySQL HeatWave Lakehouse Privileges 63
4.3.4 Supported File Formats and Data Types 63
4.3.5 Access Object Storage 74
4.3.6 Lakehouse External Table Syntax 77
4.3.7 About Lakehouse Auto Parallel Load 107
4.3.8 Create External Tables Using Auto Parallel Load 109
4.3.9 Create External Tables Manually 130
4.3.10 Create Tables with Stored Generated Columns 150
4.3.11 Load Structured Data Using Lakehouse Auto Parallel Load 153
4.3.12 Load Structured Data Manually 158
4.3.13 Load Unstructured Data 161
4.3.14 Update Tables 161
4.4 Unload Data from MySQL HeatWave Cluster 178
4.4.1 Unload Data Automatically 179
4.4.2 Unload Data Manually 183
4.5 Retrieve MySQL HeatWave Metadata 184
4.5.1 View Secondary Engine Definitions 184
4.5.2 List Excluded Columns 185
4.6 Export Query Results to Object Storage 186
4.6.1 About Exporting Query Results 186
4.6.2 Requirements to Export Query Results 187
4.6.3 Export Query Results to CSV Files 187

4.6.4 Export Query Results to Parquet Files 195
4.6.5 Export Query Results to ND-JSON Files 200
4.6.6 Redacted PAR URLs in Exported Log Files 206
4.6.7 File Name Format for Exported Query Result Files 206
4.7 Recover DB System Data 207
4.8 Monitor Data Loading 207
4.8.1 MySQL HeatWave Node Status Monitoring 207
4.8.2 MySQL HeatWave Memory Usage Monitoring 208
4.8.3 Data Load Progress and Status Monitoring 208
4.8.4 Change Propagation Monitoring 209
5 Accelerate Query Processing 211
5.1 About MySQL HeatWave 212
5.2 Supported SQL Modes 212
5.3 Supported Functions and Operators 213
5.3.1 Aggregate Functions 213
5.3.2 Arithmetic Operators 218
5.3.3 Cast Functions and Operators 218
5.3.4 Comparison Functions and Operators 219
5.3.5 Control Flow Functions and Operators 220
5.3.6 Data Masking and De-Identification Functions 220
5.3.7 Encryption and Compression Functions 221
5.3.8 JSON Functions 222
5.3.9 Logical Operators 224
5.3.10 Mathematical Functions 224
5.3.11 String Functions and Operators 225
5.3.12 Temporal Functions 228
5.3.13 Vector Functions 230
5.3.14 Window Functions 231
5.3.15 Miscellaneous Functions 231
5.4 Run Queries 232
5.4.1 Requirements for Running Queries 232
5.4.2 SELECT Statement Clauses for MySQL HeatWave 233
5.4.3 Explain Query and Check Execution Time 234
5.4.4 CREATE TABLE ... SELECT Statements 236
5.4.5 Create MySQL HeatWave Temporary Tables 237
5.4.6 INSERT ... SELECT Statements 239
5.4.7 Query Views 240
5.4.8 Query Materalized Views 241
5.4.9 View Query Runtimes and Estimates 244
5.4.10 Diagnose Query Offload 247
5.4.11 Auto Scheduling 254
5.4.12 Auto Query Plan Improvement 255
5.4.13 Dynamic Query Offload 255
5.5 Modify Tables 256
5.6 Analyze Tables 256
5.7 Run Tasks Asynchronously 257
5.8 Optimize Workloads for OLAP 260
5.8.1 String Column Encoding 261
5.8.2 Data Placement Keys 266
5.8.3 About MySQL HeatWave Autopilot Advisor 268
5.8.4 Auto Encoding 269
5.8.5 Auto Data Placement 272
5.8.6 Auto Query Time Estimation 275
5.8.7 Unload Advisor 278
5.8.8 Autopilot Report Table 279
5.8.9 Advisor Report Table 280
5.9 Optimize Workloads for OLTP 281
5.9.1 Autopilot Indexing 282

5.10 MySQL HeatWave Monitoring ... 285
5.10.1 Query Execution Monitoring ... 285
5.10.2 Query History and Statistics Monitoring ... 286
5.10.3 Scanned Data Monitoring ... 288
6 Train and Use Machine Learning Models ... 289
6.1 About MySQL HeatWave AutoML ... 289
6.1.1 MySQL HeatWave AutoML Ease of Use ... 290
6.1.2 MySQL HeatWave AutoML Workflow ... 290
6.1.3 MySQL HeatWave AutoML Learning Types ... 291
6.1.4 Oracle AutoML ... 292
6.2 Additional MySQL HeatWave AutoML Requirements ... 293
6.3 MySQL HeatWave AutoML Privileges ... 294
6.4 Supported Data Types for MySQL HeatWave AutoML ... 295
6.5 Create a Machine Learning Model ... 296
6.5.1 Prepare Data ... 296
6.5.2 Train a Model ... 300
6.5.3 Load a Model ... 301
6.5.4 Generate Predictions ... 303
6.5.5 Generate Model Explanations ... 309
6.5.6 Generate Prediction Explanations ... 312
6.5.7 Score a Model ... 319
6.6 Learn About MySQL HeatWave AutoML with Oracle Cloud Infrastructure Generative AI ... 321
6.7 Machine Learning Use Cases ... 329
6.7.1 Classify Data ... 329
6.7.2 Perform Regression Analysis ... 337
6.7.3 Generate Forecasts ... 344
6.7.4 Detect Anomalies ... 353
6.7.5 Generate Recommendations ... 370
6.7.6 Topic Modeling ... 392
6.7.7 Train a Model in an Interim Isolated Environment ... 397
6.8 Use MySQL HeatWave AutoML with Lakehouse ... 402
6.9 Manage Machine Learning Models ... 408
6.9.1 The Model Catalog ... 408
6.9.2 Work with Model Handles ... 415
6.9.3 Unload a Model ... 418
6.9.4 View Model Details ... 420
6.9.5 Delete a Model ... 422
6.9.6 Share a Model ... 423
6.9.7 Manage External ONNX Models ... 426
6.9.8 Analyze Data Drift ... 440
6.10 Track Progress for MySQL HeatWave AutoML Routines ... 446
6.11 Monitor the Status of MySQL HeatWave AutoML ... 450
7 Perform AI-Powered Search and Content Generation ... 453
7.1 About MySQL HeatWave GenAI ... 453
7.2 Additional MySQL HeatWave GenAI Requirements ... 454
7.3 MySQL HeatWave GenAI Roles and Privileges ... 455
7.4 Supported Models and Languages ... 456
7.5 Authenticate OCI Generative AI Service ... 461
7.6 Generate Text-Based Content ... 462
7.6.1 Generate New Content ... 462
7.6.2 Summarize Content ... 465
7.6.3 Summarize Unstructured Files ... 469
7.7 Set Up a Vector Store ... 471
7.7.1 About MySQL HeatWave Vector Store ... 471
7.7.2 Ingest Files into a Vector Store ... 473
7.7.3 Update a Vector Store ... 481
7.8 Generate Vector Embeddings ... 483
7.9 Perform Vector Search with RAG ... 486

7.9.1 Run Retrieval-Augmented Generation 486
7.9.2 Use Your Own Embeddings 494
7.9.3 Automatic Vector Index Creation 506
7.10 Start a Conversational Chat 506
7.10.1 Run MySQL HeatWave Chat 506
7.10.2 View Chat Session Details 510
7.11 Generate SQL Queries From Natural-Language Statements 511
8 Performance and Usage 515
8.1 MySQL HeatWave Performance Schema Tables 515
8.1.1 The rpd_column_id Table 515
8.1.2 The rpd_columns Table 516
8.1.3 The rpd_exec_stats Table 516
8.1.4 The rpd_mirror Table 517
8.1.5 The rpd_ml_stats Table 517
8.1.6 The rpd_nodes Table 519
8.1.7 The rpd_preload_stats Table 521
8.1.8 The rpd_query_stats Table 521
8.1.9 The rpd_table_id Table 523
8.1.10 The rpd_tables Table 523
8.1.11 The rpd_vector_indexes Table 528
8.2 Option Tracker 528
9 System and Status Variables 531
9.1 System Variables 531
9.2 Status Variables 542
10 Routines 545
10.1 MySQL HeatWave Routines 545
10.1.1 HEATWAVE_LOAD 545
10.1.2 HEATWAVE_UNLOAD 550
10.1.3 HEATWAVE_RELOAD 553
10.1.4 Asynchronous Task Routines 553
10.1.5 HEATWAVE_ADVISOR 573
10.2 MySQL HeatWave AutoML Routines 575
10.2.1 ML_TRAIN 575
10.2.2 ML_EXPLAIN 589
10.2.3 ML_MODEL_EXPORT 591
10.2.4 ML_MODEL_IMPORT 593
10.2.5 ML_PREDICT_ROW 597
10.2.6 ML_PREDICT_TABLE 602
10.2.7 ML_EXPLAIN_ROW 608
10.2.8 ML_EXPLAIN_TABLE 612
10.2.9 ML_SCORE 614
10.2.10 ML_MODEL_LOAD 616
10.2.11 ML_MODEL_UNLOAD 617
10.2.12 ML_MODEL_ACTIVE 617
10.2.13 NL2ML 620
10.2.14 TRAIN_TEST_SPLIT 624
10.2.15 Model Types 626
10.2.16 Optimization and Scoring Metrics 628
10.3 MySQL HeatWave GenAI Routines 631
10.3.1 ML_GENERATE 631
10.3.2 ML_GENERATE_TABLE 635
10.3.3 VECTOR_STORE_LOAD 641
10.3.4 ML_RAG 645
10.3.5 ML_RAG_TABLE 650
10.3.6 HEATWAVE_CHAT 655
10.3.7 ML_EMBED_ROW 660
10.3.8 ML_EMBED_TABLE 662
10.3.9 NL_SQL 665

10.3.10 ML_RETRIEVE_SCHEMA_METADATA 668
11 Troubleshoot 671
11.1 Issues and Errors 671
11.1.1 MySQL HeatWave Issues 671
11.1.2 MySQL HeatWave Errors 674
11.1.3 MySQL HeatWave AutoML Error Messages 699
11.1.4 MySQL HeatWave GenAI Issues 719
11.1.5 MySQL HeatWave Lakehouse Error Messages 720
11.2 Limitations 723
11.2.1 MySQL HeatWave Limitations 723
11.2.2 MySQL HeatWave Lakehouse Limitations 731
11.2.3 MySQL HeatWave AutoML Limitations 737

Chapter 1 Overview

Table of Contents

1.1 About MySQL HeatWave 1
1.2 MySQL HeatWave Features 3
1.3 About Accessing, Managing, and Connecting to MySQL HeatWave 5
1.4 Supported Cloud Platforms 6
1.5 Integration with Oracle Cloud Services 7

The sections in this chapter describe MySQL HeatWave.

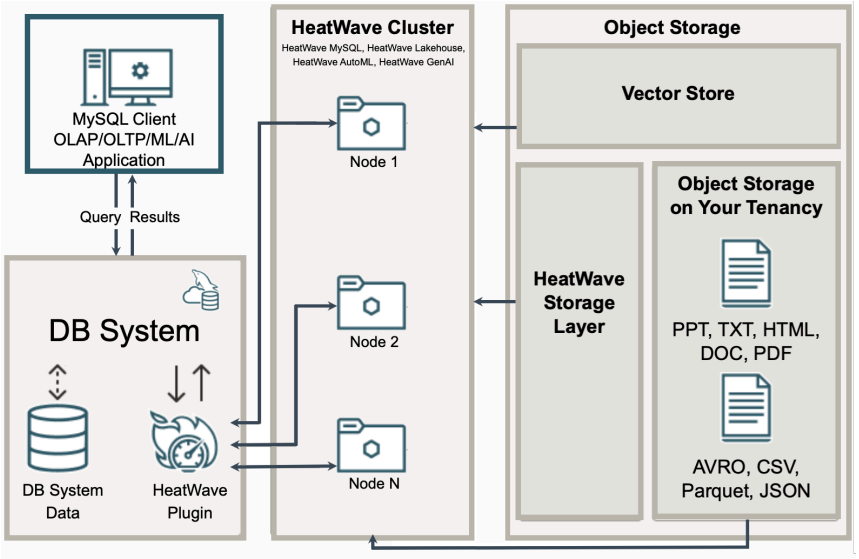
1.1 About MySQL HeatWave

MySQL HeatWave is a fully-managed cloud service that enables organizations to efficiently run analytics, transactional processing, machine learning, generative AI, and vector processing using a single service without the need for extract, transfer, and load (ETL).

MySQL HeatWave consists of a massively parallel, high-performance, in-memory query accelerator that improves query performance by orders of magnitude. This enables organizations to efficiently perform analytical and transactional processing. The scale-out architecture of MySQL HeatWave allows loading and processing data available in the DB System as well as Object Storage, delivering unmatched performance and price-performance. It can process structured, semi-structured, and unstructured data available in Object Storage. Additionally, organizations can leverage the machine learning (ML) and generative artificial intelligence (GenAI) features of MySQL HeatWave for building predictive models, creating and enhancing AI-powered applications using proprietary data, and improving the overall data management.

MySQL HeatWave Architecture

Figure 1.1 Architectural Features of MySQL HeatWave



The MySQL HeatWave service consists of two main components:

- *DB System*

  The DB System is a cloud-based Compute instance with MySQL Enterprise Edition installed. The Enterprise edition has advanced security features for encryption, data masking, authentication, and a database firewall. MySQL HeatWave is the only cloud service built on MySQL Enterprise Edition.

- *MySQL HeatWave Cluster*

  MySQL HeatWave Cluster serves as the in-memory accelerator that significantly enhances query performance. It also provides access to advanced machine learning (ML), generative artificial intelligence (GenAI), vector store, and Lakehouse capabilities. MySQL HeatWave Cluster consists of one or more MySQL HeatWave nodes. When data is loaded into MySQL HeatWave Cluster from the DB System or Object Storage, the data is sharded and distributed among MySQL HeatWave nodes.

  MySQL HeatWave Cluster enables you to perform real-time analytics and machine learning on the structured data residing in the DB System. The traffic between the DB System and MySQL HeatWave Cluster is fully encrypted. Any changes made to the data in the DB System are automatically propagated to MySQL HeatWave Cluster, ensuring that queries always access the most up-to-date information.

  MySQL HeatWave Cluster also enables you to perform analytics, machine learning, generative AI, and vector processing on the structured, semi-structured, and unstructured data residing in Object Storage. Any changes to the data in Object Storage is automatically propagated to MySQL HeatWave Cluster as well.

  Data that is loaded into MySQL HeatWave Cluster is automatically persisted to the network-attached block storage, also known as MySQL HeatWave Storage Layer, which allows data to be reloaded quickly when the MySQL HeatWave Cluster resumes after a pause or when MySQL HeatWave Cluster recovers from a cluster or node failure. The vector store tables created with MySQL HeatWave GenAI also persist in the block storage.

Design Characteristics for Unmatched Performance

The following design characteristics of MySQL HeatWave help deliver a compelling performance and cost advantage:

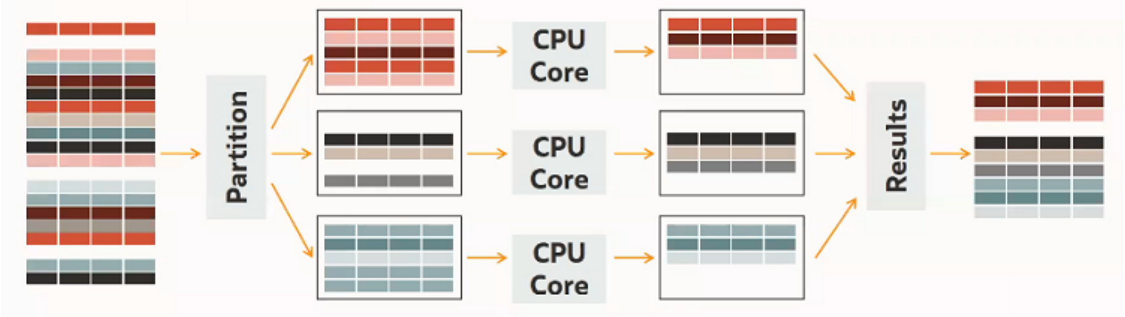
- *In-Memory Hybrid-Columnar Format*

  MySQL HeatWave Cluster stores data in main memory in a hybrid columnar format. The hybrid approach achieves the benefits of columnar format for query processing, while avoiding the materialization and update costs associated with pure columnar format. Hybrid columnar format enables the use of efficient query processing algorithms designed to operate on fixed-width data, and permits vectorized query processing.

- *Massively Parallel Architecture*

  The MySQL HeatWave massively parallel architecture uses internode and intranode partitioning of data. Each node within a MySQL HeatWave Cluster, and each CPU core within a node, processes the partitioned data in parallel. MySQL HeatWave is capable of scaling to thousands of cores. This massively parallel architecture, combined with high-fanout, workload-aware partitioning, accelerates query processing.

  Figure 1.2 MySQL HeatWave Massively Parallel Architecture

- *Push-Based Vectorized Query Processing*

MySQL HeatWave processes queries by pushing vector blocks (slices of columnar data) through the query execution plan from one operator to another. A push-based execution model avoids deep call stacks and saves valuable resources compared to tuple-based processing models.

- *Scale-Out Data Management*

  When data is loaded into MySQL HeatWave, the MySQL HeatWave Storage Layer automatically persists the data for pause and resume of the MySQL HeatWave Cluster and for fast recovery in case of a MySQL HeatWave node or cluster failure. Data is automatically restored by the MySQL HeatWave Storage Layer when the MySQL HeatWave Cluster resumes after a pause or recovers a failed node or cluster. This automated, self-managing storage layer scales to the size required for your MySQL HeatWave Cluster and operates independently in the background. MySQL HeatWave on OCI persists the data to OCI Object Storage. MySQL HeatWave on AWS persists the data to AWS S3.

- *Inbuilt Generative AI and Machine Learning Capabilities*

  MySQL HeatWave provides native generative AI and machine learning capabilities integrated directly into the database engine, eliminating the need for separate AI and ML infrastructure. This integrated approach enables organizations to perform both traditional predictive analytics and modern generative AI workloads without data movement, reducing latency and improving security while maintaining the performance benefits of in-memory processing.

- *Automated Performance Improvement*

  MySQL HeatWave Autopilot automates many of the most important and often challenging aspects of achieving high query performance at scale, including system setup, data loading, query execution and failure handling. It uses advanced machine learning techniques to sample data, collect statistics on data and queries, and build machine learning models to model memory usage, network load and execution time. MySQL HeatWave Autopilot uses these ML models to intelligently learn from queries executed in the system, resulting in continually improving system performance over time.

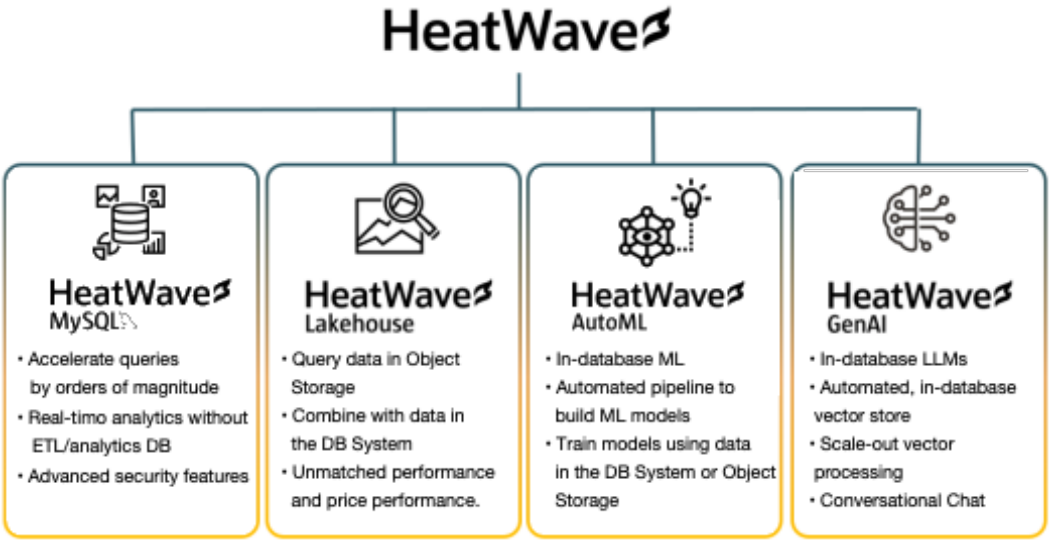
What's Next

- Learn about different features of MySQL HeatWave.
- To learn more about the technical aspects of MySQL HeatWave, see the MySQL HeatWave Technical Brief.

1.2 MySQL HeatWave Features

This topic introduces the different features of MySQL HeatWave.

Figure 1.3 MySQL HeatWave Features



MySQL HeatWave

MySQL HeatWave is ideal for data warehousing, real-time analytics, mixed workloads, transactional processing, and machine learning workloads when the main data source is the DB System without needing ETL.

When you submit a query to the DB System from a MySQL client or application, a query optimizer automatically determines whether the query should be offloaded to MySQL HeatWave Cluster for accelerated processing. This decision is based on two factors: first, whether all operators and functions referenced in the query are supported by MySQL HeatWave Cluster, and second, whether the estimated processing time with MySQL HeatWave Cluster is less than that with the MySQL instance. If both conditions are met, the query is sent to MySQL HeatWave Cluster for accelerated processing. Once the query is processed, the results are returned to the MySQL instance and the client or application that issued the query.

To learn more about MySQL, see MySQL 26.7 Reference Manual.

Real-time Analytics Without ETL

Analytics queries always access the latest data, as updates from transactions in MySQL automatically replicate in real time to MySQL HeatWave. A MySQL HeatWave Cluster can accommodate up to 512 nodes. The number of nodes required depends on the size of the data and the amount of compression achieved when loading data into MySQL HeatWave Cluster. Data loaded into MySQL HeatWave is automatically stored in Object Storage, allowing for quick reloading when MySQL HeatWave Cluster resumes after a pause or recovers from a cluster or node failure.

Transactional Processing

MySQL HeatWave is built on MySQL Enterprise Edition and provides the same robust transactional capabilities as a fully-managed cloud-based database service. Additionally, MySQL HeatWave supports JavaScript-stored programs, enabling you to express complex procedural logic within the database and seamlessly access MySQL datasets without incurring additional ETL costs.

Learn more about accelerating query processing with MySQL HeatWave .

MySQL HeatWave Lakehouse

MySQL HeatWave Lakehouse enables you to query structured and semi-structured data (such as Avro, CSV, JSON, and Parquet) and unstructured data (such as documents, text files, and

presentations) stored in Object Storage.. The query processing occurs within the MySQL HeatWave Cluster, enabling you to utilize MySQL HeatWave for both MySQL and non-MySQL workloads. The data in Object Storage is not copied into the MySQL InnoDB storage engine, thus avoiding duplication of data.

The service reads data from Object Storage, transforms it into the memory-optimized MySQL HeatWave format, and stores it in MySQL HeatWave Storage Layer within Object Storage. This data is then loaded into the memory of the MySQL HeatWave Cluster, making it available for queries.

Learn more about loading data from Object Storage into MySQL HeatWave Cluster with Lakehouse and using Lakehouse with MySQL HeatWave AutoML.

MySQL HeatWave AutoML

MySQL HeatWave AutoML is a fully-managed, highly-scalable, and cost-efficient machine learning (ML) solution for data stored in both DB System and Object Storage. It offers a simple SQL interface for training and utilizing predictive machine learning models, making it accessible to both novice and experienced ML practitioners. No specialized knowledge, tools, or algorithms are required.

With MySQL HeatWave AutoML, you can train a model with a single call to a stored routine. Similarly, generating predictions only requires a single `CALL` or `SELECT` statement, which can be easily integrated into your applications.

One of the key advantages of MySQL HeatWave AutoML is that your data and models remain within the secure MySQL HeatWave environment, saving you time and effort. MySQL HeatWave distributes ML computations across its nodes to leverage scalability and take full advantage of its massively parallel processing capabilities.

Learn more about training and using machine learning models with MySQL HeatWave AutoML.

MySQL HeatWave GenAI

MySQL HeatWave GenAI offers integrated and automated generative AI featuring in-database large language models (LLMs), in-database embedding generation, in-database vector store, scale-out vector processing, and the ability to have contextual conversations in natural language. This enables you to leverage generative AI without needing AI expertise, transferring data, or incurring additional costs.

Additionally, MySQL HeatWave GenAI features MySQL HeatWave Chat, a chatbot that enhances the generative AI and vector search capabilities. This enables you to ask multiple follow-up questions about a specific topic within a single session. MySQL HeatWave Chat can also access information from documents stored in the vector store.

Learn more about performing AI-Powered search and content generation with MySQL HeatWave GenAI.

What's Next

Learn about accessing, managing, and connecting to MySQL HeatWave.

1.3 About Accessing, Managing, and Connecting to MySQL HeatWave

MySQL HeatWave is designed as a pluggable engine for the DB System. You can manage the DB System and MySQL HeatWave Cluster using the same interface. This native integration provides a unified data management platform for Online Transaction Processing (OLTP), Online Analytical Processing (OLAP), machine learning, generative AI, and Lakehouse workloads.

MySQL HeatWave supports multiple ways of managing the service:

- Using a web-based cloud console.
- Using a command-line interface.
- Using REST API. The Oracle Cloud Infrastructure APIs are typical REST APIs that use HTTPS requests and responses.

For more information, see the *Ways to Access MySQL HeatWave Service* section in OCI Service Guide and AWS Service Guide, and see the *Connecting to MySQL HeatWave* section in the Oracle Database Service for Azure documentation.

Clients and applications connect with MySQL HeatWave via the DB System, which is a MySQL instance, utilizing standard tools and ODBC/JDBC connectors. MySQL HeatWave complies with the same ANSI SQL standard and ACID properties as MySQL, supporting the most commonly used data types. This compatibility allows existing applications to leverage MySQL HeatWave without any modifications, facilitating quick and easy integration.

For more information, see Section 2.5, “Connect to the MySQL HeatWave DB System”.

What's Next

Learn more about the supported cloud platforms from where you can access and manage MySQL HeatWave.

1.4 Supported Cloud Platforms

The MySQL HeatWave service runs completely on cloud infrastructure. This means that all the layers and components of MySQL HeatWave are deployed and fully managed on selected cloud platforms.

MySQL HeatWave also supports hybrid deployments; it is fully compatible with MySQL on-premises. This provides you with the option to keep your OLTP applications on-premises and offload your analytics, machine learning, generative AI workloads to MySQL HeatWave on cloud.

MySQL HeatWave can be accessed and managed on the following cloud platforms:

- Oracle Cloud Infrastructure (OCI): MySQL HeatWave is natively available on public cloud in all available OCI regions, OCI Dedicated Region, and Oracle Alloy.
- Amazon Web Services (AWS): MySQL HeatWave is natively available on AWS, which enables applications already deployed in AWS to benefit from the DB System without incurring the latency associated with accessing a database service running outside of AWS. This helps to avoid the high data egress fees charged by AWS that would be necessary to migrate data to a service running outside of AWS. Lastly, the tight integration of the DB System with the AWS infrastructure, including Amazon S3, CloudWatch, PrivateLink, makes it easy for developers to rely on the DB System for new applications.
- Microsoft Azure: Oracle Database Service for Azure (ODSA) is an Oracle-managed solution that provides an Azure-like portal to provision OCI database services and seamless connectivity between the Azure application tier and OCI database tier. Also, MySQL HeatWave is available on Azure via Azure interconnect. The OCI-Azure Interconnect is a customer-managed network service that facilitates the deployment of any workload requiring cross-cloud network connections.

On each platform, MySQL HeatWave supports one or more regions where user accounts are provisioned.

What's Next

Learn about integration with Oracle Cloud Services.

1.5 Integration with Oracle Cloud Services

Oracle Cloud Infrastructure (OCI) offers a wide range of services for data analytics, machine learning, and data lake. Native integration with these services makes it easier for existing applications to use MySQL HeatWave.

- *Object Storage*: Object Storage is a fully programmable, scalable, and durable cloud storage service that enables you to store and easily access an unlimited amount of data at low cost. MySQL HeatWave leverages parallel processing to read and load data from files stored in Object Storage buckets. You can also use parallel processing to export data from MySQL HeatWave to Object Storage buckets.
- *Oracle Analytics Cloud (OAC)*: It provides the industry's most comprehensive cloud analytics in a single unified platform, including self-service visualization and inline data preparation to enterprise reporting, advanced analytics, and self-learning analytics that deliver proactive insights. Integration with OAC provides a BI visualization platform for users to analyze their MySQL data.
- *OCI Data Integration Service*: It provides extract, transform and load (ETL) capabilities to target data warehousing scenarios on the OCI platform. It supports various data sources, starting with relational, cloud, and Hadoop. Integration with OCI Data Integration allows users to easily transform and import data from data sources other than MySQL to MySQL HeatWave, expanding the scope of data that can take advantage of MySQL HeatWave.
- *OCI Database Management and Ops Insight*: It provides comprehensive database performance diagnostics and management capabilities for MySQL HeatWave. It also provides information about resource use and capacity of databases and hosts for users to analyze CPU and storage resources, forecast capacity issues, and proactively identify SQL performance issues across a MySQL HeatWave database fleet.

What's Next

Learn how to get started with MySQL HeatWave.

Chapter 2 Get Started

Table of Contents

2.1 Review Requirements 9
2.2 Create a MySQL HeatWave DB System 9
2.3 Provision a MySQL HeatWave Cluster 10
2.4 Get a User Account and Required Privileges 10
2.5 Connect to the MySQL HeatWave DB System 11

The sections in this chapter describe how to prepare to use MySQL HeatWave.

2.1 Review Requirements

You need an operational MySQL HeatWave DB System with the MySQL HeatWave Cluster enabled to do the following:

- Load and manage data in the MySQL HeatWave Cluster.
- Accelerate query performance with MySQL HeatWave.
- Train and use machine learning models with MySQL HeatWave AutoML.
- Perform AI-powered search and text generation with MySQL HeatWave GenAI.

Some of these components have some additional requirements that are listed in the following sections:

- Additional requirements for using MySQL HeatWave Lakehouse.
- Additional requirements for using MySQL HeatWave AutoML.
- Additional requirements for using MySQL HeatWave GenAI.

Note

MySQL HeatWave AutoML and MySQL HeatWave GenAI are supported on DB Systems with 32 or less MySQL HeatWave Cluster nodes only.

What's Next

Learn more about creating a MySQL HeatWave DB System.

2.2 Create a MySQL HeatWave DB System

To use the additional components of MySQL HeatWave that are included with the MySQL HeatWave Cluster, you must first create a MySQL HeatWave DB System.

Before You Begin

Review the requirements to use MySQL HeatWave.

Create a DB System

- For MySQL HeatWave on Oracle Cloud Infrastructure (OCI), see Creating a DB System in the *MySQL HeatWave on OCI Service Guide*.
- For MySQL HeatWave on Amazon Web Services (AWS), see Creating a DB System in the *MySQL HeatWave on AWS Service Guide*.

- For MySQL HeatWave for Azure, see Provisioning MySQL HeatWave in the *MySQL HeatWave for Azure Service Guide*.

What's Next

Learn more about provisioning a MySQL HeatWave Cluster in the DB System.

2.3 Provision a MySQL HeatWave Cluster

After you have created the DB System, to use the additional components of MySQL HeatWave that are included with the MySQL HeatWave Cluster, enable and provision a MySQL HeatWave Cluster in the DB System.

Before You Begin

Create a MySQL HeatWave DB System.

Provision a MySQL HeatWave Cluster

To learn how to enable and provision a MySQL HeatWave Cluster in the DB System, see the following:

- For MySQL HeatWave on Oracle Cloud Infrastructure (OCI), see Adding a MySQL HeatWave Cluster in the *MySQL HeatWave on OCI Service Guide*.
- For MySQL HeatWave on Amazon Web Services (AWS), see Creating a MySQL HeatWave Cluster in the *MySQL HeatWave on AWS Service Guide*.
- For MySQL HeatWave for Azure, see Provisioning MySQL HeatWave Nodes in the *MySQL HeatWave for Azure Service Guide*.

What's Next

Learn more about getting the required privileges.

2.4 Get a User Account and Required Privileges

By default, the DB System provides only one user account for the DB System administrator. To use the additional features of MySQL HeatWave Cluster for different purposes, you might to ask your DB System administrator to create a new user account for you.

Some privileges are granted to all user accounts by default, whereas for some features, you need to ask the DB System administrator to grant you the required additional privileges.

This topic provides links to relevant documentation to help you learn about the available and required privileges for using MySQL HeatWave and its features.

Before You Begin

1. Create a MySQL HeatWave DB System.
2. Provision a MySQL HeatWave Cluster.

Review the Available and Required Privileges

To learn about the available privileges, see Privileges Provided by MySQL and Default MySQL Privileges.

To review the privileges required to use the additional components of MySQL HeatWave that are included with MySQL HeatWave Cluster, see the following topics:

- Section 4.3.3, “MySQL HeatWave Lakehouse Privileges”
- Section 6.3, “MySQL HeatWave AutoML Privileges”
- Section 7.3, “MySQL HeatWave GenAI Roles and Privileges”

Create a User Account and Grant Privileges

To create a new user account, the DB System administrator can use the following statement:

```
mysql> CREATE USER Username IDENTIFIED BY 'Password';
```

For example:

```
mysql> CREATE USER appuser1 IDENTIFIED BY 'Pa$$w0rd';
```

This creates a new user account named `appuser1` with the password `Pa$$w0rd`.

To grant a privilege, the DB System administrator can use the following command:

- To grant privileges on a particular schema:

  ```
  mysql> GRANT List_Of_Privileges ON DB_Name.* TO 'Username';
  ```

  For example:

  ```
  mysql> GRANT ALL ON demo_db.* TO 'appuser1';
  ```

  This grants all available privileges on the schema `demo_db` to the user account `appuser1`.

- To grant privileges on all schemas:

  ```
  mysql> GRANT List_Of_Privileges ON *.* TO 'Username';
  ```

  For example:

  ```
  mysql> GRANT SELECT, ALTER ON *.* TO 'appuser1';
  ```

  This grants the specified privileges, `SELECT` and `ALTER`, on all schemas to the user account `appuser1`.

What's Next

- Learn more about connecting to the MySQL HeatWave DB System.

2.5 Connect to the MySQL HeatWave DB System

This topic provides links to relevant documents that describe how to connect to a MySQL HeatWave DB System.

Before You Begin

1. Create a MySQL HeatWave DB System.
2. Provision a MySQL HeatWave Cluster.
3. Get the required privileges to use MySQL HeatWave.

Connect to the DB System

After you have created and configured your DB System, connect to the DB System.

- For MySQL HeatWave on Oracle Cloud Infrastructure (OCI), see Connecting to a DB System in the *MySQL HeatWave on OCI Service Guide*.
- For MySQL HeatWave on Amazon Web Services (AWS), see Connecting with MySQL Shell in the *MySQL HeatWave on AWS Service Guide*.

What's Next

- See Chapter 3, *Quickstarts*.
- Learn more about loading and managing data in MySQL HeatWave.

Chapter 3 Quickstarts

Table of Contents

3.1 Quickstart: tpch Analytics 13
3.2 Quickstart: AirportDB Analytics 21
3.3 Quickstart: Create and Use a Machine Learning Model 27
3.4 Quickstart: Set Up a GenAI-Powered Help Chat 34

3.1 Quickstart: tpch Analytics

The *tpch Analytics Quickstart* shows how to import data into the DB System using the MySQL Shell Parallel Table Import Utility, manually loading data into MySQL HeatWave Cluster, and running queries.

This quickstart contains the following sections:

- tpch Requirements
- Generate tpch Sample Data
- Create the tpch Sample Database and Import Data
- Load tpch Data Into MySQL HeatWave
- Run tpch Queries
- Additional tpch Queries
- Unload tpch Tables

tpch Requirements

- Review the MySQL HeatWave Quickstart Requirements.
- MySQL Shell 8.0.22 or higher. We recommend that you always run the latest version of MySQL Shell, which is currently MySQL version 9.3.0. You can download the latest version from MySQL downloads.
- Linux system with `gcc` and `make` libraries installed, to generate `tpch` sample data using the `dbgen` utility.

Generate tpch Sample Data

Examples in this Quickstart use the `tpch` sample database, which is an ad-hoc decision support database derived from the TPC Benchmark™ H (TPC-H) specification. For an overview of the `tpch` schema, refer to the *Logical Database Design* section of the specification document.

The following instructions describe how to generate `tpch` sample data using the `dbgen` utility. The instructions assume you are on a Linux system that has `gcc` and `make` libraries installed.

To generate `tpch` sample data:

1. Download the TPC-H tools zip file from TPC Download Current.
2. Extract the zip file to a local directory on your system.
3. Change to the `dbgen` directory and make a copy of the makefile template.

```
$> cd 2.18.0/dbgen
$> cp makefile.suite makefile
```

4. Configure the variables *CC*, *DATABASE*, *MACHINE* and *WORKLOAD* in the makefile, as follows:

```
CC = gcc
DATABASE = ORACLE
MACHINE = LINUX
WORKLOAD = TPCH
```

5. Run `make` to build the `dbgen` utility:

```
$> make
```

6. Run the following `dbgen` command to generate a 1GB set of data files for the `tpch` database:

```
$> ./dbgen -s 1
```

The operation may take a few minutes. When finished, the following data files appear in the working directory, one for each table in the `tpch` database:

```
$> ls -1 *.tbl
customer.tbl
lineitem.tbl
nation.tbl
orders.tbl
partsupp.tbl
part.tbl
region.tbl
supplier.tbl
```

Create the tpch Sample Database and Import Data

This topic describes how to create the tpch sample database on the DB System and import the sample data. The sample data must be available on the DB System before it can be loaded into the MySQL HeatWave Cluster.

Sample database creation and import operations are performed using MySQL Shell. We recommend that you always run the latest version of MySQL Shell, which is currently MySQL version 9.3.0. You can download the latest version from MySQL downloads".

The MySQL Shell Parallel Table Import Utility provides fast data import for large data files. The utility analyzes an input data file, divides it into chunks, and uploads the chunks to the target DB System using parallel connections. The utility is capable of completing a large data import many times faster than a standard single-threaded upload using a `LOAD DATA` statement. For additional information, see Parallel Table Import Utility.

To create the tpch sample database on the DB System and import data:

1. Start MySQL Shell with `--mysql` option and connect to the DB System endpoint.

```
$> mysqlsh --mysql Username@DBSystem_IP_Address_or_Host_Name
```

The `--mysql` option opens a `ClassicSession`, which is required when using the MySQL Shell Parallel Table Import Utility.

Note

MySQL Shell opens in `SQL` execution mode by default. If it opens in `JavaScript (JS)` mode, run the following command to switch to `SQL` execution mode.

```
mysql-js> \sql
```

2. Create the `tpch` sample database and tables:

```
mysql-sql> CREATE DATABASE tpch character set utf8mb4;
mysql-sql> USE tpch;
```

```
mysql-sql> CREATE TABLE nation  ( N_NATIONKEY INTEGER primary key,
    N_NAME       CHAR(25) NOT NULL,
    N_REGIONKEY  INTEGER NOT NULL,
    N_COMMENT    VARCHAR(152));

mysql-sql> CREATE TABLE region  ( R_REGIONKEY INTEGER primary key,
    R_NAME       CHAR(25) NOT NULL,
    R_COMMENT    VARCHAR(152));

mysql-sql> CREATE TABLE part  ( P_PARTKEY INTEGER primary key,
    P_NAME        VARCHAR(55) NOT NULL,
    P_MFGR        CHAR(25) NOT NULL,
    P_BRAND       CHAR(10) NOT NULL,
    P_TYPE        VARCHAR(25) NOT NULL,
    P_SIZE        INTEGER NOT NULL,
    P_CONTAINER   CHAR(10) NOT NULL,
    P_RETAILPRICE DECIMAL(15,2) NOT NULL,
    P_COMMENT     VARCHAR(23) NOT NULL );

mysql-sql> CREATE TABLE supplier  ( S_SUPPKEY INTEGER primary key,
    S_NAME        CHAR(25) NOT NULL,
    S_ADDRESS     VARCHAR(40) NOT NULL,
    S_NATIONKEY   INTEGER NOT NULL,
    S_PHONE       CHAR(15) NOT NULL,
    S_ACCTBAL     DECIMAL(15,2) NOT NULL,
    S_COMMENT     VARCHAR(101) NOT NULL);

mysql-sql> CREATE TABLE partsupp  ( PS_PARTKEY INTEGER NOT NULL,
    PS_SUPPKEY     INTEGER NOT NULL,
    PS_AVAILQTY    INTEGER NOT NULL,
    PS_SUPPLYCOST  DECIMAL(15,2)  NOT NULL,
    PS_COMMENT     VARCHAR(199) NOT NULL, primary key (ps_partkey, ps_suppkey) );

mysql-sql> CREATE TABLE customer  ( C_CUSTKEY INTEGER primary key,
    C_NAME        VARCHAR(25) NOT NULL,
    C_ADDRESS     VARCHAR(40) NOT NULL,
    C_NATIONKEY   INTEGER NOT NULL,
    C_PHONE       CHAR(15) NOT NULL,
    C_ACCTBAL     DECIMAL(15,2)   NOT NULL,
    C_MKTSEGMENT  CHAR(10) NOT NULL,
    C_COMMENT     VARCHAR(117) NOT NULL);

mysql-sql> CREATE TABLE orders  ( O_ORDERKEY INTEGER primary key,
    O_CUSTKEY        INTEGER NOT NULL,
    O_ORDERSTATUS    CHAR(1) NOT NULL,
    O_TOTALPRICE     DECIMAL(15,2) NOT NULL,
    O_ORDERDATE      DATE NOT NULL,
    O_ORDERPRIORITY  CHAR(15) NOT NULL,
    O_CLERK          CHAR(15) NOT NULL,
    O_SHIPPRIORITY   INTEGER NOT NULL,
    O_COMMENT        VARCHAR(79) NOT NULL);

mysql-sql> CREATE TABLE lineitem ( L_ORDERKEY INTEGER NOT NULL,
    L_PARTKEY     INTEGER NOT NULL,
    L_SUPPKEY     INTEGER NOT NULL,
    L_LINENUMBER  INTEGER NOT NULL,
    L_QUANTITY    DECIMAL(15,2) NOT NULL,
    L_EXTENDEDPRICE  DECIMAL(15,2) NOT NULL,
    L_DISCOUNT    DECIMAL(15,2) NOT NULL,
    L_TAX         DECIMAL(15,2) NOT NULL,
    L_RETURNFLAG  CHAR(1) NOT NULL,
    L_LINESTATUS  CHAR(1) NOT NULL,
    L_SHIPDATE    DATE NOT NULL,
    L_COMMITDATE  DATE NOT NULL,
    L_RECEIPTDATE DATE NOT NULL,
    L_SHIPINSTRUCT CHAR(25) NOT NULL,
    L_SHIPMODE     CHAR(10) NOT NULL,
    L_COMMENT      VARCHAR(44) NOT NULL,
    primary key(L_ORDERKEY,L_LINENUMBER));
```

3. Verify that the `tpch` schema and tables were created:

```
mysql-sql> SHOW TABLES;
+----------------+
| Tables_in_tpch |
+----------------+
| customer       |
| lineitem       |
| nation         |
| orders         |
| part           |
| partsupp       |
| region         |
| supplier       |
+----------------+
```

4. Switch to JavaScript execution mode to use the Parallel Table Import Utility:

```
mysql>sql> \js
```

5. Execute the following operations to import the data into the `tpch` database on the DB System.

Note

For information about the `util.importTable()` options used in the following commands, see Parallel Table Import Utility. The number of parallel threads specified using the `threads` option depends on the number of CPU cores of the shape. It is assumed that sample data fields are terminated by the pipe "`|`" character.

```
mysql-js> util.importTable("nation.tbl",{table: "nation", fieldsTerminatedBy:"|",
bytesPerChunk:"100M", threads:16})

mysql-js> util.importTable("region.tbl",{table: "region", fieldsTerminatedBy:"|",
bytesPerChunk:"100M", threads:16})

mysql-js> util.importTable("part.tbl",{table: "part", fieldsTerminatedBy:"|",
bytesPerChunk:"100M", threads:16})

mysql-js> util.importTable("supplier.tbl",{table: "supplier", fieldsTerminatedBy:"|" ,
bytesPerChunk:"100M", threads:16})

mysql-js> util.importTable("partsupp.tbl",{table: "partsupp", fieldsTerminatedBy:"|",
bytesPerChunk:"100M", threads:16})

mysql-js> util.importTable("customer.tbl",{table: "customer", fieldsTerminatedBy:"|",
bytesPerChunk:"100M", threads:16})

mysql-js> util.importTable("orders.tbl",{table: "orders", fieldsTerminatedBy:"|",
bytesPerChunk:"100M", threads:16})

mysql-js> util.importTable("lineitem.tbl",{table: "lineitem", fieldsTerminatedBy:"|",
bytesPerChunk:"100M", threads:16})
```

Load tpch Data Into MySQL HeatWave

To load the `tpch` sample data into the MySQL HeatWave Cluster:

Note

For MySQL HeatWave on AWS, load data into MySQL HeatWave Cluster using the MySQL HeatWave Console. See Manage Data in MySQL HeatWave with Workspaces in the *MySQL HeatWave on AWS Service Guide*.

1. Start MySQL Shell and connect to the DB System endpoint:

```
$> mysqlsh Username@DBSystem_IP_Address_or_Host_Name
```

Note

MySQL Shell opens in `SQL` execution mode by default. If it opens in `JavaScript (JS)` mode, run the following command to switch to `SQL` execution mode.

```
mysql-js> \sql
```

2. Execute the following operations to prepare the `tpch` sample database tables and load them into the MySQL HeatWave Cluster. The operations performed include defining string column encodings, defining the secondary engine, and executing `SECONDARY_LOAD` operations.

```
mysql-sql> USE tpch;

mysql-sql> ALTER TABLE nation modify `N_NAME` CHAR(25) NOT NULL COMMENT 'RAPID_COLUMN=ENCODING=SORTE
mysql-sql> ALTER TABLE nation modify `N_COMMENT` VARCHAR(152) COMMENT 'RAPID_COLUMN=ENCODING=SORTED'
mysql-sql> ALTER TABLE nation SECONDARY_ENGINE=RAPID;
mysql-sql> ALTER TABLE nation SECONDARY_LOAD;

mysql-sql> ALTER TABLE region modify `R_NAME` CHAR(25) NOT NULL COMMENT 'RAPID_COLUMN=ENCODING=SORTE
mysql-sql> ALTER TABLE region modify `R_COMMENT` VARCHAR(152) COMMENT 'RAPID_COLUMN=ENCODING=SORTED'
mysql-sql> ALTER TABLE region SECONDARY_ENGINE=RAPID;
mysql-sql> ALTER TABLE region SECONDARY_LOAD;

mysql-sql> ALTER TABLE part modify `P_MFGR` CHAR(25) NOT NULL COMMENT 'RAPID_COLUMN=ENCODING=SORTED'
mysql-sql> ALTER TABLE part modify `P_BRAND` CHAR(10) NOT NULL COMMENT 'RAPID_COLUMN=ENCODING=SORTED'
mysql-sql> ALTER TABLE part modify `P_CONTAINER` CHAR(10) NOT NULL COMMENT 'RAPID_COLUMN=ENCODING=SO
mysql-sql> ALTER TABLE part modify `P_COMMENT` VARCHAR(23) NOT NULL COMMENT 'RAPID_COLUMN=ENCODING=S
mysql-sql> ALTER TABLE part SECONDARY_ENGINE=RAPID;
mysql-sql> ALTER TABLE part SECONDARY_LOAD;

mysql-sql> ALTER TABLE supplier modify `S_NAME` CHAR(25) NOT NULL COMMENT 'RAPID_COLUMN=ENCODING=SOR
mysql-sql> ALTER TABLE supplier modify `S_ADDRESS` VARCHAR(40) NOT NULL COMMENT 'RAPID_COLUMN=ENCODI
mysql-sql> ALTER TABLE supplier modify `S_PHONE` CHAR(15) NOT NULL COMMENT 'RAPID_COLUMN=ENCODING=SO

mysql-sql> ALTER TABLE supplier SECONDARY_ENGINE=RAPID;
mysql-sql> ALTER TABLE supplier SECONDARY_LOAD;
mysql-sql> ALTER TABLE partsupp modify `PS_COMMENT` VARCHAR(199) NOT NULL COMMENT 'RAPID_COLUMN=ENCO
mysql-sql> ALTER TABLE partsupp SECONDARY_ENGINE=RAPID;
mysql-sql> ALTER TABLE partsupp SECONDARY_LOAD;

mysql-sql> ALTER TABLE customer modify `C_NAME` VARCHAR(25) NOT NULL COMMENT 'RAPID_COLUMN=ENCODING=
mysql-sql> ALTER TABLE customer modify `C_ADDRESS` VARCHAR(40) NOT NULL COMMENT 'RAPID_COLUMN=ENCODI
mysql-sql> ALTER TABLE customer modify `C_MKTSEGMENT` CHAR(10) NOT NULL COMMENT 'RAPID_COLUMN=ENCODI
mysql-sql> ALTER TABLE customer modify `C_COMMENT` VARCHAR(117) NOT NULL COMMENT 'RAPID_COLUMN=ENCOD

mysql-sql> ALTER TABLE customer SECONDARY_ENGINE=RAPID;
mysql-sql> ALTER TABLE customer SECONDARY_LOAD;

mysql-sql> ALTER TABLE orders modify `O_ORDERSTATUS` CHAR(1) NOT NULL COMMENT 'RAPID_COLUMN=ENCODING
mysql-sql> ALTER TABLE orders modify `O_ORDERPRIORITY` CHAR(15) NOT NULL COMMENT 'RAPID_COLUMN=ENCOD
mysql-sql> ALTER TABLE orders modify `O_CLERK` CHAR(15) NOT NULL COMMENT 'RAPID_COLUMN=ENCODING=SORT
mysql-sql> ALTER TABLE orders SECONDARY_ENGINE=RAPID;
mysql-sql> ALTER TABLE orders SECONDARY_LOAD;

mysql-sql> ALTER TABLE lineitem modify `L_RETURNFLAG` CHAR(1) NOT NULL COMMENT 'RAPID_COLUMN=ENCODIN
mysql-sql> ALTER TABLE lineitem modify `L_LINESTATUS` CHAR(1) NOT NULL COMMENT 'RAPID_COLUMN=ENCODIN
mysql-sql> ALTER TABLE lineitem modify `L_SHIPINSTRUCT` CHAR(25) NOT NULL COMMENT 'RAPID_COLUMN=ENCO
mysql-sql> ALTER TABLE lineitem modify `L_SHIPMODE` CHAR(10) NOT NULL COMMENT 'RAPID_COLUMN=ENCODING
mysql-sql> ALTER TABLE lineitem modify `L_COMMENT` VARCHAR(44) NOT NULL COMMENT 'RAPID_COLUMN=ENCODI
mysql-sql> ALTER TABLE lineitem SECONDARY_ENGINE=RAPID;
mysql-sql> ALTER TABLE lineitem SECONDARY_LOAD;
```

3. Verify that the `tpch` sample database tables are loaded in the MySQL HeatWave Cluster by querying `LOAD_STATUS` data from the MySQL HeatWave Performance Schema tables. Loaded tables have an `AVAIL_RPDGSTABSTATE` load status.

```
mysql-sql> USE performance_schema;
mysql-sql> SELECT NAME, LOAD_STATUS
```

```
         FROM rpd_tables,rpd_table_id
         WHERE rpd_tables.ID = rpd_table_id.ID;
+------------------------------+--------------------+
| NAME                         | LOAD_STATUS        |
+------------------------------+--------------------+
| tpch.supplier                | AVAIL_RPDGSTABSTATE |
| tpch.partsupp                | AVAIL_RPDGSTABSTATE |
| tpch.orders                  | AVAIL_RPDGSTABSTATE |
| tpch.lineitem                | AVAIL_RPDGSTABSTATE |
| tpch.customer                | AVAIL_RPDGSTABSTATE |
| tpch.nation                  | AVAIL_RPDGSTABSTATE |
| tpch.region                  | AVAIL_RPDGSTABSTATE |
| tpch.part                    | AVAIL_RPDGSTABSTATE |
+------------------------------+--------------------+
```

Run tpch Queries

This topic describes how to query `tpch` data in the MySQL HeatWave Cluster. After tables are loaded into the MySQL HeatWave Cluster, queries that qualify are automatically offloaded to the MySQL HeatWave Cluster for accelerated processing. To run queries:

Note

For MySQL HeatWave on AWS, run queries from the *Query Editor* in the MySQL HeatWave Console. See Running Queries in the *MySQL HeatWave on AWS Service Guide*.

1. Start MySQL Shell and connect to the DB System endpoint:

```
$> mysqlsh Username@DBSystem_IP_Address_or_Host_Name
```

Note

MySQL Shell opens in `SQL` execution mode by default. If it opens in `JavaScript (JS)` mode, run the following command to switch to `SQL` execution mode.

```
mysql-js> \sql
```

2. Change to the tpch database:

```
mysql-sql> USE tpch;
Default schema set to `tpch`.Fetching table and column names from `tpch` for
auto-completion... Press ^C to stop.
```

3. Before running a query, use `EXPLAIN` to verify that the query can be offloaded to the MySQL HeatWave Cluster. For example:

```
mysql-sql> EXPLAIN SELECT SUM(l_extendedprice * l_discount) AS revenue
                FROM lineitem
                WHERE l_shipdate >= date '1994-01-01';
*************************** 1. row ***************************
           id: 1
  select_type: SIMPLE
        table: lineitem
   partitions: NULL
         type: ALL
possible_keys: NULL
          key: NULL
      key_len: NULL
          ref: NULL
         rows: 56834662
     filtered: 33.33
        Extra: Using where; Using secondary engine RAPID
```

If the query can be offloaded, the `Extra` column in the `EXPLAIN` output reports `Using secondary engine RAPID`.

4. After verifying that the query can be offloaded, run the query and note the execution time.

```
mysql-sql> SELECT SUM(l_extendedprice * l_discount) AS revenue
                FROM lineitem
                WHERE l_shipdate >= date '1994-01-01';
+------------------+
| revenue          |
+------------------+
| 82752894454.9036 |
+------------------+
1 row in set (0.04 sec)
```

5. To compare the MySQL HeatWave Cluster execution time with DB System execution time, disable the `use_secondary_engine` variable to see how long it takes to run the same query on the DB System. For example:

```
mysql-sql> SET SESSION use_secondary_engine=OFF;
mysql-sql> SELECT SUM(l_extendedprice * l_discount) AS revenue
                FROM lineitem
                WHERE l_shipdate >= date '1994-01-01';
+------------------+
| revenue          |
+------------------+
| 82752894454.9036 |
+------------------+
1 row in set (24.20 sec)
```

   For other `tpch` sample database queries that you can run, see Additional tpch Queries. For more information about running queries, refer to Section 5.4, “Run Queries”.

Additional tpch Queries

This topic provides additional tpch queries that you can run to test the MySQL HeatWave Cluster.

- TPCH-Q1: Pricing Summary Report Query

  As described in the TPC Benchmark™ H (TPC-H) specification: "The Pricing Summary Report Query provides a summary pricing report for all lineitems shipped as of a given date. The date is within 60 - 120 days of the greatest ship date contained in the database. The query lists totals for extended price, discounted extended price, discounted extended price plus tax, average quantity, average extended price, and average discount. These aggregates are grouped by `RETURNFLAG` and `LINESTATUS`, and listed in ascending order of `RETURNFLAG` and `LINESTATUS`. A count of the number of lineitems in each group is included."

```
mysql-sql> SELECT
          l_returnflag,
          l_linestatus,
          SUM(l_quantity) AS sum_qty,
          SUM(l_extendedprice) AS sum_base_price,
          SUM(l_extendedprice * (1 - l_discount)) AS sum_disc_price,
          SUM(l_extendedprice * (1 - l_discount) * (1 + l_tax)) AS sum_charge,
          AVG(l_quantity) AS avg_qty,
          AVG(l_extendedprice) AS avg_price,
          AVG(l_discount) AS avg_disc,
          COUNT(*) AS count_order
      FROM
          lineitem
      WHERE
          l_shipdate <= DATE '1998-12-01' - INTERVAL '90' DAY
      GROUP BY l_returnflag , l_linestatus
      ORDER BY l_returnflag , l_linestatus;
```

- TPCH-Q3: Shipping Priority Query

  As described in the TPC Benchmark™ H (TPC-H) specification: "The Shipping Priority Query retrieves the shipping priority and potential revenue, defined as the sum of `l_extendedprice * (1-l_discount)`, of the orders having the largest revenue among those that had not been shipped

as of a given date. Orders are listed in decreasing order of revenue. If more than 10 unshipped orders exist, only the 10 orders with the largest revenue are listed."

```
mysql-sql> SELECT
          l_orderkey,
          SUM(l_extendedprice * (1 - l_discount)) AS revenue,
          o_orderdate,
          o_shippriority
      FROM
          customer,
          orders,
          lineitem
      WHERE
          c_mktsegment = 'BUILDING'
              AND c_custkey = o_custkey
              AND l_orderkey = o_orderkey
              AND o_orderdate < DATE '1995-03-15'
              AND l_shipdate > DATE '1995-03-15'
      GROUP BY l_orderkey , o_orderdate , o_shippriority
      ORDER BY revenue DESC , o_orderdate
      LIMIT 10;
```

- TPCH-Q9: Product Type Profit Measure Query

  As described in the TPC Benchmark™ H (TPC-H) specification: "The Product Type Profit Measure Query finds, for each nation and each year, the profit for all parts ordered in that year that contain a specified substring in their names and that were filled by a supplier in that nation. The profit is defined as the sum of `[(l_extendedprice*(1-l_discount)) - (ps_supplycost * l_quantity)]` for all lineitems describing parts in the specified line. The query lists the nations in ascending alphabetical order and, for each nation, the year and profit in descending order by year (most recent first). "

```
mysql-sql> SELECT
          nation, o_year, SUM(amount) AS sum_profit
      FROM
          (SELECT
              n_name AS nation,
                  YEAR(o_ORDERdate) AS o_year,
                  l_extendedprice * (1 - l_discount) - ps_supplycost * l_quantity AS amount
          FROM
              part
          STRAIGHT_JOIN partsupp
          STRAIGHT_JOIN lineitem
          STRAIGHT_JOIN supplier
          STRAIGHT_JOIN orders
          STRAIGHT_JOIN nation
          WHERE
              s_suppkey = l_suppkey
                  AND ps_suppkey = l_suppkey
                  AND ps_partkey = l_partkey
                  AND p_partkey = l_partkey
                  AND o_ORDERkey = l_ORDERkey
                  AND s_nationkey = n_nationkey
                  AND p_name LIKE '%green%') AS profit
      GROUP BY nation , o_year
      ORDER BY nation , o_year DESC;
```

Unload tpch Tables

`tpch` tables can be unloaded from MySQL HeatWave Cluster using the following statements:

Note

For MySQL HeatWave on AWS, unload data from MySQL HeatWave Cluster using the MySQL HeatWave Console. See Manage Data in MySQL HeatWave with Workspaces in the *MySQL HeatWave on AWS Service Guide*.

```
mysql-sql> USE tpch;
```

```
mysql-sql> ALTER TABLE customer SECONDARY_UNLOAD;
mysql-sql> ALTER TABLE lineitem SECONDARY_UNLOAD;
mysql-sql> ALTER TABLE nation SECONDARY_UNLOAD;
mysql-sql> ALTER TABLE orders SECONDARY_UNLOAD;
mysql-sql> ALTER TABLE part SECONDARY_UNLOAD;
mysql-sql> ALTER TABLE partsupp SECONDARY_UNLOAD;
mysql-sql> ALTER TABLE region SECONDARY_UNLOAD;
mysql-sql> ALTER TABLE supplier SECONDARY_UNLOAD;
```

3.2 Quickstart: AirportDB Analytics

The *MySQL HeatWave airportdb Quickstart* shows how to import data into the DB System using the MySQL Shell Dump Load utility, loading data into MySQL HeatWave Cluster using Auto Parallel Load, and running queries.

For an online workshop that demonstrates MySQL HeatWave using the `airportdb` sample database, see Turbocharge Business Insights with MySQL HeatWave Service and MySQL HeatWave.

This quickstart contains the following sections:

- AirportDB Requirements
- Download AirportDB Data
- Load AirportDB into DB system
- Load AirportDB into MySQL HeatWave Cluster
- Run AirportDB Queries
- Additional AirportDB Queries
- Unload AirportDB Tables

AirportDB Requirements

- Review the requirements to setup and use MySQL HeatWave.
- Confirm you have the latest version of MySQL Shell installed. To learn more about downloading and installing MySQL Shell, see Installing MySQL Shell.
- You will need to connect to the DB System in one of the following ways:
  - Through a VPN
  - Through a compute instance

  To learn more about connecting to your DB system, see Networking.

Download AirportDB Data

Download the `airportdb` sample database. The `airportdb` sample database is provided for download as a compressed `tar` or Zip archive. The download is approximately 640 MBs in size.

To run these commands, make sure you have the `wget` utility installed. Alternatively, you can download the files by pasting the URL in a browser window.

```
$> wget https://downloads.mysql.com/docs/airport-db.tar.gz
```

or

```
$> wget https://downloads.mysql.com/docs/airport-db.zip
```

Load AirportDB into DB system

Depending on how you connect to your DB system, follow the appropriate steps to load the `airportdb` database into the DB system. To learn more about connecting to your DB system, see Networking.

To load the data, you will need to use the MySQL Shell Dump Loading utility. See MySQL Shell Dump Loading Utility.

VPN Connection

If you connect to the DB system through a VPN connection, you can load `airportdb` locally from your computer.

To load the `airportdb` database through a VPN connection:

1. Unpack the compressed `tar` or Zip archive you downloaded previously. It creates a single directory named `airport-db`, which contains the data files.

   ```
   $> tar xvzf airport-db.tar.gz
   ```

   or

   ```
   $> unzip airport-db.zip
   ```

2. Copy the file path to the unpacked `airport-db` folder. For example, on Windows it might be `C:\\Temp\\airport-db`, or on Mac it might be `/Users/johnsmith/airport-db`.

3. From a terminal window, start MySQL Shell and connect to the DB System IP address. You can review the IP address (private IP address) and other connection details in the *Connections* tab for the DB system. For additional information about connecting to a DB System, see Connecting to a DB System in the *MySQL HeatWave on OCI Service Guide*.

   ```
   $> mysqlsh Username@DBSystem_IP_Address_or_Host_Name
   ```

4. Enable `js` mode.

   ```
   mysql-sql> \js
   ```

5. Load the `airportdb` database into the DB system using MySQL Shell Dump Loading Utility. In the command, replace `airport-db` with the file path you copied in the previous step.

   ```
   mysql-js> util.loadDump("file path to the unpacked airport-db folder", {threads: 16, deferTableIndexes:
   ```

   The command uses the following options:

   - `threads`: Specifies the number of parallel threads to use to upload chunks of data to the target MySQL instance.
   - `deferTableIndexes`: This option is set to "all", so it defers the creation of secondary indexes until after the table data is loaded. This can reduce loading time. If you intend to use `airportdb` with only MySQL HeatWave, which does not use secondary indexes, you can avoid creating secondary indexes by specifying the `loadIndexes: "FALSE"` option instead of `deferTableIndexes: "all"`. For more information about MySQL Dump Load options, see Dump Loading Utility.
   - `ignoreVersion`: This option is enabled, which imports the dump even if the major version number of the MySQL instance from which the data was dumped is non-consecutive to the major version number of the MySQL instance to which the data will be uploaded.

Compute Instance Connection

If you connect to the DB system through a compute instance, you first need to upload the `airportdb` database to the compute instance before running the MySQL Shell Dump Loading Utility.

To load the `airportdb` database through a compute instance connection:

1. In the terminal window, go to the local directory containing the `airportdb` files.
2. Upload the contents of the `airport-db` folder to the appropriate directory of the compute instance.

   ```
   $> scp -v -i ssh-key.key airport-db.zip opc@ComputeInstancePublicIP:/home/opc/
   ```

   Replace the following:

   - *`ssh-key.key`*: The full file path to the SSH key file (.key) for the compute instance.
   - *`airport-db.zip`*: The full file path to the `airport-db.zip` file on your device.
   - *`opc@ComputeInstancePublicIP`*: The appropriate username and public IP for the compute instance.
   - *`/home/opc/`*: The appropriate file path to upload the airport-db folder to in the compute instance.
3. Once the upload successfully completes, SSH into the Compute instance using the public IP address of the Compute instance.

   ```
   $> ssh -i ssh-key.key opc@computeInstancePublicIP
   ```

   Replace the following:

   - *`ssh-key.key`*: The full file path to the SSH key file (.key) for the compute instance.
   - *`opc@ComputeInstancePublicIP`*: The appropriate username and public IP for the compute instance.
4. Unpack the compressed `tar` or Zip archive you downloaded previously. It creates a single directory named `airport-db`, which contains the data files.

   ```
   $> tar xvzf airport-db.tar.gz
   ```

   or

   ```
   $> unzip airport-db.zip
   ```
5. From a terminal window, start MySQL Shell and connect to the DB System IP address. You can review the IP address (private IP address) and other connection details in the *Connections* tab for the DB system. For additional information about connecting to a DB System, see Connecting to a DB System in the *MySQL HeatWave on OCI Service Guide*.

   ```
   $> mysqlsh Username@DBSystem_IP_Address_or_Host_Name
   ```
6. Enable `js` mode.

   ```
   mysql-sql> \js
   ```
7. Load the `airportdb` database into the DB system using MySQL Shell Dump Loading Utility. If needed, replace `airport-db` to the appropriate file path in the compute instance.

   ```
   mysql-js> util.loadDump("airport-db", {threads: 16, deferTableIndexes: "all", ignoreVersion: true})
   ```

   The command uses the following options:

   - `threads`: Specifies the number of parallel threads to use to upload chunks of data to the target MySQL instance.

- `deferTableIndexes`: This option is set to "all", so it defers the creation of secondary indexes until after the table data is loaded. This can reduce loading time. If you intend to use `airportdb` with only MySQL HeatWave, which does not use secondary indexes, you can avoid creating secondary indexes by specifying the `loadIndexes: "FALSE"` option instead of `deferTableIndexes: "all"`. For more information about MySQL Dump Load options, see Dump Loading Utility.
- `ignoreVersion`: This option is enabled, which imports the dump even if the major version number of the MySQL instance from which the data was dumped is non-consecutive to the major version number of the MySQL instance to which the data will be uploaded.

Load AirportDB into MySQL HeatWave Cluster

To load the `airportdb` from the DB System into MySQL HeatWave Cluster:

Note

For MySQL HeatWave on AWS, load data into MySQL HeatWave Cluster using the MySQL HeatWave Console. See Manage Data in MySQL HeatWave with Workspaces in the *MySQL HeatWave on AWS Service Guide*.

1. From a terminal window, start MySQL Shell and connect to the DB System IP address. You can review the IP address (private IP address) and other connection details in the *Connections* tab for the DB system.

```
$> mysqlsh Username@DBSystem_IP_Address_or_Host_Name
```

2. Run the following Auto Parallel Load command to load the `airportdb` tables into MySQL HeatWave Cluster.

```
mysql-sql> CALL sys.heatwave_load(JSON_ARRAY('airportdb'), NULL);
```

For information about the Auto Parallel Load utility, see Section 4.2.5, “Load Data Using Auto Parallel Load”.

Run AirportDB Queries

After airportdb sample database tables are loaded into the MySQL HeatWave Cluster, queries that qualify are automatically offloaded to the MySQL HeatWave Cluster for accelerated processing. To run queries:

Note

For MySQL HeatWave on AWS, run queries from the *Query Editor* in the MySQL HeatWave Console. See Running Queries in the *MySQL HeatWave on AWS Service Guide*.

1. From a terminal window, start MySQL Shell and connect to the DB System IP address. You can review the IP address (private IP address) and other connection details in the *Connections* tab for the DB system.

```
$> mysqlsh Username@DBSystem_IP_Address_or_Host_Name
```

2. Change to the `airportdb` database.

```
mysql-sql> USE airportdb;
Default schema set to `airportdb`.Fetching table and column names from `airportdb` for
auto-completion... Press ^C to stop.
```

3. Before running a query, use `EXPLAIN` to verify that the query can be offloaded to the MySQL HeatWave Cluster. For example:

```
mysql-sql> EXPLAIN SELECT booking.price, count(*)
```

```
                FROM booking
                WHERE booking.price > 500
                GROUP BY booking.price
                ORDER BY booking.price LIMIT
                10\G
*************************** 1. row ***************************
           id: 1
  select_type: SIMPLE
        table: booking
   partitions: NULL
         type: ALL
possible_keys: NULL
          key: NULL
      key_len: NULL
          ref: NULL
         rows: 54081693
     filtered: 33.32999801635742
        Extra: Using where; Using temporary; Using filesort; Using secondary engine RAPID
```

If the query can be offloaded, the `Extra` column in the `EXPLAIN` output reports `Using secondary engine RAPID`.

4. After verifying that the query can be offloaded, run the query and note the execution time.

```
mysql-sql> SELECT booking.price, count(*)
                FROM booking
                WHERE booking.price > 500
                GROUP BY booking.price
                ORDER BY booking.price
                LIMIT 10;
+--------+----------+
| price  | count(*) |
+--------+----------+
| 500.01 |      860 |
| 500.02 |     1207 |
| 500.03 |     1135 |
| 500.04 |     1010 |
| 500.05 |     1016 |
| 500.06 |     1039 |
| 500.07 |     1002 |
| 500.08 |     1095 |
| 500.09 |     1117 |
| 500.10 |     1106 |
+--------+----------+
10 rows in set (0.0537 sec)
```

5. To compare the MySQL HeatWave Cluster execution time with DB System execution time, disable the `use_secondary_engine` variable to see how long it takes to run the same query on the DB System; for example:

```
mysql-sql> SET SESSION use_secondary_engine=OFF;
mysql-sql> SELECT booking.price, count(*)
                FROM booking
                WHERE booking.price > 500
                GROUP BY booking.price
                ORDER BY booking.price
                LIMIT 10;
+--------+----------+
| price  | count(*) |
+--------+----------+
| 500.01 |      860 |
| 500.02 |     1207 |
| 500.03 |     1135 |
| 500.04 |     1010 |
| 500.05 |     1016 |
| 500.06 |     1039 |
| 500.07 |     1002 |
| 500.08 |     1095 |
| 500.09 |     1117 |
| 500.10 |     1106 |
+--------+----------+
```

```
10 rows in set (9.3859 sec)
```

For other `airportdb` sample database queries that you can run, see Additional AirportDB Queries. For more information about running queries, see Section 5.4, “Run Queries”.

Additional AirportDB Queries

This topic provides additional airportdb queries that you can run to test the MySQL HeatWave Cluster.

- Query 1: Number of Tickets > $500.00, Grouped By Price

```
mysql-sql> SELECT booking.price, count(*)
      FROM booking
      WHERE booking.price > 500
      GROUP BY booking.price
      ORDER BY booking.price
      LIMIT 10;
```

- Query 2: Average Age of Passengers By Country, Per Airline

```
mysql-sql> SELECT airline.airlinename, AVG(datediff(departure,birthdate)/365.25) as avg_age, count(*) as
      FROM booking, flight, airline, passengerdetails
      WHERE booking.flight_id=flight.flight_id
      AND airline.airline_id=flight.airline_id
      AND booking.passenger_id=passengerdetails.passenger_id
      AND country IN ("SWITZERLAND", "FRANCE", "ITALY")
      GROUP BY airline.airlinename
      ORDER BY airline.airlinename, avg_age
      LIMIT 10;
```

- Query 3: Most Tickets Sales by Airline for Departures from US Airports

```
mysql-sql> SELECT airline.airlinename, SUM(booking.price) as price_tickets, count(*) as nb_tickets
      FROM booking, flight, airline, airport_geo
      WHERE booking.flight_id=flight.flight_id
      AND airline.airline_id=flight.airline_id
      AND flight.from=airport_geo.airport_id
      AND airport_geo.country = "UNITED STATES"
      GROUP BY airline.airlinename
      ORDER BY nb_tickets desc, airline.airlinename
      LIMIT 10;
```

Unload AirportDB Tables

`airportdb` tables can be unloaded from MySQL HeatWave Cluster using the following statements:

Note

For MySQL HeatWave on AWS, unload data from MySQL HeatWave Cluster using the MySQL HeatWave Console. See Manage Data in MySQL HeatWave with Workspaces in the *MySQL HeatWave on AWS Service Guide*.

```
mysql-sql> USE airportdb;

ALTER TABLE booking SECONDARY_UNLOAD;
ALTER TABLE flight SECONDARY_UNLOAD;
ALTER TABLE flight_log SECONDARY_UNLOAD;
ALTER TABLE airport SECONDARY_UNLOAD;
ALTER TABLE airport_reachable SECONDARY_UNLOAD;
ALTER TABLE airport_geo SECONDARY_UNLOAD;
ALTER TABLE airline SECONDARY_UNLOAD;
ALTER TABLE flightschedule SECONDARY_UNLOAD;
ALTER TABLE airplane SECONDARY_UNLOAD;
ALTER TABLE airplane_type SECONDARY_UNLOAD;
ALTER TABLE employee SECONDARY_UNLOAD;
ALTER TABLE passenger SECONDARY_UNLOAD;
ALTER TABLE passengerdetails SECONDARY_UNLOAD;
ALTER TABLE weatherdata SECONDARY_UNLOAD;
```

3.3 Quickstart: Create and Use a Machine Learning Model

This quickstart illustrates an end-to-end example of creating and using a predictive machine learning model using MySQL HeatWave AutoML. It steps through preparing data, using the `ML_TRAIN` routine to train a model, and using `ML_PREDICT_*` and `ML_EXPLAIN_*` routines to generate predictions and explanations. The tutorial also demonstrates how to assess the quality of a model using the `ML_SCORE` routine, and how to view a model explanation to understand how the model works.

For an online workshop based on this tutorial, see Get started with MySQL MySQL HeatWave AutoML.

The tutorial uses the publicly available Iris Data Set from the UCI Machine Learning Repository.

Dua, D. and Graff, C. (2019). UCI Machine Learning Repository [http://archive.ics.uci.edu/ml]. Irvine, CA: University of California, School of Information.

This quickstart contains the following sections:

- Overview
- Before You Begin
- Prepare Datasets
- Train Model
- Load the Model
- Make Predictions
- Generating Explanations
- Score the Model
- Unloading the Model

Overview

The *Iris Data Set* has the following data, where the sepal and petal features are used to predict the `class` label, which is the type of Iris plant:

- sepal length (cm)
- sepal width (cm)
- petal length (cm)
- petal width (cm)
- class. Possible values include:
  - Iris Setosa
  - Iris Versicolour
  - Iris Virginica

Data is stored in the MySQL database in the following schema and tables:

- `ml_data` schema: The schema containing training and test dataset tables.
- `iris_train` table: The training dataset (labeled). Includes feature columns (sepal length, sepal width, petal length, petal width) and a populated `class` target column with ground truth values.
- `iris_test` table: The test dataset (unlabeled). Includes feature columns (sepal length, sepal width, petal length, petal width) but no target column.

- `iris_validate` table: The validation dataset (labeled). Includes feature columns (sepal length, sepal width, petal length, petal width) and a populated `class` target column with ground truth values.

Before You Begin

Review the MySQL HeatWave Quickstart Requirements.

Prepare Datasets

1. Create and use the the database `ml_data` to store the table of data.

```
mysql> CREATE SCHEMA ml_data;
mysql> USE ml_data;
```

2. Create the table of training data.

```
mysql> CREATE TABLE `iris_train` (
          `sepal length` float DEFAULT NULL,
          `sepal width` float DEFAULT NULL,
          `petal length` float DEFAULT NULL,
          `petal width` float DEFAULT NULL,
          `class` varchar(16) DEFAULT NULL);
```

3. Insert the training data into the table. Copy and paste the following commands.

```
INSERT INTO iris_train VALUES(6.4,2.8,5.6,2.2,'Iris-virginica');
INSERT INTO iris_train VALUES(5.0,2.3,3.3,1.0,'Iris-setosa');
INSERT INTO iris_train VALUES(4.9,2.5,4.5,1.7,'Iris-virginica');
INSERT INTO iris_train VALUES(4.9,3.1,1.5,0.1,'Iris-versicolor');
INSERT INTO iris_train VALUES(5.7,3.8,1.7,0.3,'Iris-versicolor');
INSERT INTO iris_train VALUES(4.4,3.2,1.3,0.2,'Iris-versicolor');
INSERT INTO iris_train VALUES(5.4,3.4,1.5,0.4,'Iris-versicolor');
INSERT INTO iris_train VALUES(6.9,3.1,5.1,2.3,'Iris-virginica');
INSERT INTO iris_train VALUES(6.7,3.1,4.4,1.4,'Iris-setosa');
INSERT INTO iris_train VALUES(5.1,3.7,1.5,0.4,'Iris-versicolor');
INSERT INTO iris_train VALUES(5.2,2.7,3.9,1.4,'Iris-setosa');
INSERT INTO iris_train VALUES(6.9,3.1,4.9,1.5,'Iris-setosa');
INSERT INTO iris_train VALUES(5.8,4.0,1.2,0.2,'Iris-versicolor');
INSERT INTO iris_train VALUES(5.4,3.9,1.7,0.4,'Iris-versicolor');
INSERT INTO iris_train VALUES(7.7,3.8,6.7,2.2,'Iris-virginica');
INSERT INTO iris_train VALUES(6.3,3.3,4.7,1.6,'Iris-setosa');
INSERT INTO iris_train VALUES(6.8,3.2,5.9,2.3,'Iris-virginica');
INSERT INTO iris_train VALUES(7.6,3.0,6.6,2.1,'Iris-virginica');
INSERT INTO iris_train VALUES(6.4,3.2,5.3,2.3,'Iris-virginica');
INSERT INTO iris_train VALUES(5.7,4.4,1.5,0.4,'Iris-versicolor');
INSERT INTO iris_train VALUES(6.7,3.3,5.7,2.1,'Iris-virginica');
INSERT INTO iris_train VALUES(6.4,2.8,5.6,2.1,'Iris-virginica');
INSERT INTO iris_train VALUES(5.4,3.9,1.3,0.4,'Iris-versicolor');
INSERT INTO iris_train VALUES(6.1,2.6,5.6,1.4,'Iris-virginica');
INSERT INTO iris_train VALUES(7.2,3.0,5.8,1.6,'Iris-virginica');
INSERT INTO iris_train VALUES(5.2,3.5,1.5,0.2,'Iris-versicolor');
INSERT INTO iris_train VALUES(5.8,2.6,4.0,1.2,'Iris-setosa');
INSERT INTO iris_train VALUES(5.9,3.0,5.1,1.8,'Iris-virginica');
INSERT INTO iris_train VALUES(5.4,3.0,4.5,1.5,'Iris-setosa');
INSERT INTO iris_train VALUES(6.7,3.0,5.0,1.7,'Iris-setosa');
INSERT INTO iris_train VALUES(6.3,2.3,4.4,1.3,'Iris-setosa');
INSERT INTO iris_train VALUES(5.1,2.5,3.0,1.1,'Iris-setosa');
INSERT INTO iris_train VALUES(6.4,3.2,4.5,1.5,'Iris-setosa');
INSERT INTO iris_train VALUES(6.8,3.0,5.5,2.1,'Iris-virginica');
INSERT INTO iris_train VALUES(6.2,2.8,4.8,1.8,'Iris-virginica');
INSERT INTO iris_train VALUES(6.9,3.2,5.7,2.3,'Iris-virginica');
INSERT INTO iris_train VALUES(6.5,3.2,5.1,2.0,'Iris-virginica');
INSERT INTO iris_train VALUES(5.8,2.8,5.1,2.4,'Iris-virginica');
INSERT INTO iris_train VALUES(5.1,3.8,1.5,0.3,'Iris-versicolor');
INSERT INTO iris_train VALUES(4.8,3.0,1.4,0.3,'Iris-versicolor');
INSERT INTO iris_train VALUES(7.9,3.8,6.4,2.0,'Iris-virginica');
INSERT INTO iris_train VALUES(5.8,2.7,5.1,1.9,'Iris-virginica');
INSERT INTO iris_train VALUES(6.7,3.0,5.2,2.3,'Iris-virginica');
INSERT INTO iris_train VALUES(5.1,3.8,1.9,0.4,'Iris-versicolor');
INSERT INTO iris_train VALUES(4.7,3.2,1.6,0.2,'Iris-versicolor');
```

```
INSERT INTO iris_train VALUES(6.0,2.2,5.0,1.5,'Iris-virginica');
INSERT INTO iris_train VALUES(4.8,3.4,1.6,0.2,'Iris-versicolor');
INSERT INTO iris_train VALUES(7.7,2.6,6.9,2.3,'Iris-virginica');
INSERT INTO iris_train VALUES(4.6,3.6,1.0,0.2,'Iris-versicolor');
INSERT INTO iris_train VALUES(7.2,3.2,6.0,1.8,'Iris-virginica');
INSERT INTO iris_train VALUES(5.0,3.3,1.4,0.2,'Iris-versicolor');
INSERT INTO iris_train VALUES(6.6,3.0,4.4,1.4,'Iris-setosa');
INSERT INTO iris_train VALUES(6.1,2.8,4.0,1.3,'Iris-setosa');
INSERT INTO iris_train VALUES(5.0,3.2,1.2,0.2,'Iris-versicolor');
INSERT INTO iris_train VALUES(7.0,3.2,4.7,1.4,'Iris-setosa');
INSERT INTO iris_train VALUES(6.0,3.0,4.8,1.8,'Iris-virginica');
INSERT INTO iris_train VALUES(7.4,2.8,6.1,1.9,'Iris-virginica');
INSERT INTO iris_train VALUES(5.8,2.7,5.1,1.9,'Iris-virginica');
INSERT INTO iris_train VALUES(6.2,3.4,5.4,2.3,'Iris-virginica');
INSERT INTO iris_train VALUES(5.0,2.0,3.5,1.0,'Iris-setosa');
INSERT INTO iris_train VALUES(5.6,2.5,3.9,1.1,'Iris-setosa');
INSERT INTO iris_train VALUES(6.7,3.1,5.6,2.4,'Iris-virginica');
INSERT INTO iris_train VALUES(6.3,2.5,5.0,1.9,'Iris-virginica');
INSERT INTO iris_train VALUES(6.4,3.1,5.5,1.8,'Iris-virginica');
INSERT INTO iris_train VALUES(6.2,2.2,4.5,1.5,'Iris-setosa');
INSERT INTO iris_train VALUES(7.3,2.9,6.3,1.8,'Iris-virginica');
INSERT INTO iris_train VALUES(4.4,3.0,1.3,0.2,'Iris-versicolor');
INSERT INTO iris_train VALUES(7.2,3.6,6.1,2.5,'Iris-virginica');
INSERT INTO iris_train VALUES(6.5,3.0,5.5,1.8,'Iris-virginica');
INSERT INTO iris_train VALUES(5.0,3.4,1.5,0.2,'Iris-versicolor');
INSERT INTO iris_train VALUES(4.7,3.2,1.3,0.2,'Iris-versicolor');
INSERT INTO iris_train VALUES(6.6,2.9,4.6,1.3,'Iris-setosa');
INSERT INTO iris_train VALUES(5.5,3.5,1.3,0.2,'Iris-versicolor');
INSERT INTO iris_train VALUES(7.7,3.0,6.1,2.3,'Iris-virginica');
INSERT INTO iris_train VALUES(6.1,3.0,4.9,1.8,'Iris-virginica');
INSERT INTO iris_train VALUES(4.9,3.1,1.5,0.1,'Iris-versicolor');
INSERT INTO iris_train VALUES(5.5,2.4,3.8,1.1,'Iris-setosa');
INSERT INTO iris_train VALUES(5.7,2.9,4.2,1.3,'Iris-setosa');
INSERT INTO iris_train VALUES(6.0,2.9,4.5,1.5,'Iris-setosa');
INSERT INTO iris_train VALUES(6.4,2.7,5.3,1.9,'Iris-virginica');
INSERT INTO iris_train VALUES(5.4,3.7,1.5,0.2,'Iris-versicolor');
INSERT INTO iris_train VALUES(6.1,2.9,4.7,1.4,'Iris-setosa');
INSERT INTO iris_train VALUES(6.5,2.8,4.6,1.5,'Iris-setosa');
INSERT INTO iris_train VALUES(5.6,2.7,4.2,1.3,'Iris-setosa');
INSERT INTO iris_train VALUES(6.3,3.4,5.6,2.4,'Iris-virginica');
INSERT INTO iris_train VALUES(4.9,3.1,1.5,0.1,'Iris-versicolor');
INSERT INTO iris_train VALUES(6.8,2.8,4.8,1.4,'Iris-setosa');
INSERT INTO iris_train VALUES(5.7,2.8,4.5,1.3,'Iris-setosa');
INSERT INTO iris_train VALUES(6.0,2.7,5.1,1.6,'Iris-setosa');
INSERT INTO iris_train VALUES(5.0,3.5,1.3,0.3,'Iris-versicolor');
INSERT INTO iris_train VALUES(6.5,3.0,5.2,2.0,'Iris-virginica');
INSERT INTO iris_train VALUES(6.1,2.8,4.7,1.2,'Iris-setosa');
INSERT INTO iris_train VALUES(5.1,3.5,1.4,0.3,'Iris-versicolor');
INSERT INTO iris_train VALUES(4.6,3.1,1.5,0.2,'Iris-versicolor');
INSERT INTO iris_train VALUES(6.5,3.0,5.8,2.2,'Iris-virginica');
INSERT INTO iris_train VALUES(4.6,3.4,1.4,0.3,'Iris-versicolor');
INSERT INTO iris_train VALUES(4.6,3.2,1.4,0.2,'Iris-versicolor');
INSERT INTO iris_train VALUES(7.7,2.8,6.7,2.0,'Iris-virginica');
INSERT INTO iris_train VALUES(5.9,3.2,4.8,1.8,'Iris-setosa');
INSERT INTO iris_train VALUES(5.1,3.8,1.6,0.2,'Iris-versicolor');
INSERT INTO iris_train VALUES(4.9,3.0,1.4,0.2,'Iris-versicolor');
INSERT INTO iris_train VALUES(4.9,2.4,3.3,1.0,'Iris-setosa');
INSERT INTO iris_train VALUES(4.5,2.3,1.3,0.3,'Iris-versicolor');
INSERT INTO iris_train VALUES(5.8,2.7,4.1,1.0,'Iris-setosa');
INSERT INTO iris_train VALUES(5.0,3.4,1.6,0.4,'Iris-versicolor');
INSERT INTO iris_train VALUES(5.2,3.4,1.4,0.2,'Iris-versicolor');
INSERT INTO iris_train VALUES(5.3,3.7,1.5,0.2,'Iris-versicolor');
INSERT INTO iris_train VALUES(5.0,3.6,1.4,0.2,'Iris-versicolor');
INSERT INTO iris_train VALUES(5.6,2.9,3.6,1.3,'Iris-setosa');
INSERT INTO iris_train VALUES(4.8,3.1,1.6,0.2,'Iris-versicolor');
INSERT INTO iris_train VALUES(6.3,2.7,4.9,1.8,'Iris-virginica');
INSERT INTO iris_train VALUES(5.7,2.8,4.1,1.3,'Iris-setosa');
INSERT INTO iris_train VALUES(5.0,3.0,1.6,0.2,'Iris-versicolor');
INSERT INTO iris_train VALUES(6.3,3.3,6.0,2.5,'Iris-virginica');
INSERT INTO iris_train VALUES(5.0,3.5,1.6,0.6,'Iris-versicolor');
INSERT INTO iris_train VALUES(5.5,2.6,4.4,1.2,'Iris-setosa');
INSERT INTO iris_train VALUES(5.7,3.0,4.2,1.2,'Iris-setosa');
```

```
INSERT INTO iris_train VALUES(4.4,2.9,1.4,0.2,'Iris-versicolor');
INSERT INTO iris_train VALUES(4.8,3.0,1.4,0.1,'Iris-versicolor');
INSERT INTO iris_train VALUES(5.5,2.4,3.7,1.0,'Iris-setosa');
```

4. Create the table of test data, which has the same columns as the table of training data.

```
mysql> CREATE TABLE `iris_test` LIKE `iris_train`;
```

5. Insert the test data into the table. Copy and paste the following commands.

```
INSERT INTO iris_test VALUES(5.9,3.0,4.2,1.5,'Iris-setosa');
INSERT INTO iris_test VALUES(6.9,3.1,5.4,2.1,'Iris-virginica');
INSERT INTO iris_test VALUES(5.1,3.3,1.7,0.5,'Iris-versicolor');
INSERT INTO iris_test VALUES(6.0,3.4,4.5,1.6,'Iris-setosa');
INSERT INTO iris_test VALUES(5.5,2.5,4.0,1.3,'Iris-setosa');
INSERT INTO iris_test VALUES(6.2,2.9,4.3,1.3,'Iris-setosa');
INSERT INTO iris_test VALUES(5.5,4.2,1.4,0.2,'Iris-versicolor');
INSERT INTO iris_test VALUES(6.3,2.8,5.1,1.5,'Iris-virginica');
INSERT INTO iris_test VALUES(5.6,3.0,4.1,1.3,'Iris-setosa');
INSERT INTO iris_test VALUES(6.7,2.5,5.8,1.8,'Iris-virginica');
INSERT INTO iris_test VALUES(7.1,3.0,5.9,2.1,'Iris-virginica');
INSERT INTO iris_test VALUES(4.3,3.0,1.1,0.1,'Iris-versicolor');
INSERT INTO iris_test VALUES(5.6,2.8,4.9,2.0,'Iris-virginica');
INSERT INTO iris_test VALUES(5.5,2.3,4.0,1.3,'Iris-setosa');
INSERT INTO iris_test VALUES(6.0,2.2,4.0,1.0,'Iris-setosa');
INSERT INTO iris_test VALUES(5.1,3.5,1.4,0.2,'Iris-versicolor');
INSERT INTO iris_test VALUES(5.7,2.6,3.5,1.0,'Iris-setosa');
INSERT INTO iris_test VALUES(4.8,3.4,1.9,0.2,'Iris-versicolor');
INSERT INTO iris_test VALUES(5.1,3.4,1.5,0.2,'Iris-versicolor');
INSERT INTO iris_test VALUES(5.7,2.5,5.0,2.0,'Iris-virginica');
INSERT INTO iris_test VALUES(5.4,3.4,1.7,0.2,'Iris-versicolor');
INSERT INTO iris_test VALUES(5.6,3.0,4.5,1.5,'Iris-setosa');
INSERT INTO iris_test VALUES(6.3,2.9,5.6,1.8,'Iris-virginica');
INSERT INTO iris_test VALUES(6.3,2.5,4.9,1.5,'Iris-setosa');
INSERT INTO iris_test VALUES(5.8,2.7,3.9,1.2,'Iris-setosa');
INSERT INTO iris_test VALUES(6.1,3.0,4.6,1.4,'Iris-setosa');
INSERT INTO iris_test VALUES(5.2,4.1,1.5,0.1,'Iris-versicolor');
INSERT INTO iris_test VALUES(6.7,3.1,4.7,1.5,'Iris-setosa');
INSERT INTO iris_test VALUES(6.7,3.3,5.7,2.5,'Iris-virginica');
INSERT INTO iris_test VALUES(6.4,2.9,4.3,1.3,'Iris-setosa');
```

6. Create the table of validation data. It has the same columns and data as the test data.

```
mysql> CREATE TABLE `iris_validate` LIKE `iris_test`;
mysql> INSERT INTO `iris_validate` SELECT * FROM `iris_test`;
```

Train Model

Train the model with the `ML_TRAIN` routine and use the `iris_train` table previously created. Before training the model, it is good practice to define the model handle instead of automatically creating one. See Define Model Handle.

1. Optionally, set the value of the session variable, which sets the model handle to this same value.

```
mysql> SET @model='iris_model';
```

   The model handle is set to `iris_model`.

2. Run the `ML_TRAIN` routine.

```
mysql> CALL sys.ML_TRAIN('ml_data.iris_train', 'class', JSON_OBJECT('task', 'classification'), @model);
```

3. When the training operation finishes, the model handle is assigned to the `@model` session variable, and the model is stored in the model catalog. View the entry in the model catalog with the following query. Replace *`user1`* with your MySQL account name.

```
mysql> SELECT model_id, model_handle, train_table_name FROM ML_SCHEMA_user1.MODEL_CATALOG WHERE model_h
+----------+--------------------------------------------+--------------------+
| model_id | model_handle                               | train_table_name   |
```

```
+---------+--------------------------------------------+--------------------+
|       1 | iris_model                                 | ml_data.iris_train |
+---------+--------------------------------------------+--------------------+
```

Load the Model

Load the model into MySQL HeatWave AutoML using `ML_MODEL_LOAD` routine:

```
mysql> CALL sys.ML_MODEL_LOAD(@model, NULL);
```

A model must be loaded before you can use it. The model remains loaded until you unload it or the MySQL HeatWave Cluster is restarted.

Make Predictions

You can make predictions on a single row of data or on the table of data.

- Make a prediction for a single row of data using the `ML_PREDICT_ROW` routine. In this example, data is assigned to a `@row_input` session variable, and the variable is called by the routine. The model handle is called using the `@model` session variable. Optionally, use `\G` to view the output in an easily readable format.

```
mysql> SET @row_input = JSON_OBJECT(
           "sepal length", 7.3,
           "sepal width", 2.9,
           "petal length", 6.3,
           "petal width", 1.8);

mysql> SELECT sys.ML_PREDICT_ROW(@row_input, @model, NULL)\G
*************************** 1. row ***************************
sys.ML_PREDICT_ROW(@row_input, @iris_model, NULL): {"Prediction": "Iris-virginica", "ml_results": {"p
                                                    "probabilities": {"Iris-setosa": 0.0, "Iris-virgi
                                                    "petal width": 1.8, "sepal width": 2.9, "petal le
1 row in set (0.7336 sec)
```

  Based on the feature inputs that were provided, the model predicts that the Iris plant is of the class `Iris-virginica`. The feature values used to make the prediction are also shown.

- Make predictions for a table of data using the `ML_PREDICT_TABLE` routine. The routine takes data from the `iris_test` table as input and writes the predictions to an `iris_predictions` output table.

```
mysql> CALL sys.ML_PREDICT_TABLE('ml_data.iris_test', @model, ml_data.iris_predictions', NULL);
```

  To view `ML_PREDICT_TABLE` results, query the output table. Optionally, use `\G` to view the output in an easily readable format. For example:

```
mysql> SELECT * from ml_data.iris_predictions LIMIT 5\G
*************************** 1. row ***************************
_4aad19ca6e_pk_id: 1
     sepal length: 5.9
      sepal width: 3
     petal length: 4.2
      petal width: 1.5
            class: Iris-setosa
       Prediction: Iris-setosa
       ml_results: {"predictions": {"class": "Iris-setosa"}, "probabilities": {"Iris-setosa": 0.9938,
*************************** 2. row ***************************
_4aad19ca6e_pk_id: 2
     sepal length: 6.9
      sepal width: 3.1
     petal length: 5.4
      petal width: 2.1
            class: Iris-virginica
       Prediction: Iris-virginica
       ml_results: {"predictions": {"class": "Iris-virginica"}, "probabilities": {"Iris-setosa": 0.44
*************************** 3. row ***************************
_4aad19ca6e_pk_id: 3
```

```
     sepal length: 5.1
      sepal width: 3.3
     petal length: 1.7
      petal width: 0.5
            class: Iris-versicolor
       Prediction: Iris-versicolor
       ml_results: {"predictions": {"class": "Iris-versicolor"}, "probabilities": {"Iris-setosa": 0.4237,
*************************** 4. row ***************************
_4aad19ca6e_pk_id: 4
     sepal length: 6
      sepal width: 3.4
     petal length: 4.5
      petal width: 1.6
            class: Iris-setosa
       Prediction: Iris-setosa
       ml_results: {"predictions": {"class": "Iris-setosa"}, "probabilities": {"Iris-setosa": 0.9846, "Ir
*************************** 5. row ***************************
_4aad19ca6e_pk_id: 5
     sepal length: 5.5
      sepal width: 2.5
     petal length: 4
      petal width: 1.3
            class: Iris-setosa
       Prediction: Iris-setosa
       ml_results: {"predictions": {"class": "Iris-setosa"}, "probabilities": {"Iris-setosa": 0.9982, "Ir
5 rows in set (0.0474 sec)
```

The table shows the predictions and the feature column values used to make each prediction.

Generating Explanations

After creating predictions (either on a single row of data or the table of data), you can generate explanations to understand how the predictions were made and review which features had the most influence on predictions.

- Generate an explanation for a prediction made on a row of data using the `ML_EXPLAIN_ROW` routine with the Permutation Importance prediction explainer. Optionally, use `\G` to view the output in an easily readable format:

```
mysql> SELECT sys.ML_EXPLAIN_ROW(JSON_OBJECT("sepal length", 7.3, "sepal width", 2.9,  "petal length", 6.
       @iris_model, JSON_OBJECT('prediction_explainer', 'permutation_importance'))\G
*************************** 1. row ***************************
sys.ML_EXPLAIN_ROW(JSON_OBJECT("sepal length", 7.3, "sepal width", 2.9,  "petal length", 6.3, "petal widt
       @iris_model, JSON_OBJECT('prediction_explainer', 'permutation_importance')):
                              {"Notes": "petal length (6.3) contributed the most against predicting Iri
                               "Prediction": "Iris-virginica", "ml_results": {"notes": "petal length (6
                               "predictions": {"class": "Iris-virginica"},
                               "attributions": {"petal width": -0.13, "petal length": -0.8, "sepal leng
1 row in set (1.6709 sec)
```

For the Permutation Importance prediction explainer, the attribution values show which features contributed most to the prediction, with petal length and petal width being the most important features. The other features have a 0 value indicating that they did not contribute to the prediction.

- Generate explanations for predictions made for a table of data using the `ML_EXPLAIN_TABLE` routine with the Permutation Importance prediction explainer.

  For the Permutation Importance prediction explainer, feature importance is presented as an attribution value ranging from -1 to 1. A positive value indicates that a feature contributed toward the prediction. A negative value indicates that the feature contributes positively towards one of the other possible predictions.

```
mysql> CALL sys.ML_EXPLAIN_TABLE('ml_data.iris_test', @iris_model, 'ml_data.iris_explanations', JSON_OBJE
```

  To view `ML_EXPLAIN_TABLE` results, query the output table. Optionally, use `\G` to view the output in an easily readable format. For example:

```
mysql> SELECT * FROM ml_data.iris_explanations LIMIT 5\G
*************************** 1. row ***************************
       _4aad19ca6e_pk_id: 1
            sepal length: 5.9
             sepal width: 3
            petal length: 4.2
             petal width: 1.5
                   class: Iris-setosa
              Prediction: Iris-setosa
sepal length_attribution: -0.66
 petal width_attribution: -0.5
petal length_attribution: -0.37
                   Notes: sepal length (5.900000095367432) contributed the most against predicting Ir
              ml_results: {"attributions": {"sepal length": -0.66, "petal width": -0.5, "petal length
                           "notes": "sepal length (5.900000095367432) contributed the most against pr
*************************** 2. row ***************************
       _4aad19ca6e_pk_id: 2
            sepal length: 6.9
             sepal width: 3.1
            petal length: 5.4
             petal width: 2.1
                   class: Iris-virginica
              Prediction: Iris-virginica
sepal length_attribution: -0.03
 petal width_attribution: -0.49
petal length_attribution: -0.92
                   Notes: petal length (5.400000095367432) contributed the most against predicting Ir
              ml_results: {"attributions": {"sepal length": -0.03, "petal width": -0.49, "petal lengt
                           "notes": "petal length (5.400000095367432) contributed the most against pr
*************************** 3. row ***************************
       _4aad19ca6e_pk_id: 3
            sepal length: 5.1
             sepal width: 3.3
            petal length: 1.7
             petal width: 0.5
                   class: Iris-versicolor
              Prediction: Iris-versicolor
sepal length_attribution: -0.04
 petal width_attribution: -0.79
petal length_attribution: -0.77
                   Notes: petal width (0.5) contributed the most against predicting Iris-versicolor
              ml_results: {"attributions": {"sepal length": -0.04, "petal width": -0.79, "petal lengt
                           "notes": "petal width (0.5) contributed the most against predicting Iris-v
*************************** 4. row ***************************
       _4aad19ca6e_pk_id: 4
            sepal length: 6
             sepal width: 3.4
            petal length: 4.5
             petal width: 1.6
                   class: Iris-setosa
              Prediction: Iris-setosa
sepal length_attribution: -0.97
 petal width_attribution: -0.51
petal length_attribution: -0.38
                   Notes: sepal length (6.0) contributed the most against predicting Iris-setosa
              ml_results: {"attributions": {"sepal length": -0.97, "petal width": -0.51, "petal lengt
                           "notes": "sepal length (6.0) contributed the most against predicting Iris-
*************************** 5. row ***************************
       _4aad19ca6e_pk_id: 5
            sepal length: 5.5
             sepal width: 2.5
            petal length: 4
             petal width: 1.3
                   class: Iris-setosa
              Prediction: Iris-setosa
sepal length_attribution: -0.47
 petal width_attribution: -0.54
petal length_attribution: -0.57
                   Notes: petal length (4.0) contributed the most against predicting Iris-setosa
              ml_results: {"attributions": {"sepal length": -0.47, "petal width": -0.54, "petal lengt
                           "notes": "petal length (4.0) contributed the most against predicting Iris-
```

```
5 rows in set (0.0447 sec)
```

Score the Model

Score the model with `ML_SCORE` to assess the reliability of the model. This example uses the `balanced_accuracy` metric, which is one of the many scoring metrics that MySQL HeatWave AutoML supports.

```
mysql> CALL sys.ML_SCORE('ml_data.iris_validate', 'class', @iris_model, 'balanced_accuracy', @score, NULL);
```

To retrieve the computed score, query the `@score` session variable.

```
mysql> SELECT @score;
+--------------------+
| @score             |
+--------------------+
| 0.9583333134651184 |
+--------------------+
```

Unloading the Model

Unload the model using `ML_MODEL_UNLOAD`:

```
mysql> CALL sys.ML_MODEL_UNLOAD(@model);
```

To avoid consuming too much memory, it is good practice to unload a model when you are finished using it.

3.4 Quickstart: Set Up a GenAI-Powered Help Chat

This quickstart shows how to set up a GenAI-powered Help Chat that uses the information available in MySQL HeatWave User Guide to provide answers to MySQL HeatWave-related questions.

Note

This quickstart assumes that you are familiar with MySQL HeatWave.

This quickstart contains the following sections:

- Before You Begin
- Set Up the Object Storage Bucket
- Connect to the DB System
- Set Up the Vector Store
- Start a Chat Session
- Clean Up

Before You Begin

- Review the requirements to use MySQL HeatWave and MySQL HeatWave GenAI.
  - To run this quickstart, you need a DB System version 9.0.0 - Innovation or higher.

    Learn how to create a DB System and edit a DB System in Oracle Cloud Infrastructure (OCI).
  - If not already done, add a MySQL HeatWave Cluster of shape `HeatWave.512GB` to your DB System.

    Learn how to add a MySQL HeatWave Cluster and edi a MySQL HeatWave Cluster in OCI.

  - If not already done, enable MySQL HeatWave Lakehouse on the DB System.
  - Enable the DB System to access an OCI Object Storage bucket. For more information, see Resource Principals.
- To Set Up the Vector Store, get the `VECTOR_STORE_LOAD_EXEC` privilege. For more information, see Section 7.3, “MySQL HeatWave GenAI Roles and Privileges”.

Set Up the Object Storage Bucket

1. Download the MySQL HeatWave user guide PDF.
2. Create an OCI Object Storage bucket with the name `quickstart_bucket`.
3. Upload the PDF file to the Object Storage Bucket using the prefix `quickstart/`. This prefix creates a new folder, `quickstart`, in `quickstart_bucket`.

Connect to the DB System

- Connect to your DB System.

Note

If you want to Ingest Files Using Asynchronous Load, X protocol is not supported. To set up a vector store using this method, ensure that you use the classic MySQL protocol while connecting to the database.

```
mysqlsh -uDBSystemAdminName -pDBSystemPassword -hDBSystemPrivateIP --sqlc
```

Set Up the Vector Store

1. Create and use a new database:

```
mysql> CREATE DATABASE quickstart_db;
mysql> USE quickstart_db;
```

2. Create the vector table and load the source document:

```
mysql> CALL sys.VECTOR_STORE_LOAD('oci://quickstart_bucket@Namespace/quickstart/heatwave-en.a4.pdf',
  '{"table_name": "quickstart_embeddings"}');
```

   Replace *`Namespace`* with the name of the Object Storage bucket namespace that you are using.

   This creates an asynchronous task in the background which loads the vector embeddings into the specified vector store table `quickstart_embeddings`.

   The routine output that is displayed contains a query with the task ID.

3. To track the progress of the task, run the task query displayed on the screen:

```
SELECT mysql_tasks.task_status_brief("TaskID");
```

   Replace *`TaskID`* with the displayed task ID.

   The output looks similar to the following:

```
+-----------------------------------------------------------------------------------------+
| mysql_tasks.task_status_brief("TaskID")                                                 |
+-----------------------------------------------------------------------------------------+
| {"data": {"tables_to_load": "[{\"table_name\": \"quickstart_embeddings\",                |
| \"load_progress\": 40.0}]"}, "status": "RUNNING", "message": "Loading in progress...",  |
| "progress": 40}                                                                         |
+-----------------------------------------------------------------------------------------+
```

4. When the task status changes to `Completed`, verify that embeddings are loaded in the vector store table:

```
mysql> SELECT COUNT(*) FROM quickstart_embeddings;
```

If you see a numerical value in the output, your embeddings are successfully loaded in the vector store table.

Start a Chat Session

1. Clear the previous chat history and states:

```
mysql> SET @chat_options=NULL;
```

2. Ask your question using MySQL HeatWave Chat:

```
mysql> CALL sys.HEATWAVE_CHAT("What is HeatWave AutoML?");
```

The `HEATWAVE_CHAT` routine automatically loads the LLM and runs a semantic search on the available vector stores and retrieves context, by default. The output is similar to the following:

```
|  MySQL HeatWave AutoML is a feature of MySQL HeatWave that makes it easy to use
machine learning, whether you are a novice user or an experienced ML practitioner.
It analyzes the characteristics of the data and creates an optimized machine
learning model that can be used to generate predictions and explanations. The
data and models never leave MySQL HeatWave, saving time and effort while keeping
the data and models secure. MySQL HeatWave AutoML is optimized for MySQL HeatWave shapes and
scaling, and all processing is performed on the MySQL HeatWave Cluster. |
```

3. Ask a follow-up question:

```
mysql> CALL sys.HEATWAVE_CHAT("How to set it up?");
```

The output is similar to the following:

```
|  Answer: To set up MySQL HeatWave AutoML in MySQL HeatWave, you need to
    follow these steps:

1. Create a new user with the necessary privileges.
2. Create a new schema named ML_SCHEMA_user_name, where the user_name
    is the name of the owning user.
3. Create an InnoDB table named MODEL_CATALOG in the ML_SCHEMA_user_name
    schema.
4. Grant the necessary privileges to the new user for the ML_SCHEMA_user_name
    schema and the MODEL_CATALOG table.
5. Create a new machine learning model using MySQL HeatWave AutoML.

To monitor MySQL HeatWave AutoML status, you can query the rapid_ml_status variable
or the ML_STATUS column of the performance_schema.rpd_nodes table. The
rapid_ml_status variable provides the status of MySQL HeatWave AutoML, with possible
values of ON and OFF. If the value is ON, MySQL HeatWave AutoML is up and running,
and if it's OFF, MySQL HeatWave AutoML is down. You can also query the ML            |
```

You can continue asking follow-up questions in the same chat session.

Clean Up

To avoid being billed for the resources that you created for this quickstart, perform the following steps:

1. Delete the database that you created:

```
mysql> DROP DATABASE quickstart_db;
```

2. If you created a new DB System, then delete the DB System. For more information, see Deleting a DB System.

3. Delete `quickstart_bucket`. For more information, see Deleting the Object Storage Bucket.

Chapter 4 Load and Manage Data

Table of Contents

4.1 Load Data into DB System 38
4.1.1 Import Data 38
4.1.2 Bulk Ingest Data 38
4.2 Load Data from DB System into MySQL HeatWave Cluster 44
4.2.1 Supported Data Types for MySQL HeatWave 45
4.2.2 Prepare to Load Data 46
4.2.3 Enable or Disable Data Compression 49
4.2.4 About Auto Parallel Load 49
4.2.5 Load Data Using Auto Parallel Load 51
4.2.6 Load Data Manually 54
4.2.7 About Change Propagation 57
4.2.8 Reload Tables 59
4.3 Load Data from Object Storage into MySQL HeatWave Cluster 59
4.3.1 About MySQL HeatWave Lakehouse 59
4.3.2 Additional MySQL HeatWave Lakehouse Requirements 62
4.3.3 MySQL HeatWave Lakehouse Privileges 63
4.3.4 Supported File Formats and Data Types 63
4.3.5 Access Object Storage 74
4.3.6 Lakehouse External Table Syntax 77
4.3.7 About Lakehouse Auto Parallel Load 107
4.3.8 Create External Tables Using Auto Parallel Load 109
4.3.9 Create External Tables Manually 130
4.3.10 Create Tables with Stored Generated Columns 150
4.3.11 Load Structured Data Using Lakehouse Auto Parallel Load 153
4.3.12 Load Structured Data Manually 158
4.3.13 Load Unstructured Data 161
4.3.14 Update Tables 161
4.4 Unload Data from MySQL HeatWave Cluster 178
4.4.1 Unload Data Automatically 179
4.4.2 Unload Data Manually 183
4.5 Retrieve MySQL HeatWave Metadata 184
4.5.1 View Secondary Engine Definitions 184
4.5.2 List Excluded Columns 185
4.6 Export Query Results to Object Storage 186
4.6.1 About Exporting Query Results 186
4.6.2 Requirements to Export Query Results 187
4.6.3 Export Query Results to CSV Files 187
4.6.4 Export Query Results to Parquet Files 195
4.6.5 Export Query Results to ND-JSON Files 200
4.6.6 Redacted PAR URLs in Exported Log Files 206
4.6.7 File Name Format for Exported Query Result Files 206
4.7 Recover DB System Data 207
4.8 Monitor Data Loading 207
4.8.1 MySQL HeatWave Node Status Monitoring 207
4.8.2 MySQL HeatWave Memory Usage Monitoring 208
4.8.3 Data Load Progress and Status Monitoring 208
4.8.4 Change Propagation Monitoring 209

The sections in this chapter describe how to load and manage data in MySQL HeatWave.

4.1 Load Data into DB System

The sections in this topic describe how to load data into the DB System.

4.1.1 Import Data

This topic describes how to import data into the DB System.

Before You Begin

- Review the Requirements.

Import Data

Use MySQL Shell Shell for importing data into a DB System. Its dump and load utilities are specifically built for this task and support all types of data exports and imports. You can export data to Object Storage and also import it from there. Ensure that the source MySQL version is 5.7.9 or higher.

- For MySQL HeatWave on OCI, see Importing and Exporting Databases in the *MySQL HeatWave on OCI Service Guide*.
- For MySQL HeatWave on AWS, see Importing Data in the *MySQL HeatWave on AWS Service Guide*.
- For MySQL HeatWave for Azure, see Importing Data to MySQL HeatWave in the *MySQL HeatWave for Azure Service Guide*.

Tip

For an OLTP workload that resides in an on-premises instance of MySQL Server, inbound replication is recommended for replicating data to the DB System for offload to the MySQL HeatWave Cluster. For more information, see Replication in the *MySQL HeatWave on OCI Service Guide*.

What's Next

- Learn how to Bulk Ingest Data to MySQL Server.

4.1.2 Bulk Ingest Data

MySQL includes a bulk load extension to the `LOAD DATA` statement that is only available in MySQL HeatWave on AWS. It can do the following:

- Optimize the loading of data sorted by primary key.
- Optimize the loading of unsorted data.
- Optimize the loading of data from an object store.
- Optimize the loading of data from a series of files.
- Load a MySQL Shell dump file (As of MySQL 8.4.0).
- Load ZSTD compressed CSV files (As of MySQL 8.4.0).
- Monitor bulk load progress with the Performance Schema (As of MySQL 8.4.0).
- Large data support (As of MySQL 9.0.0).
- Support for tables with generated columns (As of MySQL 9.5.0). This includes tables with stored and virtual generated columns. Stored generated columns can be part of secondary indexes and

the primary key. Virtual generated columns can be part of secondary indexes. Tables can have a combination of multiple stored and virtual generated columns. The input CSV files must contain data for the stored and virtual generated columns.

- Support for bulk load into tables with existing data (non-empty tables) (As of MySQL 9.6.0). The bulk load operation compares the records from the input CSV files and the existing table, and then inserts the data into the existing table so that it is sorted correctly.
- Support for tables with foreign keys (As of MySQL 9.7.0). MySQL HeatWave ensures that all foreign key constraints are satisfied for each record of the table. You can load both single-column and multi-column foreign keys. Checks on foreign keys are controlled by the `foreign_key_checks` system variable.
- Support for bulk load into partitioned tables (As of MySQL 26.7.0). Use `PARTITION` with the `FILES` keyword to map one file or a range of files to each partition. If rows from the mapped files do not belong to the named partition, the bulk load fails.
- Support for bulk load into tables with prefix indexes (As of MySQL 26.7.0). Bulk load supports prefix indexes in clustered and secondary indexes.
- Support for bulk load into tables with `CHECK` constraints (As of MySQL 26.7.0). After the bulk load completes, MySQL HeatWave validates `CHECK` constraints for the loaded rows. If any row fails validation, the bulk load fails and the table remains unmodified.

  See: CHECK Constraints.

Use a second session to monitor bulk load progress:

- If the data is sorted, there is a single stage: `loading`.
- If the data is unsorted, there are two stages: `sorting` and `loading`.

See: Section 11.2.1.2, “Bulk Ingest Data to MySQL Server Limitations”.

Bulk Ingest Data Type Support

`LOAD DATA` with `ALGORITHM=BULK` supports the following data types.

Table 4.1 Supported Data Types for Bulk Ingest

Data Type	Available MySQL version	Comments
`INT`	All	
`SMALLINT`	All	
`TINYINT`	All	
`BIGINT`	All	
`CHAR`	All	
`BINARY`	All	
`VARCHAR`	All	Before MySQL 9.0.0, the record must fit in the page as there is no large data support.
`VARBINARY`	All	Before MySQL 9.0.0, the record must fit in the page as there is no large data support.
`NUMERIC`	All	
`DECIMAL`	All	
`UNSIGNED NUMERIC`	All	

Data Type	Available MySQL version	Comments
UNSIGNED DECIMAL	All	
DOUBLE	All	
FLOAT	All	
DATE	All	
DATETIME	All	
BIT	As of MySQL 9.0.0	
ENUM	As of MySQL 9.0.0	
JSON	As of MySQL 9.0.0	
SET	As of MySQL 9.0.0	
TIMESTAMP	As of MySQL 9.0.0	
YEAR	As of MySQL 9.0.0	
TINYBLOB	As of MySQL 9.0.0	
BLOB	As of MySQL 9.0.0	
MEDIUMBLOB	As of MySQL 9.0.0	
LONGBLOB	As of MySQL 9.0.0	
TINYTEXT	As of MySQL 9.0.0	
TEXT	As of MySQL 9.0.0	
MEDIUMTEXT	As of MySQL 9.0.0	
LONGTEXT	As of MySQL 9.0.0	
GEOMETRY	As of MySQL 9.0.0	
GEOMETRYCOLLECTION	As of MySQL 9.0.0	
POINT	As of MySQL 9.0.0	
MULTIPOINT	As of MySQL 9.0.0	
LINESTRING	As of MySQL 9.0.0	
MULTILINESTRING	As of MySQL 9.0.0	
POLYGON	As of MySQL 9.0.0	
MULTIPOLYGON	As of MySQL 9.0.0	
VECTOR	As of MySQL 9.3.0	

For the data types that MySQL HeatWave supports, see: Section 4.2.1, “Supported Data Types for MySQL HeatWave”

See: Section 11.2.1.2, “Bulk Ingest Data to MySQL Server Limitations”.

Bulk Ingest Syntax

```
mysql> LOAD DATA
  [LOW_PRIORITY | CONCURRENT]
  [FROM]
  {INFILE | URL | URI | S3} {'file_prefix' | 'options'} [COUNT N]
  [IN PRIMARY KEY ORDER]
  INTO TABLE tbl_name
  [PARTITION (partition_name FILES file_number [TO file_number] [, partition_name FILES file_number [TO fi
  [CHARACTER SET charset_name] [COMPRESSION = {'ZSTD'}]
  [{FIELDS | COLUMNS}
      [TERMINATED BY 'string']
      [[OPTIONALLY] ENCLOSED BY 'char']
      [ESCAPED BY 'char']
  ]
```

```
  [LINES
      [TERMINATED BY 'string']
  ]
  [IGNORE number {LINES | ROWS}]
  [PARALLEL = number]
  [MEMORY = M]
  [ALGORITHM = BULK]

options: {
   JSON_OBJECT("key","value"[,"key","value"] ...)
        "key","value": {
        "url-prefix","prefix"
        ["url-sequence-start",0]
        ["url-suffix","suffix"]
        ["url-prefix-last-append","@"]
        ["is-dryrun",{true|false}]
        }
}
```

The additional `LOAD DATA` clauses are:

- `FROM`: Makes the statement more readable.
- `URL`: A URL accessible with a HTTP GET request.
- `URI`: Available as of MySQL 9.4.0. The file location of the URI.
- `S3`: The AWS S3 file location.

  This requires the user privilege `LOAD_FROM_S3`.

  See: Section 11.2.1.2, “Bulk Ingest Data to MySQL Server Limitations”.
- `COUNT`: The number of files in a series of files.

  For `COUNT 5` and *`file_prefix`* set to `data.csv.`, the five files would be: `data.csv.1`, `data.csv.2`, `data.csv.3`, `data.csv.4`, and `data.csv.5`.
- `PARTITION`: Use the `PARTITION` clause to map a single file or a range of files to each named table partition during bulk load. Bulk load processes one mapped partition at a time. Support for bulk load into partitioned tables is available as of MySQL 26.7.0.

  Use `FILES n` to map a single file, or `FILES first TO last` to map a range of files.

  Note

  The regular `PARTITION (partition_name [, partition_name] ...)` clause without `FILES` mappings is not supported with `ALGORITHM=BULK`.
- `IN PRIMARY KEY ORDER`: Use when the data is already sorted. The values should be in ascending order within the file.

  For a file series, the primary keys in each file must be disjoint and in ascending order from one file to the next.
- `PARALLEL`: The number of concurrent threads to use. A typical value might be 16, 32 or 48. The default value is 16.

  `PARALLEL` does not require `CONCURRENT`.
- `MEMORY`: The amount of memory to use. A typical value might be 512M or 4G. The default value is 1G.
- `ALGORITHM`: Set to `BULK` for bulk load. The file format is CSV.

- `COMPRESSION`: The file compression algorithm. Bulk load supports the ZSTD algorithm.
- `options` is a JSON object literal that includes:
  - `url-prefix`: The common URL prefix for the files to load.
  - `url-sequence-start`: The sequence number for the first file.

    The default value is 1, and the minimum value is 0. The value cannot be a negative number. The value can be a string or a number, for example, "134", or "default".
  - `url-suffix`: The file suffix.
  - `url-prefix-last-append`: The string to append to the prefix of the last file.

    This supports MySQL Shell dump files.
  - `is-dryrun`: Set to `true` to run basic checks and report if bulk load is possible on the given table. The default value is `false`.

    To enable `is-dryrun`, use any of the following values: `true`, `"true"`, `"1"`, `"on"` or `1`.

    To disable `is-dryrun`, use any of the following values: `false`, `"false"`, `"0"`, `"off"` or `0`.

Unsupported Bulk Ingest Clauses

`LOAD DATA` with `ALGORITHM=BULK` does not support these clauses.

```
LOAD DATA
  [LOCAL]
  [REPLACE | IGNORE]
  [LINES
      [STARTING BY 'string']
  ]
  [(col_name_or_user_var
      [, col_name_or_user_var] ...)]
  [SET col_name={expr | DEFAULT}
      [, col_name={expr | DEFAULT}] ...]
```

Syntax Examples

- An example that loads unsorted data from AWS S3 with 48 concurrent threads and 4G of memory:

```
mysql> GRANT LOAD_FROM_S3 ON *.* TO load_user@localhost;

mysql> LOAD DATA FROM S3 's3-us-east-1://innodb-bulkload-dev-1/lineitem.tbl'
        INTO TABLE lineitem
        FIELDS TERMINATED BY "|"
        OPTIONALLY ENCLOSED BY '"'
        LINES TERMINATED BY '\n'
        PARALLEL = 48
        MEMORY = 4G
        ALGORITHM=BULK;
```

- An example that loads eight files of sorted data from AWS S3. The `file_prefix` ends with a period. The files are `lineitem.tbl.1`, `lineitem.tbl.2`, ... `lineitem.tbl.8`:

```
mysql> GRANT LOAD_FROM_S3 ON *.* TO load_user@localhost;

mysql> LOAD DATA FROM S3 's3-us-east-1://innodb-bulkload-dev-1/lineitem.tbl.' COUNT 8
        IN PRIMARY KEY ORDER
        INTO TABLE lineitem
        FIELDS TERMINATED BY "|"
        OPTIONALLY ENCLOSED BY '"'
        LINES TERMINATED BY '\n'
        ALGORITHM=BULK;
```

- An example that loads sorted data into a partitioned table by mapping file groups to specific partitions (supported as of MySQL 26.7.0):

```
mysql> LOAD DATA FROM INFILE 'OUT.' COUNT 16
       IN PRIMARY KEY ORDER
       INTO TABLE test.t1
       PARTITION (
         p0 FILES 0 TO 3,
         p1 FILES 4 TO 7,
         p2 FILES 8 TO 11,
         p3 FILES 12 TO 15
       )
       FIELDS TERMINATED BY ","
       OPTIONALLY ENCLOSED BY '"'
       ALGORITHM=BULK;
```

  For a partition that has a single file, specify `FILES n` instead of a range.

- An example that performs a dry run on a sequence of MySQL Shell dump files compressed with the ZSTD algorithm:

```
mysql> GRANT LOAD_FROM_URL ON *.* TO load_user@localhost;

mysql> LOAD DATA FROM URL
       '{"url-prefix","https://example.com/bucket/test@lineitem@","url-sequence-start",0,"url-suffix
       COUNT 20
       INTO TABLE lineitem
       CHARACTER SET ???? COMPRESSION = {'ZSTD'}
       FIELDS TERMINATED BY "|"
       OPTIONALLY ENCLOSED BY '"'
       LINES TERMINATED BY '\n'
       IGNORE 20000 LINES
       ALGORITHM=BULK;
```

- An example that loads data with the URI keyword (supported as of MySQL 9.4.0):

```
mysql> GRANT LOAD_FROM_URL ON *.* TO load_user@localhost;

mysql> LOAD DATA FROM URI 'https://data_files.com/data_files_1.tbl'
       INTO TABLE lineitem
       FIELDS TERMINATED BY "|"
       OPTIONALLY ENCLOSED BY '"'
       LINES TERMINATED BY '\n'
       ALGORITHM=BULK;
```

- An example that monitors bulk load progress in a second session.

  - Review the list of stages with the following query:

```
mysql> SELECT NAME, ENABLED, TIMED FROM performance_schema.setup_instruments
       WHERE ENABLED='YES' AND NAME LIKE "stage/bulk_load%";
```

  - Enable the `events_stages_current` with the following query:

```
mysql> UPDATE performance_schema.setup_consumers
       SET ENABLED = 'YES' WHERE NAME LIKE 'events_stages_current';
```

  - Use one session to run bulk load, and monitor progress in a second session:

```
mysql> SELECT thread_id, event_id, event_name, WORK_ESTIMATED,  WORK_COMPLETED
       FROM performance_schema.events_stages_current;
--------------
SELECT thread_id, event_id, event_name, WORK_ESTIMATED,  WORK_COMPLETED FROM performance_schema.eve
--------------

+-----------+----------+---------------------------------+----------------+----------------+
| thread_id | event_id | event_name                      | WORK_ESTIMATED | WORK_COMPLETED |
+-----------+----------+---------------------------------+----------------+----------------+
|        49 |        5 | stage/bulk_load_unsorted/sorting |     1207551343 |      583008145 |
+-----------+----------+---------------------------------+----------------+----------------+
```

4.2 Load Data from DB System into MySQL HeatWave Cluster

MySQL HeatWave provides the following methods to load data to MySQL HeatWave:

- Automatic loading: loads tables and partitions based on usage.
- Loading data using Auto Parallel Load: loads one or more schemas at a time and facilitates loading by automating manual steps and optimizing the number of parallel load threads for a faster load.
- Loading data manually: loads one table at a time and involves executing multiple statements for each table.
- Load data using the MySQL HeatWave Console: applicable to MySQL HeatWave on AWS users. This GUI-based and MySQL HeatWave Autopilot enabled method loads selected schemas and tables using an optimized number of parallel load threads for a faster load.

MySQL HeatWave loads data with batched, multi-threaded reads from DB System tables. MySQL HeatWave then converts the data into columnar format and sends it over the network to distribute it among MySQL HeatWave nodes in horizontal slices. MySQL HeatWave partitions data by the table primary key, unless the table definition includes data placement keys.

Concurrent DML operations and queries on the MySQL node are supported while a data load operation is in progress; however, concurrent operations on the MySQL node can affect load performance and vice versa.

After loading tables, change propagation automatically updates MySQL HeatWave with any data changes made on the DB System node.

For each table that is loaded in MySQL HeatWave, the default heap segment size is 64KB per table, and this is allocated from the root heap. The root heap available to MySQL HeatWave is approximately 400GB.

MySQL HeatWave compresses data as it is loaded, which permits MySQL HeatWave nodes to store more data at a minor cost to performance. If you do not want to compress data as it is loaded in MySQL HeatWave, you must disable compression before loading data.

The loading of data into MySQL HeatWave can be classified into three types: *Initial Bulk Load*, *Incremental Bulk Load*, and *Change Propagation*.

- *Initial Bulk Load*: Performed when loading data into MySQL HeatWave for the first time, or when reloading data. The best time to perform an initial bulk load is during off-peak hours, as bulk load operations can affect OLTP performance on the DB System.
- *Incremental Bulk Load*: Performed when there is a substantial amount of data to load into tables that are already loaded in MySQL HeatWave. An incremental bulk load involves these steps:

  1. Performing a `SECONDARY_UNLOAD` operation to unload a table from MySQL HeatWave.
  2. Importing data into the table on the DB System node.
  3. Performing a `SECONDARY_LOAD` operation to reload the table into MySQL HeatWave.

  Depending on the amount of data, an incremental bulk load may be a faster method of loading new data than waiting for change propagation to occur. It also provides greater control over when new data is loaded. As with initial build loads, the best time to perform an incremental bulk load is during off-peak hours, as bulk load operations can affect OLTP performance on the DB System.

- Change Propagation: After tables are loaded into MySQL HeatWave, data changes are automatically propagated from `InnoDB` tables on the DB System to their counterpart tables in MySQL HeatWave.

The topics in this section describe how to load structured data in MySQL HeatWave.

4.2.1 Supported Data Types for MySQL HeatWave

MySQL HeatWave supports the following data types. Columns with unsupported data types must be excluded, and defined as `NOT SECONDARY` before loading a table. See Section 4.2.6.1, “Exclude Table Columns”.

- Numeric data types:
  - `BIGINT`
  - `BOOL`
  - `DECIMAL`
  - `DOUBLE`
  - `FLOAT`
  - `INT`
  - `INTEGER`
  - `MEDIUMINT`
  - `SMALLINT`
  - `TINYINT`
- Temporal data types:
  - `DATE`
  - `DATETIME`
  - `TIME`
  - `TIMESTAMP`
  - `YEAR`

  Temporal types are only supported with strict SQL mode. See Strict SQL Mode.

  For limitations related to `TIMESTAMP` data type support, see Section 11.2.1.1, “Data Type Limitations”.
- String and text data types:
  - `CHAR`
  - `VARCHAR`
  - `TEXT`
  - `TINYTEXT`
  - `MEDIUMTEXT`
  - `LONGTEXT`
  - `VECTOR`
  - `BINARY` (as of MySQL 9.6.1).

- `BINARY` is not supported for MySQL HeatWave Lakehouse.
- For Auto Parallel Load, columns with `BINARY` must have variable-length encoding.
- Columns with `BINARY` are considered as strings during data analysis to enhance cluster memory estimation.

For string and text data type limitations, see Section 11.2.1.1, “Data Type Limitations”.

- `ENUM`

  For `ENUM` limitations, see Section 11.2.1.1, “Data Type Limitations”.

- JSON

  As of MySQL 9.0.0, the `JSON` data type supports the following functions and operators:

  - Cast-functions and operators, see: Section 5.3.3, “Cast Functions and Operators”.
  - `COALESCE()` and `IN()`, see: Section 5.3.4, “Comparison Functions and Operators”.
  - Flow control functions and operators, see: Section 5.3.5, “Control Flow Functions and Operators”.
  - Mathematical functions, see: Section 5.3.10, “Mathematical Functions”.
  - String functions and operators, see: Section 5.3.11, “String Functions and Operators”.

What's Next

- Learn how to perform the following tasks:
  - Run queries
  - Modify tables
  - Retrieve MySQL HeatWave Metadata
  - Optimize workloads for OLAP and OLTP
  - Monitor MySQL HeatWave processes

4.2.2 Prepare to Load Data

This topic describes how to prepare your data for loading into MySQL HeatWave.

Before You Begin

- Import Data into MySQL HeatWave DB System.

Prepare Data

Before loading data, ensure that you meet the following requirements:

- The tables you intend to load must be set to `ENGINE=InnoDB` tables. You can manually convert tables by using the following `ALTER TABLE` statement:

```
mysql> ALTER TABLE tbl_name ENGINE=InnoDB;
```

- The tables you intend to load must be defined with a primary key. You can add a primary key using the following syntax:

- ```
  mysql> ALTER TABLE tbl_name ADD PRIMARY KEY (column);
  ```
  ```

Adding a primary key is a table-rebuilding operation.

Primary key columns defined with column prefixes are not supported.

Load time is affected if the primary key contains more than one column, or if the primary key column is not an `INTEGER` column. The impact on MySQL performance during load, change propagation, and query processing depends on factors such as data properties, available resources (compute, memory, and network), and the rate of transaction processing on the DB System.

- Identify all of the tables that your queries access to ensure that you load all of them into MySQL HeatWave. If a query accesses a table that is not loaded into MySQL HeatWave, it will not be offloaded to MySQL HeatWave for processing.
- To minimize the number of MySQL HeatWave nodes required for your data, exclude table columns that are not accessed by your queries.
- The maximum column widths supported for different string and text data types are as follows:

Table 4.2 Supported Column Widths

String/Text Datatype	Column Width Supported from MySQL 9.2.2	Column Width Supported Prior to MySQL 9.2.2
`MEDIUMTEXT`	4192192 bytes	65532 bytes
`LONGTEXT`	4192192 bytes	65532 bytes
`JSON`	4192192 bytes	65532 bytes
`TEXT`	65535 bytes	65532 bytes
`CHAR`	65532 bytes	65532 bytes
`VARCHAR`	65532 bytes	65532 bytes
`VECTOR`	65532 bytes	65532 bytes

This applies to intermediate relations and final query results as well.

- The number of columns per table cannot exceed 1017.
- To save space in memory, set `CHAR`, `VARCHAR`, and `TEXT` type column lengths to the minimum length required for the longest string value.
- Data is partitioned by the table primary key when no data placement keys are defined. Only consider defining data placement keys if partitioning data by the primary key does not provide suitable performance.

  Reserve the use of data placement keys for the most time-consuming queries. In such cases, define data placement keys on:

  - The most frequently used `JOIN` keys.
  - The keys of the longest running queries.

  Consider using Auto Data Placement for data placement recommendations.

- Apply dictionary encoding to `CHAR`, `VARCHAR`, and `TEXT` columns whenever appropriate. It reduces memory consumption on the MySQL HeatWave Cluster nodes. Use the following criteria when selecting string columns for dictionary encoding:

  1. The column is not used as a key in `JOIN` queries.
  2. Your queries do not perform operations such as `LIKE`, `SUBSTR`, `CONCAT`, etc., on the column. Variable-length encoding supports string functions and operators as well as `LIKE` predicates; dictionary encoding does not.
  3. The column has a limited number of distinct values. Dictionary encoding is best suited to columns with a limited number of distinct values, such as `country` columns.
  4. The column is expected to have few new values added during change propagation. Avoid dictionary encoding for columns with a high number of inserts and updates. Adding a significant number of a new, unique values to a dictionary encoded column can cause a change propagation failure.

  When in doubt about choosing an encoding type, use variable-length encoding, which is applied by default when tables are loaded into MySQL HeatWave, or use Auto Encoding to obtain encoding recommendations.

- *Increase the number of read threads*

  For medium to large tables, increase the number of read threads to 32 by setting the `innodb_parallel_read_threads` variable on the DB System.

  ```
  mysql> SET SESSION innodb_parallel_read_threads = 32;
  ```

  If the DB System is not busy, you can increase the value to 64.

  Tip

  The Auto Parallel Load utility automatically optimizes the number of parallel read threads for each table. See Section 4.2.5, “Load Data Using Auto Parallel Load”. For users of MySQL HeatWave on AWS, the number of parallel read threads is also optimized when loading data from the MySQL HeatWave Console. See Manage Data in MySQL HeatWave with Workspaces in the *MySQL HeatWave on AWS Service Guide*.

- *Load tables concurrently*

  If you have many small and medium tables (less than 20GB in size), load tables from multiple sessions:

  ```
  Session 1:
  mysql> ALTER TABLE supplier SECONDARY_LOAD;

  Session 2:
  mysql> ALTER TABLE parts SECONDARY_LOAD;

  Session 3:
  mysql> ALTER TABLE region SECONDARY_LOAD;

  Session 4:
  mysql> ALTER TABLE partsupp SECONDARY_LOAD;
  ```

- *Avoid or reduce conflicting operations*

  Data load operations share resources with other OLTP DML and DDL operations on the DB System. To improve load performance, avoid or reduce conflicting DDL and DML operations. For example, avoid running DDL and large DML operations on the `LINEITEM` table while executing an `ALTER TABLE LINEITEM SECONDARY_LOAD` operation.

What's Next

- Learn how to enable or disable data compression.
- Learn how to load data using Auto Parallel Load or manually.

4.2.3 Enable or Disable Data Compression

MySQL HeatWave compresses data as it is loaded, which permits MySQL HeatWave nodes to store more data. More data per node reduces costs by minimizing the size of the MySQL HeatWave Cluster required to store the data.

While data compression results in a smaller MySQL HeatWave Cluster, decompression operations that occur as data is accessed affect performance to a small degree. Specifically, decompression operations have a minor effect on query runtimes, on the rate at which queries are offloaded to MySQL HeatWave during change propagation, and on recovery time from Object Storage.

If data storage size is not a concern, disable data compression by setting the `rapid_compression` session variable to `OFF` before loading data:

```
mysql> SET SESSION rapid_compression=OFF;
```

As of MySQL version 8.3.0, the default option is `AUTO` which automatically chooses the best compression algorithm for each column. Prior to that only `ON` and `OFF` were supported.

What's Next

- Learn about Auto Parallel Load.
- Learn how to load data using Auto Parallel Load.
- Learn how to load data using Auto Parallel Load or manually.

4.2.4 About Auto Parallel Load

Auto Parallel Load is a feature of MySQL HeatWave that automates many of the steps for loading data, which include the following:

- Excluding schemas, tables, and columns that cannot be loaded.
- Defining `RAPID` as the secondary engine for tables that are to be loaded.
- Verifying that there is sufficient memory available for the data.
- Optimizing load parallelism based on machine learning models.
- Loading data into MySQL HeatWave.

Requirements and Privileges for Auto Parallel Load

To run Auto Parallel Load in `normal` mode, the MySQL HeatWave Cluster must be active.

To use Auto Parallel Load, ask the admin user to grant you the following privileges. Replace *`user_name`* and *`database_name`* in the commands with the appropriate user name and database name.

The `PROCESS` privilege on the appropriate database in the DB System.

```
mysql> GRANT PROCESS ON database_name.* TO 'user_name'@'%';
```

The `EXECUTE` privilege on the `sys` schema.

```
mysql> GRANT EXECUTE ON sys.* TO 'user_name'@'%';
```

The `SELECT` privilege.

```
mysql> GRANT SELECT ON performance_schema.* TO 'user_name'@'%';
```

Automatic Loading and Unloading of DB System Tables and Partitions

As of MySQL 8.4.0, Auto Parallel Load adds the functionality to use Autopilot to collect statistics about frequently used DB System tables. Auto Parallel Load then automatically loads these tables into MySQL HeatWave. Conversely, if an automatically loaded table becomes cold due to not being used, then Auto Parallel Load automatically unloads it. Also, if there is not enough memory for a manually loaded table due to automatically loaded tables taking up too much memory, Auto Unload unloads automatically loaded tables to allow the manually loaded table to load.

As of MySQL 9.3.1, this feature extends to partitions. Auto Parallel Load automatically loads frequently used partitions, and automatically unloads partitions that are not being used.

Auto Parallel Load automatically loads any table that does not have the secondary engine set, or the secondary engine is set to `RAPID`.

Auto Parallel Load has a 24-hour wait interval between checking the table statistics, and only checks them during a quiet period. Therefore, within 24 hours of the first query that accesses a table, Auto Parallel Load automatically loads the table into MySQL HeatWave if there is sufficient memory.

Check Load Status of a Table

You can verify if a table is automatically loaded or loaded by a user by querying the `performance_schema.rpd_tables` table and the `LOAD_TYPE` column. A value of `SELF` indicates the table was automatically loaded, and a value of `USER` indicates the table was loaded by a user. Learn more about the rpd_tables table.

To verify the load type of a table, query the performance_schema.rpd_tables and performance_schema.rpd_table_id tables and use an `INNER JOIN` clause to combine the `ID` of the table you want to query.

Run the following commands:

```
mysql> USE performance_schema;
mysql> SELECT rpd_tables.ID, rpd_table_id.NAME, rpd_tables.LOAD_TYPE
        FROM rpd_tables INNER JOIN rpd_table_id ON rpd_table_id.ID=rpd_tables.ID
        WHERE NAME='table_name';
```

Replace *`table_name`* with the fully qualified name of the table you want to query (*`schema_name.table_name)`*.

For example:

```
mysql> USE performance_schema;
mysql> SELECT rpd_tables.ID, rpd_table_id.NAME, rpd_tables.LOAD_TYPE
        FROM rpd_tables INNER JOIN rpd_table_id ON rpd_table_id.ID=rpd_tables.ID
        WHERE NAME='schema_1.table_1';
+-----+------------------+-----------+
| ID  | NAME             | LOAD_TYPE |
+-----+------------------+-----------+
| 303 | schema_1.table_1 | USER      |
+-----+------------------+-----------+
```

To review all DB System tables, omit `WHERE NAME='table_name'`.

Remove Table from Automatic Loading

You have the option to ensure that a table is not considered for automatic loading by setting the `SECONDARY_ENGINE` of a table to `NO_AUTO`.

Run the following command:

```
mysql> ALTER TABLE table_name SECONDARY_ENGINE=NO_AUTO;
```

Replace *`table_name`* with the appropriate table name.

If the table is not currently loaded to MySQL HeatWave and SECONDARY_ENGINE is not currently set, you can run the single command. For example:

```
mysql> ALTER TABLE data_files_1 SECONDARY_ENGINE=NO_AUTO;
```

If the table is currently loaded to MySQL HeatWave and SECONDARY_ENGINE is set to RAPID, do the following:

1. Unload the table from MySQL HeatWave.

   ```
   mysql> ALTER TABLE table_name SECONDARY_UNLOAD;
   ```

   Replace table_name with your value.

   The following example unloads the data_files_1 table.

   ```
   mysql> ALTER TABLE data_files_1 SECONDARY_UNLOAD;
   ```

2. Set the SECONDARY_ENGINE of the table to NULL.

   ```
   mysql> ALTER TABLE table_name SECONDARY_ENGINE=NULL;
   ```

   The following example sets the SECONDARY_ENGINE of the table to NULL.

   ```
   mysql> ALTER TABLE data_files_1 SECONDARY_ENGINE=NULL;
   ```

3. Set the SECONDARY_ENGINE of the table to NO_AUTO. This is the setting that removes the table from consideration for automatic loading.

   ```
   mysql> ALTER TABLE table_name SECONDARY_ENGINE=NO_AUTO;
   ```

   The following example sets the SECONDARY_ENGINE of the table to NO_AUTO.

   ```
   mysql> ALTER TABLE data_files_1 SECONDARY_ENGINE=NO_AUTO;
   Records: 0  Duplicates: 0  Warnings: 1
   Warning (code 1105): HeatWave Self-Load feature is enabled. Removing `data_files_db`.`data_files_1`
   ```

   A message confirms that the table is removed from automatic loading.

What's Next

Learn how to load data using Auto Parallel Load.

4.2.5 Load Data Using Auto Parallel Load

You can load data into a MySQL HeatWave Cluster by using the Auto Parallel Load feature.

Auto Parallel Load, which can be run manually from any MySQL client or connector, is implemented as a stored procedure named heatwave_load, which resides in the MySQL sys schema. Running Auto Parallel Load involves issuing a CALL statement for the stored procedure, which takes schemas and *options* as arguments.

Before You Begin

- Import Data into MySQL HeatWave DB System.

Load Data Using Auto Parallel Load

First, run Auto Parallel Load in dryrun mode to check for errors and warnings and to inspect the generated load script. To load a single schema in dryrun mode:

```
mysql> CALL sys.heatwave_load(JSON_ARRAY("schema_name"), JSON_OBJECT("mode","dryrun"));
```

```
mysql> CALL sys.heatwave_load(JSON_ARRAY("tpch"), JSON_OBJECT("mode","dryrun"));
+-----------------------------------------------------------------------------------------------------------
```

```
| CAPACITY ESTIMATION
+-----------------------------------------------------------------------------------------------------------------------
| Default encoding for string columns: VARLEN (unless specified in the schema)
| Estimating memory footprint for 1 schema(s)
|
|                                TOTAL       ESTIMATED       ESTIMATED       TOTAL     DICTIONARY      VARL
| SCHEMA                     OFFLOADABLE   HEATWAVE NODE      MYSQL NODE      STRING        ENCODED     ENCOD
| NAME                            TABLES       FOOTPRINT       FOOTPRINT     COLUMNS        COLUMNS     COLUM
| ------                     -----------       ---------       ---------     -------     ----------     -----
| `tpch`                               8        1.79 GiB        1.15 GiB          29             24
|
| Sufficient MySQL host memory available to load all tables.
| Sufficient HeatWave cluster memory available to load all tables.
|
+-----------------------------------------------------------------------------------------------------------------------
12 rows in set (0.2959 sec)

+-----------------------------------------------------------------------------------------------------------------------
| LOAD SCRIPT GENERATION
+-----------------------------------------------------------------------------------------------------------------------
| Dryrun mode only generates the load script
| Set mode to "normal" in options to load tables
|
| Retrieve load script containing 16 generated DDL commands using the query below:
| Deprecation Notice: "heatwave_load_report" will be deprecated, please switch to "heatwave_autopilot_repor
|   SELECT log->>"$.sql" AS "Load Script" FROM sys.heatwave_autopilot_report WHERE type = "sql" ORDER BY id
|
| Applying changes will take approximately 1.25 min
|
| Caution: Executing the generated load script may alter column definitions and secondary engine flags in t
|
| Total errors encountered: 0
| Total warnings encountered: 0
|
+-----------------------------------------------------------------------------------------------------------------------
14 rows in set (0.2959 sec)
```

This statement loads `tpch` schema in `dryrun` mode, and displays the load summary

In `dryrun` mode, Auto Parallel Load sends the load script to the `heatwave_autopilot_report` table only. See Section 5.8.8, “Autopilot Report Table”. It does not load data into MySQL HeatWave.

If Auto Parallel Load fails with an error, inspect the errors with a query to the `heatwave_autopilot_report` table.

```
mysql> SELECT log FROM sys.heatwave_autopilot_report
      WHERE type="error";
```

When Auto Parallel Load finishes running, query the `heatwave_autopilot_report` table to check for warnings.

```
mysql> SELECT log FROM sys.heatwave_autopilot_report
      WHERE type="warn";
```

Issue the following query to inspect the load script that was generated.

```
mysql> SELECT log->>"$.sql" AS "Load Script"
      FROM sys.heatwave_autopilot_report
      WHERE type = "sql" ORDER BY id;
+-----------------------------------------------------------------------------+
| Load Script                                                                 |
+-----------------------------------------------------------------------------+
| SET SESSION innodb_parallel_read_threads = 3;                               |
| ALTER TABLE /*+ AUTOPILOT_DISABLE_CHECK */ `tpch`.`customer` SECONDARY_LOAD; |
| SET SESSION innodb_parallel_read_threads = 4;                               |
| ALTER TABLE /*+ AUTOPILOT_DISABLE_CHECK */ `tpch`.`lineitem` SECONDARY_LOAD; |
| SET SESSION innodb_parallel_read_threads = 1;                               |
| ALTER TABLE /*+ AUTOPILOT_DISABLE_CHECK */ `tpch`.`nation` SECONDARY_LOAD;   |
| SET SESSION innodb_parallel_read_threads = 4;                               |
| ALTER TABLE /*+ AUTOPILOT_DISABLE_CHECK */ `tpch`.`orders` SECONDARY_LOAD;   |
```

```
| SET SESSION innodb_parallel_read_threads = 4;                                |
| ALTER TABLE /*+ AUTOPILOT_DISABLE_CHECK */ `tpch`.`part` SECONDARY_LOAD;     |
| SET SESSION innodb_parallel_read_threads = 4;                                |
| ALTER TABLE /*+ AUTOPILOT_DISABLE_CHECK */ `tpch`.`partsupp` SECONDARY_LOAD; |
| SET SESSION innodb_parallel_read_threads = 1;                                |
| ALTER TABLE /*+ AUTOPILOT_DISABLE_CHECK */ `tpch`.`region` SECONDARY_LOAD;   |
| SET SESSION innodb_parallel_read_threads = 1;                                |
| ALTER TABLE /*+ AUTOPILOT_DISABLE_CHECK */ `tpch`.`supplier` SECONDARY_LOAD; |
+------------------------------------------------------------------------------+
16 rows in set (0.3680 sec)
```

Once you are satisfied with the Auto Parallel Load `CALL` statement and the generated load script, reissue the `CALL` statement in `normal` mode to load the data into MySQL HeatWave. For example:

```
mysql> CALL sys.heatwave_load(JSON_ARRAY("tpch"), JSON_OBJECT("mode","normal"));
```

Note

Retrieve DDL statements in a table or use the following statements to produce a list of DDL statements to easily copy and paste.

```
mysql> SET SESSION group_concat_max_len = 1000000;
mysql> SELECT GROUP_CONCAT(log->>"$.sql" SEPARATOR ' ')
      FROM sys.heatwave_autopilot_report
      WHERE type = "sql" ORDER BY id;
*************************** 1. row ***************************
GROUP_CONCAT(log->>"$.sql" SEPARATOR ' '): SET SESSION innodb_parallel_read_threads =
ALTER TABLE /*+ AUTOPILOT_DISABLE_CHECK */ `tpch`.`customer` SECONDARY_LOAD; SET SESS
ALTER TABLE /*+ AUTOPILOT_DISABLE_CHECK */ `tpch`.`lineitem` SECONDARY_LOAD; SET SESS
ALTER TABLE /*+ AUTOPILOT_DISABLE_CHECK */ `tpch`.`nation` SECONDARY_LOAD; SET SESSIO
ALTER TABLE /*+ AUTOPILOT_DISABLE_CHECK */ `tpch`.`orders` SECONDARY_LOAD; SET SESSIO
ALTER TABLE /*+ AUTOPILOT_DISABLE_CHECK */ `tpch`.`part` SECONDARY_LOAD; SET SESSION
ALTER TABLE /*+ AUTOPILOT_DISABLE_CHECK */ `tpch`.`partsupp` SECONDARY_LOAD; SET SESS
ALTER TABLE /*+ AUTOPILOT_DISABLE_CHECK */ `tpch`.`region` SECONDARY_LOAD; SET SESSIO
ALTER TABLE /*+ AUTOPILOT_DISABLE_CHECK */ `tpch`.`supplier` SECONDARY_LOAD;
1 row in set (0.2575 sec)
```

The time required to load data depends on the data size. Auto Parallel Load provides an estimate of the time required to complete the load operation.

Tables are loaded in sequence, ordered by schema and table name. Load-time errors are reported as they are encountered. If an error is encountered while loading a table, the operation is not terminated. Auto Parallel Load continues running, moving on to the next table.

When Auto Parallel Load finishes running, it checks if tables are loaded and shows a summary with the number of tables that were loaded and the number of tables that failed to load.

Memory Estimation for String Column Encoding

The `auto_enc` option is run in `check` mode by default to ensure that there is enough memory for string column encoding.

The following example uses the `auto_enc` option in `check` mode, if you want to ensure that there is sufficient memory for string column encoding before attempting a load operation. Insufficient memory can cause a load failure.

```
mysql> CALL sys.heatwave_load(JSON_ARRAY("tpch"),
      JSON_OBJECT("mode","dryrun","auto_enc",JSON_OBJECT("mode","check")));
+-----------------------------------------------------------------------------------------------------------------
| CAPACITY ESTIMATION
+-----------------------------------------------------------------------------------------------------------------
| Default encoding for string columns: VARLEN (unless specified in the schema)
| Estimating memory footprint for 1 schema(s)
|
|                                TOTAL       ESTIMATED       ESTIMATED       TOTAL     DICTIONARY
| SCHEMA                     OFFLOADABLE   HEATWAVE NODE      MYSQL NODE      STRING        ENCODED     E
| NAME                            TABLES       FOOTPRINT       FOOTPRINT     COLUMNS        COLUMNS     C
```

```
| ------                    -----------       ---------       ---------     -------     ----------     -----
| `tpch`                              8        1.79 GiB        1.15 GiB          29             24
|
| Sufficient MySQL host memory available to load all tables.
| Sufficient HeatWave cluster memory available to load all tables.
|
+-------------------------------------------------------------------------------------------------------------
12 rows in set (0.2401 sec)

+-------------------------------------------------------------------------------------------------------------
| LOAD SCRIPT GENERATION
+-------------------------------------------------------------------------------------------------------------
| Dryrun mode only generates the load script
| Set mode to "normal" in options to load tables
|
| Retrieve load script containing 16 generated DDL commands using the query below:
| Deprecation Notice: "heatwave_load_report" will be deprecated, please switch to "heatwave_autopilot_repor
|   SELECT log->>"$.sql" AS "Load Script" FROM sys.heatwave_autopilot_report WHERE type = "sql" ORDER BY id
|
| Applying changes will take approximately 1.25 min
|
| Caution: Executing the generated load script may alter column definitions and secondary engine flags in t
|
| Total errors encountered: 0
| Total warnings encountered: 0
|
+-------------------------------------------------------------------------------------------------------------
14 rows in set (0.2401 sec)
```

Note

The `auto_enc` option runs in `check` mode regardless of whether it is specified explicitly in the Auto Parallel Load call statement.

Look for capacity estimation data in the Auto Parallel Load output. The results indicate whether there is sufficient memory to load all tables.

Auto Parallel Load Report Table

When MySQL runs Auto Parallel Load, it sends the output including execution logs and a generated load script to the `MySQL HeatWave_load_report` table in the `sys` schema.

As of MySQL 8.0.32, the `heatwave_load_report` table is deprecated, and replaced with heatwave_autopilot_report table in the `sys` schema.

What's Next

- Learn how to load data manually.

4.2.6 Load Data Manually

You can manually load data from InnoDB tables in your DB System into MySQL HeatWave Cluster by using SQL data definition language (DDL) such as `CREATE TABLE` and `ALTER TABLE` statements.

Perform the following steps sequentially for each table that you want to load:

- Exclude table columns with unsupported data types or columns not used in queries
- Define a secondary engine on the table.
- Load the data from the table or table partitions.

Note

Loading partitions is available as of MySQL 9.1.0.

MySQL version 8.2.0 or higher has a guided load feature which allows you to skip the first two steps and load the data from a table or one or partitions of a table directly with a single `ALTER TABLE` statement.

4.2.6.1 Exclude Table Columns

Before loading a table into MySQL HeatWave, exclude the columns with unsupported data types. Otherwise, the table cannot be loaded. For a list of data types that MySQL HeatWave supports, see Section 4.2.1, “Supported Data Types for MySQL HeatWave”.

With MySQL version 8.2.0 or higher, this step is not required as the guided load feature automatically excludes all unsupported columns when you load the table.

Optionally, exclude columns that are not relevant to the intended queries. Excluding irrelevant columns is not required but doing so reduces load time and the amount of memory required to store table data.

Before You Begin

- Import Data into MySQL HeatWave DB System.

Excluding Table Columns

The `NOT SECONDARY` column attribute prevents a column from being loaded into MySQL HeatWave when executing a table load operation. To exclude a column, specify the `NOT SECONDARY` column attribute in an `ALTER TABLE` or `CREATE TABLE` statement, as shown below.

```
mysql> ALTER TABLE tbl_name MODIFY column_name datatype NOT SECONDARY;
```

```
mysql> CREATE TABLE tbl_name (
column_1 datatype,
column_2 datatype,
...
column_n datatype NOT SECONDARY);
```

The following example creates a table `orders` with 2 columns `order_ID` (`INT` datatype) and `Description` (`BLOB` datatype)

```
mysql> CREATE TABLE orders (
order_ID  INT,
Description BLOB);
```

`BLOB` is not supported by MySQL HeatWave and hence you can exclude this column while loading to MySQL HeatWave, using the following statement :

```
mysql> ALTER TABLE orders MODIFY Description BLOB NOT SECONDARY;
```

Note

If a query accesses a column defined with the `NOT SECONDARY` attribute, the query is executed on the DB System by default.

To include the `Description` column that was previously excluded, remove the `NOT SECONDARY` column attribute using the `ALTER TABLE` statement and reload the table.

What's Next

- Learn how to define secondary engine.

4.2.6.2 Define the Secondary Engine

Before You Begin

- Exclude table columns.

Defining Secondary Engine

For each table that you want to load into MySQL HeatWave Cluster, you must define the MySQL HeatWave query processing engine (RAPID) as the secondary engine for the table. To define RAPID as the secondary engine, specify the SECONDARY_ENGINE table option in an ALTER TABLE or CREATE TABLE statement:

```
mysql> CREATE TABLE customer(
 cust_id INTEGER primary key,
 total_items INTEGER NOT NULL,
 total_cost DECIMAL (4,2) NOT NULL,
 order_status CHAR(1) NOT NULL);
```

```
mysql> ALTER TABLE customer SECONDARY_ENGINE = RAPID;
```

This example creates a new table customer with the specified column names. Then, the MySQL HeatWave query processing engine (RAPID) is set as the secondary engine for the table, using ALTER TABLE statement.

With MySQL version 8.2.0 or higher, this step is not required as the guided load feature automatically define the secondary engine when you load a table into MySQL HeatWave cluster.

What's Next

- Learn how to load tables.
- Learn how to load partitions.

4.2.6.3 Load Tables

You can load tables into HeatWave using the SECONDARY_LOAD clause.

Before You Begin

- Define the secondary engine.

Loading Tables Statement

To load a table into MySQL HeatWave, specify the SECONDARY_LOAD clause in an ALTER TABLE statement and use the following syntax.

```
ALTER TABLE table_name SECONDARY_LOAD [PARTITION (...)] [GUIDED {ON | OFF}];
```

Specify the following clauses in the ALTER TABLE statement:

- table_name: Specify the name of the table to load to MySQL HeatWave.
- PARTITION: Available as of MySQL 9.1.0. To learn more about partitions, see Load Partitions. If including partitions in the command, add them as needed.
- GUIDED: Available as of MySQL 9.4.1. Allows you to disable and enable Guided Load as needed. Set to OFF to disable Guided Load. By default, Guided Load is enabled.

The SECONDARY_LOAD clause has these properties:

- Data is read using the READ COMMITTED isolation level.

Guided Load

Available as of MySQL 8.2.0, the Guided Load feature performs a set of checks and validations before loading data.

These checks include the following:

- Automatically exclude tables and columns that cannot be loaded and set them with the NOT SECONDARY attribute.

- Automatically set `SECONDARY_ENGINE` to `rapid`.

As of MySQL 9.4.1, you have the option to disable Guided Load if you want to skip these checks.

Loading Data Examples

The following example loads the `airport_geo`. The Guided Load feature is enabled by default.

```
mysql> ALTER TABLE airport_geo SECONDARY_LOAD;

Warning (code 3877): Command executed during preprocessing: 'ALTER TABLE `airportdb`.`airport_geo` MODI
Warning (code 3877): Command executed during preprocessing: 'ALTER TABLE `airportdb`.`airport_geo` SECO
Warning (code 3877): 1 InnoDB parallel read thread was requested to load airportdb.airport_geo (session
```

The following example loads the `airport_geo` and disables Guided Load.

```
mysql> ALTER TABLE airport_geo SECONDARY_LOAD GUIDED OFF;
```

What's Next

- Learn how to load partitions.

4.2.6.4 Load Partitions

As of MySQL 9.1.0, MySQL HeatWave supports partitions for DB System tables. See Partition Selection.

Before You Begin

- Define the secondary engine.

Loading Partitions using Secondary _Load Clause

To load partitions into MySQL HeatWave, specify the `SECONDARY_LOAD` clause in an `ALTER TABLE` statement with the `PARTITION` clause.

```
mysql> ALTER TABLE tbl_name SECONDARY_LOAD PARTITION (p0, p1, ..., pn);
```

The `SECONDARY_LOAD` clause can load a subset of partitions:

```
mysql> ALTER TABLE tbl_name SECONDARY_LOAD PARTITION (p4, p5, p10);
```

To load a new partition in MySQL HeatWave after adding it to a previously loaded table:

```
mysql> ALTER TABLE t1 SECONDARY_LOAD;
mysql> ALTER TABLE t1 ADD PARTITION (PARTITION p4 VALUES LESS THAN (2002));
mysql> ALTER TABLE t1 SECONDARY_LOAD PARTITION (p4);
```

The `SECONDARY_LOAD` clause has these properties:

- Data is read using the `READ COMMITTED` isolation level.

  With the guided load feature, the `SECONDARY_LOAD` clause also define the `SECONDARY_ENGINE` of the table to `rapid` and marked all columns with unsupported data types with the `NOT SECONDARY` attribute.

See: Section 11.2.1.7, “Partition Selection Limitations”.

What's Next

- Learn about change propagation.

4.2.7 About Change Propagation

After tables are loaded into MySQL HeatWave, data changes are automatically propagated from DB System tables to their counterpart tables in the MySQL HeatWave Cluster.

DML operations, `INSERT`, `UPDATE`, and `DELETE`, on the DB System do not wait for changes to be propagated to the MySQL HeatWave Cluster; that is, DML operations on the DB System are not delayed by MySQL HeatWave change propagation.

Data changes on the DB System node are propagated to MySQL HeatWave in batch transactions. Change propagation is initiated as follows:

- Every 200ms.
- When the change propagation buffer reaches its 64MB capacity.
- When data updated by DML operations on the DB System are read by a subsequent MySQL HeatWave query.

Before You Begin

- Import Data into MySQL HeatWave DB System.

Verify Change Propagation

To check if change propagation is enabled globally, query the `rapid_change_propagation_status` variable:

```
mysql> SELECT VARIABLE_VALUE
        FROM performance_schema.global_status
        WHERE VARIABLE_NAME = 'rapid_change_propagation_status';
+----------------+
| VARIABLE_VALUE |
+----------------+
| ON             |
+----------------+
```

To check if change propagation is enabled for individual tables, query the `POOL_TYPE` data in MySQL HeatWave Performance Schema tables.

- As of MySQL 9.2.1, `TRANSACTIONAL` indicates that change propagation is enabled for the table. `SNAPSHOT` indicates that change propagation is disabled.
- In previous versions of MySQL, `RAPID_LOAD_POOL_TRANSACTIONAL` indicates that change propagation is enabled for the table. `RAPID_LOAD_POOL_SNAPSHOT` indicates that change propagation is disabled.

```
mysql> SELECT NAME, POOL_TYPE
          FROM rpd_tables,rpd_table_id
          WHERE rpd_tables.ID = rpd_table_id.ID AND SCHEMA_NAME LIKE 'tpch';
+---------------+---------------+
| NAME          | POOL_TYPE     |
+---------------+---------------+
| tpch.orders   | TRANSACTIONAL |
| tpch.region   | TRANSACTIONAL |
| tpch.lineitem | TRANSACTIONAL |
| tpch.supplier | TRANSACTIONAL |
| tpch.partsupp | TRANSACTIONAL |
| tpch.part     | TRANSACTIONAL |
| tpch.customer | TRANSACTIONAL |
+---------------+---------------+
```

See Section 11.2.1.3, “Change Propagation Limitations”.

Change Propagation Failure

A change propagation failure can cause tables in MySQL HeatWave to become stale, and queries that access stale tables are not offloaded to MySQL HeatWave for processing.

Tables that have become stale due to change propagation failures resulting from out-of-code errors are automatically reloaded. A check for stale tables is performed periodically when the MySQL HeatWave Cluster is idle.

If change propagation failure has occurred for some other reason causing a table to become stale, you must unload and reload the table manually to restart change propagation for the table. See Section 4.4.2, “Unload Data Manually”, and Section 4.2, “Load Data from DB System into MySQL HeatWave Cluster”.

What's Next

- Learn how to reload tables.

4.2.8 Reload Tables

As of MySQL 8.4.0, MySQL HeatWave lets you reload all tables using the `HEATWAVE_RELOAD` routine.

Before You Begin

- Complete the steps to load data using Auto Parallel Load or manually.

Reload Tables

- Reload tables with default routine option values:

```
mysql> CALL sys.HEATWAVE_RELOAD(NULL);
```

```
mysql> CALL sys.HEATWAVE_RELOAD(JSON_OBJECT());
```

- Reload all user-loaded and automatically-loaded tables:

```
mysql> CALL sys.HEATWAVE_RELOAD(JSON_OBJECT("only_user_loaded_tables",false));
```

- Reload all user-loaded tables with the `output` option set to `silent`:

```
mysql> CALL sys.HEATWAVE_RELOAD(JSON_OBJECT("output","silent"));
```

- Reload all user-loaded and automatically-loaded tables with the `output` option set to `silent`:

```
mysql> CALL sys.HEATWAVE_RELOAD(JSON_OBJECT("only_user_loaded_tables",false,"output","silent"));
```

To learn more about user-loaded and automatically-loaded tables, see Automatic Loading and Unloading of DB System Tables and Partitions.

What's Next

- Perform Transactional and Analytical Processing Using MySQL HeatWave.
- Learn how to use the Machine Learning capabilities of MySQL HeatWave.
- Learn how to Unload Data.

4.3 Load Data from Object Storage into MySQL HeatWave Cluster

The sections in this topic describe how to load and manage data from external data sources using MySQL HeatWave Lakehouse. It also describes how to export data to external sources.

4.3.1 About MySQL HeatWave Lakehouse

The Lakehouse feature of MySQL HeatWave enables query processing on data in Object Storage. MySQL HeatWave Lakehouse reads the source data from Object Storage, transforms it to the memory optimized MySQL HeatWave format, saves it in the MySQL HeatWave persistence storage layer in Object Storage, and then loads the data to MySQL HeatWave Cluster memory. While Lakehouse provides in-memory query processing on data in Object Storage, it does not load data into a DB System table.

To learn how to use Lakehouse with MySQL HeatWave AutoML, see Use MySQL HeatWave AutoML with Lakehouse.

External Tables

When MySQL HeatWave loads data and creates a table from a file in Object Storage, it is an external Lakehouse table. External tables are tables that do not store any data, but refer to data stored externally in Object Storage. The external table stores the location of the data.

See the following to learn more:

- Review Supported File Formats of external files for loading external tables.
- Review Access Object Storage to learn about the different ways Lakehouse can access external data.

Lakehouse Engine

The `lakehouse` storage engine allows you to create tables that point to external data sources.

For MySQL HeatWave Lakehouse, `lakehouse` is the primary engine, and `rapid` is the secondary engine.

As of MySQL 9.4.0, you can use the `CREATE EXTERNAL TABLE` statement.

For example:

```
mysql> CREATE EXTERNAL TABLE table_1 (col_1 INT)
```

In versions earlier than MySQL 9.4.0, you must use the `CREATE TABLE` statement, and manually set `ENGINE` to `lakehouse`, and `SECONDARY_ENGINE` to `rapid`.

For example:

```
mysql> CREATE TABLE table_1 (col_1 INT)
ENGINE=LAKEHOUSE
SECONDARY_ENGINE=RAPID;
```

Data Storage

When using Lakehouse, you do not directly access the source data stored in Object Storage. This memory optimized internal version of the data is retained only as long as MySQL HeatWave Cluster has the Lakehouse option enabled.

The data in external tables is deleted if you do any of the following:

- Drop the external table. See DROP TABLE Statement.
- Unload the external table. See Unload Tables.
- Delete the MySQL HeatWave Cluster. See Deleting a MySQL HeatWave Cluster in the *MySQL HeatWave on OCI Service Guide*, Deleting a MySQL HeatWave Cluster in the *MySQL HeatWave on AWS Service Guide*, or Deleting a Database in the *MySQL HeatWave for Azure Service Guide*.

Workflows

You have three options to load data into MySQL HeatWave Cluster: uniform resource identifiers (URIs) (as of MySQL 9.3.1), resource principals, and pre-authenticated requests (PAR). Refer to the following workflows and resources showing how to use each method to load data.

The workflow to use URIs:

1. Prepare the files to load in the proper format. See Supported File Formats.

2. Upload the files to load into Object Storage. See Uploading an Object Storage Object to a Bucket in *Oracle Cloud Infrastructure Documentation*.
3. Create the external table either using Auto Parallel Load or manually. Review Auto Parallel Load URI examples or manual URI examples.
4. Load the external files into Lakehouse either using Auto Parallel Load or manually.
5. Once Lakehouse successfully creates the external table and loads data from specified files, you can:
   - Use the data in MySQL HeatWave to do the following:
     - Accelerate Query Processing
     - Train and Use Machine Learning Models
     - Perform AI-Powered Search and Content Generation
   - Refresh the loaded data as needed. Use the following methods to refresh data:
     - Refresh Data Using Event-Based Incremental Load
     - Refresh Data Using Incremental Load
     - Add or Remove Files Using Selective Load

The workflow to use Resource Principals:

1. Prepare the files to load in the proper format. See Supported File Formats.
2. Upload the files to load into Object Storage. See Uploading an Object Storage Object to a Bucket in *Oracle Cloud Infrastructure Documentation*.
3. Create the external table either using Auto Parallel Load or manually. Review Auto Parallel Load resource principal examples or manual resource principal examples.
4. Load the external files into Lakehouse either using Auto Parallel Load or manually.
5. Once Lakehouse successfully creates the external table and loads data from specified files, you can:
   - Use the data in MySQL HeatWave to do the following:
     - Accelerate Query Processing
     - Train and Use Machine Learning Models
     - Perform AI-Powered Search and Content Generation
   - Refresh the loaded data as needed. Use the following methods to refresh data:
     - Refresh Data Using Event-Based Incremental Load
     - Refresh Data Using Incremental Load
     - Add or Remove Files Using Selective Load

The workflow to use PARs:

1. Prepare the files to load in the proper format. See Supported File Formats.
2. Upload the files to load into Object Storage. See Uploading an Object Storage Object to a Bucket in *Oracle Cloud Infrastructure Documentation*.

3. Create the PARs for the files previously uploaded. You have three options to create PARs: through the OCI console, the command line, and API. See Creating a Pre-Authenticated Request in Object Storage in *Oracle Cloud Infrastructure Documentation*.
4. Create the external table either using Auto Parallel Load or manually. Review Auto Parallel Load PAR examples or manual PAR examples.
5. Once Lakehouse successfully creates the external table and loads data from specified files, you can:
   - Use the data in MySQL HeatWave to do the following:
     - Accelerate Query Processing
     - Train and Use Machine Learning Models
     - Perform AI-Powered Search and Content Generation
   - Refresh the loaded data as needed. Use the following methods to refresh data:
     - Refresh Data Using Event-Based Incremental Load
     - Refresh Data Using Incremental Load
     - Add or Remove Files Using Selective Load

What's Next

- Review MySQL HeatWave Requirements and Additional MySQL HeatWave Lakehouse Requirements.

4.3.2 Additional MySQL HeatWave Lakehouse Requirements

Depending on the service you use to access MySQL HeatWave Lakehouse, review the appropriate requirements.

Before You Begin

- Review Requirements for using MySQL HeatWave.

MySQL HeatWave on OCI Lakehouse Requirements

- The DB System must have Lakehouse enabled. See Creating a DB System in the *MySQL HeatWave on OCI Service Guide*.
- The MySQL HeatWave Cluster in the DB System must meet the following requirements:
  - As Of MySQL 8.4.0, the MySQL HeatWave Cluster can use a 32GB or 512GB shape.
  - Before MySQL 8.4.0, the MySQL HeatWave Cluster must use a 512GB shape.
  - See Adding a MySQL HeatWave Cluster in the *MySQL HeatWave on OCI Service Guide*.
- For replicated DB Systems, see Lakehouse Limitations for all File Formats.
- You must have the appropriate privileges to access an Oracle Cloud Infrastructure bucket for storing files that you want to load. See MySQL HeatWave Lakehouse Privileges and Access Object Storage.

MySQL HeatWave on AWS Lakehouse Requirements

- The DB System must have Lakehouse enabled. See Creating a MySQL HeatWave Cluster in the *MySQL HeatWave on AWS Service Guide*.

- The MySQL HeatWave Cluster in the DB System must use a 256GB shape.
- For replicated DB Systems, see Lakehouse Limitations for all File Formats.
- You must have the appropriate privileges to access an AWS S3 bucket for storing files that you want to load.

  See the following to learn more:

  - MySQL HeatWave Lakehouse Privileges.
  - Manage AWS Access and Creating Lakehouse Mapping from the *MySQL HeatWave on AWS Service Guide*.

What's Next

- Review MySQL HeatWave Lakehouse Privileges.

4.3.3 MySQL HeatWave Lakehouse Privileges

Review the following privileges required to use MySQL HeatWave Lakehouse.

Before You Begin

- Review Additional MySQL HeatWave Lakehouse Requirements.

Object Storage Privileges

To access object storage resources and files to load data, you require specific privileges. See the following to learn more:

- PAR Privileges
- Resource Principal Privileges

MySQL HeatWave on AWS Privileges

For MySQL HeatWave on AWS, you require the appropriate privileges to access S3 resources. To learn more, see Manage AWS Access and Creating Lakehouse Mapping from the *MySQL HeatWave on AWS Service Guide*.

What's Next

- To use Auto Parallel Load with Lakehouse, review Requirements and Privileges for Auto Parallel Load.
- To export query results, review Required Privileges to Export Query Results.
- Review Supported File Formats.

4.3.4 Supported File Formats and Data Types

Review the various file formats supported for loading structured, semi-structured, and unstructured data in MySQL HeatWave from different sources.

This topic contains the following sections:

- Before You Begin
- Supported File Formats for Structured and Semi-Structured Data

- Supported File Formats for Unstructured Data
- Parquet File Format
- Avro File Format
- Vector Data Type
- Delta Lake Tables
- What's Next

Before You Begin

- Review the following:
  - Additional MySQL HeatWave Lakehouse Requirements
  - MySQL HeatWave Lakehouse Privileges

Supported File Formats for Structured and Semi-Structured Data

MySQL HeatWave Lakehouse supports loading structured and semi-structured relational data in the following file formats:

- Avro
- CSV
- JSON

  As of MySQL 8.4.0, Lakehouse supports Newline Delimited JSON (ND-JSON).
- Parquet
- Delta lake tables (as of MySQL 9.5.0)

Supported File Formats for Unstructured Data

A Vector store can ingest files from the following unstructured formats: PPT, TXT, HTML, DOC, and PDF. Each file can be upto 100 MB in size.

To load unstructured data, see Ingest Files into a Vector Store.

Parquet File Format

Review the following about Parquet files:

- Parquet Types
- Lakehouse Limitations for the Parquet File Format
- As of MySQL 9.5.0, you can choose how to map columns from Parquet files to external tables by setting the `match_columns_by` option when using external table JSON syntax. You can map columns by column order or by column name (case sensitive or case insensitive). See Lakehouse External Table JSON Syntax to learn more.
- MySQL 9.6.0 introduces support for complex and nested Parquet types, including `LIST`, `MAP`, and other nested structures. Group and nested structures are flattened into a JSON representation.
  - Simple lists are converted to JSON arrays

- Maps and complex nested types are converted to JSON objects.

Review the following table of Parquet data type conversions.

Table 4.3 Parquet Data Type Conversions

MySQL Data Type	Parquet Logical Data Type	Comments
BOOL	STRING	Valid values: TRUE, FALSE, T, F, true, false, 0, 1
BOOL	INT	All signed and unsigned integers with any bit width, but the value must be 0 or 1
BOOL	None	Physical Type BOOLEAN, FLOAT, or DOUBLE.
CHAR, VARCHAR, TEXT	DECIMAL	The maximum precision is 38 and Lakehouse supports the Physical Type FIXED_LEN_BYTE_ARRAY
CHAR, VARCHAR, TEXT	STRING	The character set must not be BINARY
CHAR, VARCHAR, TEXT	INT	All signed and unsigned integers
CHAR, VARCHAR, TEXT	LIST	Must be a supported simple (non-nested) list: a list of numeric type (all signed and unsigned integers or Physical Type FLOAT or DOUBLE), STRING or BOOL.
CHAR, VARCHAR, TEXT	None	Physical Type FLOAT or DOUBLE
DATE	STRING	Only supports the format yyyy-mm-dd
DATE	DATE	
DATE	TIMESTAMP	isAdjustedToUTC = false, timeUnit = x: Lakehouse only loads the date portion
DATE	None	Physical Type INT96: Lakehouse only loads the date portion
DATETIME	STRING	DATETIME accepts the range from 0000-00-00 to 9999-12-12 The maximum precision is six digits. Values with more precision are truncated.
DATETIME	DATE	DATETIME accepts the range from 0000-00-00 to 9999-12-12 The maximum precision is six digits. Values with more precision are truncated.
DATETIME	TIME	isAdjustedToUTC = false, timeUnit = x: DATETIME accepts the range from 0000-00-00 to 9999-12-12

MySQL Data Type	Parquet Logical Data Type	Comments
		The maximum precision is six digits. Values with more precision are truncated.
DATETIME	TIMESTAMP	isAdjustedToUTC = false, timeUnit = x: DATETIME accepts the range from 0000-00-00 to 9999-12-12 The maximum precision is six digits. Values with more precision are truncated.
DATETIME	None	Physical Type INT96: DATETIME accepts the range from 0000-00-00 to 9999-12-12 The maximum precision is six digits. Values with more precision are truncated.
DECIMAL	DECIMAL	The maximum precision is 38 and Lakehouse supports the Physical Type FIXED_LEN_BYTE_ARRAY
DECIMAL	STRING	
DECIMAL	INT	All signed and unsigned integers
DECIMAL	None	Physical Type FLOAT or DOUBLE
DOUBLE, FLOAT	DECIMAL	The maximum precision is 38 and Lakehouse supports the Physical Type FIXED_LEN_BYTE_ARRAY
DOUBLE, FLOAT	STRING	
DOUBLE, FLOAT	INT	All signed and unsigned integers
DOUBLE, FLOAT	None	Physical Type FLOAT or DOUBLE
ENUM	STRING	
ENUM	ENUM	Physical Type BYTE_ARRAY or FIXED_LEN_BYTE_ARRAY. Must be UTF-8 encoded.
ENUM	INT	All signed and unsigned integers
ENUM	None	Physical Type FLOAT or DOUBLE
JSON	STRING	
JSON	JSON	For JSON and the non-UTF-8 encoding byte array, the value should be JSON binary (representative of MySQL JSON).
JSON	LIST	
JSON	MAP	The key type should be STRING (UTF-8)
JSON	Nested/Group Nodes	

MySQL Data Type	Parquet Logical Data Type	Comments
JSON	None	Physical Type BYTE_ARRAY, Binary encoded. The non-UTF-8 encoding should be JSON binary, see: The JSON Data Type.
TIME	STRING	Only supports format hh:mm:ss or hh:mm::ss.xxx The maximum precision is six digits. Values with more precision are truncated.
TIME	TIME	isAdjustedToUTC = false, timeUnit = x The maximum precision is six digits. Values with more precision are truncated.
TIMESTAMP	STRING	TIMESTAMP accepts the range from 1970-01-01 to 2038-01-19. Lakehouse might not load out of range values correctly. Use DATETIME instead. The maximum precision is six digits. Values with more precision are truncated.
TIMESTAMP	DATE	TIMESTAMP accepts the range from 1970-01-01 to 2038-01-19. Lakehouse might not load out of range values correctly. Use DATETIME instead. The maximum precision is six digits. Values with more precision are truncated.
TIMESTAMP	TIME	isAdjustedToUTC = false, timeUnit = x: TIMESTAMP accepts the range from 1970-01-01 to 2038-01-19. Lakehouse might not load out of range values correctly. Use DATETIME instead. The maximum precision is six digits. Values with more precision are truncated.
TIMESTAMP	TIMESTAMP	isAdjustedToUTC = false, timeUnit = x: TIMESTAMP accepts the range from 1970-01-01 to 2038-01-19. Lakehouse might not load out of range values correctly. Use DATETIME instead.

MySQL Data Type	Parquet Logical Data Type	Comments
		The maximum precision is six digits. Values with more precision are truncated.
`TIMESTAMP`	`None`	Physical Type `INT96`: `TIMESTAMP` accepts the range from 1970-01-01 to 2038-01-19. Lakehouse might not load out of range values correctly. Use `DATETIME` instead. The maximum precision is six digits. Values with more precision are truncated.
All signed and unsigned `TINYINT`, `SMALLINT`, `MEDIUMINT`, `INTEGER`, `INT`, `BIGINT`	`DECIMAL`	The maximum precision is 38 and Lakehouse supports the Physical Type `FIXED_LEN_BYTE_ARRAY`
All signed and unsigned `TINYINT`, `SMALLINT`, `MEDIUMINT`, `INTEGER`, `INT`, `BIGINT`	`STRING`	
All signed and unsigned `TINYINT`, `SMALLINT`, `MEDIUMINT`, `INTEGER`, `INT`, `BIGINT`	`INT`	All signed and unsigned integers
All signed and unsigned `TINYINT`, `SMALLINT`, `MEDIUMINT`, `INTEGER`, `INT`, `BIGINT`	`None`	Physical Type `FLOAT` or `DOUBLE`
`VECTOR`	`STRING`	Should be a proper string-formatted vector: comma-separated numeric values contained within square brackets (for example, [1.1, 2.2, -3.3, 4e-4]). The number of values should be within the limits of the vector's dimension, and the individual values should be in the range of float (32-bits).
`VECTOR`	`LIST`	Must be a simple (non-nested) list of numeric type (All signed and unsigned integers or Physical Type `FLOAT` or `DOUBLE`). The list lengths should be within the limits of the vector's dimension, and the values should be in the range of float (32-bits).
`YEAR`	`STRING`	The value should be in the range `0` to `99` or the range `1901` to `2155`

MySQL Data Type	Parquet Logical Data Type	Comments
YEAR	INT	The value should be in the range 0 to 99 or the range 1901 to 2155
YEAR	None	Physical Type FLOAT or DOUBLE: The value should be in the range 0 to 99 or the range 1901 to 2155
YEAR	None	Physical Type INT96: The value should be in the range 1901 to 2155, Lakehouse only loads the year portion
YEAR	DATE	The value should be in the range 1901 to 2155, Lakehouse only loads the year portion
YEAR	TIMESTAMP	isAdjustedToUTC = false: The value should be in the range 1901 to 2155, Lakehouse only loads the year portion

Avro File Format

Review the following about Avro files:

- Avro Specification
- Lakehouse Limitations for the Avro File Format
- As of MySQL 9.7.0, you can choose how to map columns from Avro files to external tables by setting the match_columns_by option when using external table JSON syntax. You can map columns by column order or by column name (case sensitive or case insensitive). See Column Matching Options to learn more.

Review the following table of Avro data type conversions.

Table 4.4 Avro Data Type Conversions

MySQL Data Type	Avro Data Type	Avro Logical Data Type	Comments
BOOL	Boolean	None	Stored in MySQL as TINYINT
BOOL	INT, LONG	None	
BOOL	Float, Double	None	
BOOL	String	None	Can use TRUE/FALSE, true/false, or 0/1.
BOOL	Enum	None	
BOOL	Fixed, Bytes	Decimal	
TINYINT, SMALLINT, MEDIUMINT, INTEGER, INT, BIGINT	Boolean	None	
TINYINT, SMALLINT, MEDIUMINT, INTEGER, INT, BIGINT	Int, Long	None	Attempts narrowing/widening conversion. An error occurs if there is an overflow.

MySQL Data Type	Avro Data Type	Avro Logical Data Type	Comments
TINYINT, SMALLINT, MEDIUMINT, INTEGER, INT, BIGINT	Float, Double	None	
TINYINT, SMALLINT, MEDIUMINT, INTEGER, INT, BIGINT	String	None	Attempts parsing of numbers. An error occurs if this is unsuccessful.
TINYINT, SMALLINT, MEDIUMINT, INTEGER, INT, BIGINT	Enum	None	
TINYINT, SMALLINT, MEDIUMINT, INTEGER, INT, BIGINT	Fixed, Bytes	Decimal	
CHAR, VARCHAR, TEXT	String	None	
CHAR, VARCHAR, TEXT	String	UUID	Treated as a regular string.
CHAR, VARCHAR, TEXT	Enum		
CHAR, VARCHAR, TEXT	Fixed, Bytes	Decimal	The maximum number of digits when converting to string is 1000 digits. If the string representation of the decimal exceeds the maximum, the string is truncated with an error/ warning.
DOUBLE, FLOAT, REAL	Boolean	None	
DOUBLE, FLOAT, REAL	Int, Long	None	
DOUBLE, FLOAT, REAL	Float, Double	None	A warning displays about loss of precision when converting Double to Float.
DOUBLE, FLOAT, REAL	String	None	Attempts parsing of numbers. An error occurs if this is unsuccessful.
DOUBLE, FLOAT, REAL	Enum	None	
DOUBLE, FLOAT, REAL	Fixed, Bytes	Decimal	
ENUM	Int, Long	None	
ENUM	String	None	
ENUM	Enum	None	
DECIMAL	Int, Long	None	
DECIMAL	Float, Double	None	
DECIMAL	String	None	Attempts parsing of numbers. An error occurs if this is unsuccessful.

MySQL Data Type	Avro Data Type	Avro Logical Data Type	Comments
`DECIMAL`	`Fixed`, `Bytes`	`Decimal`	Errors/Warnings occur for the following: • If the MySQL maximum scale of 30 is exceeded, the decimal is truncated. • If the MySQL maximum precision of 65 is exceeded by the number of digits before the decimal place, the value is set to maximum representable, with a report for the overflow. • If the MySQL maximum precision of 65 is exceeded, but the number of digits before the decimal place is fewer than 65, then digits after the decimal place are truncated as needed.
`TIME`	`Int`	`Time-Millis`	Value is interpreted as milliseconds since midnight, and corresponds exactly to `TIME` (3). The maximum precision is six digits. Values with more precision are truncated.
`TIME`	`Long`	`Time-Micros`	Value is interpreted as milliseconds since midnight, and corresponds exactly to `TIME` (6). The maximum precision is six digits. Values with more precision are truncated.
`TIME`	`String`	None	The maximum precision is six digits. Values with more precision are truncated.

MySQL Data Type	Avro Data Type	Avro Logical Data Type	Comments
DATETIME, TIMESTAMP	Long	Timestamp-Millis	Value is interpreted as milliseconds since January, 1, 1970. The maximum precision is six digits. Values with more precision are truncated.
DATETIME, TIMESTAMP	Long	Timestamp-Micros	Value is interpreted as microseconds since January, 1, 1970. The maximum precision is six digits. Values with more precision are truncated.
DATETIME, TIMESTAMP	Int	Date	The days since January 1, 1970. The maximum precision is six digits. Values with more precision are truncated.
DATETIME, TIMESTAMP	String	None	The maximum precision is six digits. Values with more precision are truncated.
DATE	Long	Timestamp-Millis	Value is interpreted as milliseconds since January, 1, 1970.
DATE	Long	Timestamp-Micros	Value is interpreted as microseconds since January, 1, 1970.
DATE	Int	Date	The days since January 1, 1970.
DATE	String	None	
YEAR	Long	Timestamp-Millis	Value is interpreted as milliseconds since January, 1, 1970.
YEAR	Long	Timestamp-Micros	Value is interpreted as microseconds since January, 1, 1970.
YEAR	Int	Date	The days since January 1, 1970.
YEAR	String	None	

Vector Data Type

As of MySQL 9.3.2, Lakehouse supports the VECTOR data type for CSV and Parquet files.

- For CSV files, Lakehouse processes the data as string-formatted vectors.

- For Parquet files, Lakehouse processes the data as either string-formatted vectors or in the Parquet `List` type.

Vectors are created using an embedding model. When creating a table with the `VECTOR` data type, you can provide the model to use in the `ENGINE_ATTRIBUTE` for either the column or table.

- To set the embedding model for all `VECTOR` columns in the table, load the data automatically and set the `dialect` parameter `embed_model_id` accordingly.
- To set the embedding model for the column, load the data manually and set the `ENGINE_ATTRIBUTE` parameter `model` accordingly. If you set the embedding model for the table, that model is set for all columns in the table.

Review the current list of MySQL HeatWave In-Database Embedding Models.

The following example loads a table using Lakehouse Auto Parallel Load and sets the embedding model for the table to `minilm`:

```
mysql> SET @input_list = '[{
  "db_name": "data_db",
  "tables": [{
    "table_name": "table_1",
    "engine_attribute": {
      "dialect": {"format": "csv", "embed_model_id": "minilm"},
      "file": [{"uri": "oci://mybucket@mynamespace/data_files/data_file_1.csv"}]
    }
  }]
}]';
mysql> CALL sys.HEATWAVE_LOAD(CAST(@input_list AS JSON), NULL);
```

The following example manually loads a table and sets the embedding model for the column to `multilingual-e5-small`. If you are on a version earlier than MySQL 9.4.0, you must use `CREATE TABLE` and set `ENGINE` to `lakehouse`, and `SECONDARY_ENGINE` to `rapid`. See CREATE TABLE Statement with JSON syntax. You cannot set the embedding model with SQL syntax for the `CREATE TABLE` statement.

```
mysql> CREATE EXTERNAL TABLE table_1(col_1 VECTOR(5) ENGINE_ATTRIBUTE '{"model": "minilm"}')
        FILE_FORMAT = (FORMAT csv)
        FILES = (URI = 'oci://mybucket@mynamespace/data_files/data_file_1.csv');
mysql> ALTER TABLE table_1 SECONDARY_LOAD;
```

Delta Lake Tables

As of MySQL 9.5.0 (supported in MySQL HeatWave on OCI only), you have the option to load delta lake tables.

Review Limitations for Delta Lake Tables.

When loading delta lake tables, keep in mind the following:

- You must specify an Object Storage external file location with a prefix (a trailing `/`). If the specified file location does not have a trailing `/` character, Lakehouse implies one and adds it to the file location. Specifying file names or file patterns is not supported.
- For pre-authenticated requests (PAR), you must specify a bucket PAR or prefix PAR. Specifying an object PAR is not supported.
- You must only specify one file location. Specifying multiple file locations generates an error.
- During schema inference, partitioning columns are set as `NOT SECONDARY`.
- You have the option to incrementally refresh external tables loaded with delta lake tables. See Update Tables to learn more.

Review the following table of supported delta lake data type conversions. To review unsupported delta lake data types, see Limitations for Delta Lake Tables.

Table 4.5 Delta Lake Data Type Conversions

MySQL Data Type	Delta Lake Logical Data Type
LONGTEXT	STRING
TINYINT	TINYINT
SMALLINT	SMALLINT
INT	INT
BIGINT	BIGINT
DECIMAL	DECIMAL
FLOAT	FLOAT
DOUBLE	DOUBLE
TIMESTAMP	TIMESTAMP
DATETIME	TIMESTAMP_NTZ

The following example creates and then loads an external table with a delta lake table by using Auto Parallel Load:

```
mysql> SET @input_list = '[{
  "db_name": "delta_lake_db",
  "tables": [{
    "table_name": "delta_table_1",
    "engine_attribute": {
      "dialect": {"format": "delta"},
      "file": [{"uri": "oci://mybucket@mynamespace/delta_lake/"}]
    }
  }]
}]';
mysql> CALL sys.HEATWAVE_LOAD(CAST(@input_list AS JSON), NULL);
```

The following example manually creates and then loads an external table with a delta lake table:

```
mysql> CREATE EXTERNAL TABLE delta_table_1(col_1 INT)
        FILE_FORMAT = (FORMAT delta)
        FILES = (URI = 'oci://mybucket@mynamespace/delta_lake/');
mysql> ALTER TABLE delta_table_1 SECONDARY_LOAD;
```

What's Next

- Review MySQL HeatWave Lakehouse Limitations for supported file formats.
- Review how to Access Object Storage.
- Learn how to load structured data using Lakehouse Auto Parallel Load or manually.

4.3.5 Access Object Storage

MySQL HeatWave on OCI can access external data stored on Oracle Cloud Infrastructure in Object Storage.

MySQL HeatWave Lakehouse provides three methods to access Object Storage:

- Uniform Resource Identifier (URI) (Available as of MySQL 9.3.1 for MySQL HeatWave on OCI)
- Resource principal
- Pre-authenticated request (PAR)

For accessing external resources on MySQL HeatWave on AWS, see the following from the *MySQL HeatWave on AWS Service Guide*:

- To grant access to an Amazon S3 bucket, see Creating an IAM Role to Access an Amazon S3 Bucket.
- To allow MySQL HeatWave Lakehouse to process data in Amazon S3, see Creating Lakehouse Mapping.

4.3.5.1 Access Object Storage with Uniform Resource Identifiers (URI)

As of MySQL 9.3.1 (supported in MySQL HeatWave on OCI only), MySQL HeatWave Lakehouse lets you use Uniform Resource Identifiers (URI) in a unified way to specify Object Storage resources for loading data into MySQL HeatWave Lakehouse.

Before You Begin

- Review the following:
  - Additional MySQL HeatWave Lakehouse Requirements
  - MySQL HeatWave Lakehouse Privileges
  - Supported File Formats

URI Privileges

To use URIs, you require the existing privileges to use pre-authenticated requests and resource principals. See the following to learn more:

- To use PAR URIs, make sure you have the required PAR Privileges.
- To use Native URIs and OCIFS URIs, make sure you have the required Resource Principal Privileges.

What's Next

- Learn how to do the following:
  - Create External Tables Using Auto Parallel Load. Review Auto Parallel Load URI examples
  - Create External Tables Manually. Review manual URI examples.

4.3.5.2 Access Object Storage with Resource Principals

Resource principals allow you to authenticate and access Oracle Cloud Infrastructure (OCI) Object Storage resources.

You require the appropriate privileges for Resource Principals to do the following:

- Load data using Lakehouse Auto Parallel Load and manually with resource principals.
- Load data using Lakehouse Auto Parallel Load and manually with uniform resource identifiers (URI).
- Export Query Results to Object Storage (as of MySQL 9.1.1-u1)

Before You Begin

- Review the following:
  - Additional MySQL HeatWave Lakehouse Requirements

- MySQL HeatWave Lakehouse Privileges
- Supported File Formats

Resource Principal Privileges

To use resource principals to access Object Storage resources in OCI, you need to properly set up dynamic groups and policies.

Dynamic groups allow you to group DB Systems as principal actors, similar to user groups. You can then create policies to permit DB Systems in these groups to make API calls against services, such as Object Storage. Membership in the group is determined by a set of criteria called matching rules.

Policies define what your groups can and cannot do. For MySQL HeatWave Lakehouse to access Object Storage, you must define a policy which grants the dynamic group's resources access to buckets and their contents in a specific compartment.

To set up resource principals when you have multiple identity domains, add the identity domain associated with the dynamic group as a prefix. To set up resource principals when the DB System is located in a child compartment, provide the OCI ID of the child compartment associated with the DB System.

The following steps show how to set up a dynamic group for all DB Systems in a defined compartment.

1. Create a new dynamic group or update a dynamic group's matching rules, and the following matching rule to include all MySQL HeatWave DB systems in the defined compartment:

   ```
   ALL{resource.type = 'mysqldbsystem', resource.compartment.id = 'ocid1.compartment.oc1..AlphanumericStri
   ```

2. Create the policies as needed for the dynamic group.

   To create a policy to allow access to load data using Lakehouse Auto Parallel Load or manually to Lakehouse with resource principals, add the following policies for the dynamic group:

   ```
   allow dynamic-group IdentityDomainName/GroupName to read buckets in compartment CompartmentName
   allow dynamic-group IdentityDomainName/GroupName to read objects in compartment CompartmentName
   ```

   To create a policy to allow access to export query results to object storage, add the following policies for the dynamic group:

   ```
   allow dynamic-group IdentityDomainName/GroupName to read buckets in compartment CompartmentName
   allow dynamic-group IdentityDomainName/GroupName to manage objects in compartment CompartmentName
   where any {request.permission='OBJECT_CREATE', request.permission='OBJECT_INSPECT', request.permission=
   ```

   Replace the following:

   - *`IdentityDomainName`*: the identity domain name.

     Note

     If the dynamic group belongs to the default identity domain, you can omit specifying the identity domain name.

   - *`GroupName`*: the dynamic group name
   - *`CompartmentID`*: the compartment ID of the DB system

To learn more, see the following:

- Resource Principals (Refer to the sections *Defining a Policy for MySQL HeatWave Lakehouse* and *Defining a Policy for Exporting Query Results to Object Storage*)

- Writing Matching Rules to Define Dynamic Groups

Once you gain the appropriate privileges for resource principals, see the following information on loading data or see Export Query Results to Object Storage.

What's Next

- Learn how to do the following:
  - Create External Tables Using Auto Parallel Load. Review Auto Parallel Load resource principal examples.
  - Create External Tables Manually. Review manual resource principal examples.

4.3.5.3 Access Object Storage with Pre-Authenticated Requests

Pre-authenticated requests (PAR) allow you to access a bucket or an object in Oracle Cloud Infrastructure (OCI) Object Storage without having your own credentials. A pre-authenticated request is a unique URL, and anyone who has this URL can access Object Storage resources with standard HTTP tools like curl and wget.

For more information about pre-authenticated requests, see Object Storage Pre-Authenticated Requests.

Before You Begin

- Review the following:
  - Additional MySQL HeatWave Lakehouse Requirements
  - MySQL HeatWave Lakehouse Privileges
  - Supported File Formats

PAR Privileges

To create pre-authenticated requests to access Object Storage resources, you require the appropriate OCI privileges. See Required Permssions to learn more.

Once you have the required privileges to use PARs, review the following information on using them to access Object Storage resources.

What's Next

- Learn how to do the following:
  - Create External Tables Using Auto Parallel Load. Review Auto Parallel Load PAR examples.
  - Create External Tables Manually. Review manual PAR examples.

4.3.6 Lakehouse External Table Syntax

To set options when creating external tables, you have the following options:

- As of MySQL 9.4.0, you can use SQL syntax to set table-level and column-level options for external tables. Review Lakehouse External Table SQL Syntax.
- You can use table-level and column-level `ENGINE_ATTRIBUTE` options of the `CREATE TABLE` statement to specify the parameters needed to process data stored externally in Object Storage. The external table `ENGINE_ATTRIBUTE` is a `JSON` object that consists of the dialect parameters and the file parameters. Review Lakehouse External Table JSON Syntax.

4.3.6.1 Lakehouse External Table SQL Syntax

As of MySQL 9.4.0, you can set external table options using SQL syntax. The following options apply to `CREATE TABLE` and `ALTER TABLE` statements. For `ALTER TABLE` statements, `ADD COLUMN` is not supported for external tables.

This topic contains the following sections:

- CREATE TABLE Statement
- External Table Options with SQL Syntax
- Column Definitions
- Table Parameters
- File Specifications
- File Format Options
- Additional File Format Options
- Additional Syntax Examples
- What's Next

CREATE TABLE Statement

The following syntax shows how to specify external table options with SQL syntax.

```
CREATE EXTERNAL TABLE table_name
(col_name data_type ['Col_Options_SQL'], ...) Tbl_Options_SQL
[PARTITION BY KEY (partition_column_name)];
```

The `PARTITION BY KEY` clause is optional. To learn more, see Create Tables with Key Partitioning.

After creating a table, you can load data into the table in the following ways:

- Load the data automatically with Lakehouse Auto Parallel Load by using the `HEATWAVE_LOAD` command. See how to Load External Data Using Lakehouse Auto Parallel Load with an Existing Table.
- Load the data manually with the `ALTER TABLE` statement and the option to use Guided Load.

External Table Options with SQL Syntax

```
Tbl_Options_SQL
  (table_option [table_option]...)

table_option: {
  | FILE_FORMAT [=] format_spec
  | FILES [=] (file_spec[, file_spec] ...)
  | ALLOW_MISSING_FILES [=] {0 | 1 | DEFAULT}
  | AUTO_REFRESH_SOURCE [=] {'string' | NONE}
  | VERIFY_KEY_CONSTRAINTS [=] {0 | 1 | DEFAULT}
  | STRICT_LOAD [=] {0 | 1 | DEFAULT}
  }

format_spec: {
  [general_options]
  [field_options]
  [LINES [TERMINATED BY 'string']]
  [IGNORE number {LINES | ROWS}]
}
  general_options: {
    general_option [general_option] ...}
    general_option: {
```

```
        [FORMAT {csv | avro | json | parquet | delta}]
      | [COMPRESSION {zip | gzip | bzip2 | auto}]
      | [CHARACTER SET charset_name]
      | [HEADER {ON | OFF}]
}
  field_options:{
    {FIELDS | COLUMNS} field_option [field_option]...}
    field_option:{
        DATE FORMAT 'date_format'
      | DATETIME FORMAT 'datetime_format'
      | TIME FORMAT 'time_format'
      | ESCAPED BY {'char' | ''}
      | TERMINATED BY
      | {  [OPTIONALLY] ENCLOSED BY {'char' | ''}}
}
file_spec: {
  file_attribute [file_attribute] ...}
  file_attribute: {
      {URI | URL} [=] 'uri_string'
    | FILE_NAME [=] 'object_name'
    | FILE_PATTERN [=] 'pattern'
    | FILE_PREFIX [=] 'prefix_name'
    | ALLOW_MISSING_FILES [=] {0 | 1 | DEFAULT}
    | STRICT_LOAD [=] {0 | 1 | DEFAULT}
}
```

Note

- For a complete list of table options, see `CREATE TABLE`.
- You can enclose options for `CHARACTER SET` with or without quotation characters.
- As of MySQL 9.5.0, you can enclose options for `FORMAT` and `COMPRESSION` with or without quotation characters. In earlier versions, you must not enclose `FORMAT` options with quotation characters, and you must enclose `COMPRESSION` options in quotation characters.

For `FILES` options, you can set the following:

- Uniform Resource Identifiers (URI)

  Available for MySQL HeatWave on OCI. You can provide an OCIFS URI, PAR URI, or Native URI.

- Pre-Authenticated Requests (PAR)

  See Object Storage Pre-Authenticated Requests in *Oracle Cloud Infrastructure Documentation*.

You cannot use resource principals to specify details for Object Storage files. To use resource principals, you must use External Table ENGINE_ATTRIBUTE Options with JSON syntax. Alternatively, you can convert resource principals to URIs.

Column Definitions

For versions earlier than MySQL 9.5.0, you can override some options at the column level by specifying column-level parameters with the `EXTERNAL_FORMAT` statement. For MySQL 9.5.0 and later, you can do this by loading data manually with Guided Load disabled. You can set `DATE`, `TIME`, `DATETIME`, and `TIMESTAMP` as optional column-level parameters.

As of MySQL 9.5.0, Lakehouse Auto Parallel Load and Guided Load can infer supported formats for temporal data types by setting the appropriate dialect-level temporal formats to `auto`. The `auto` setting is the default value if no temporal format is specified. If you set a specific format at the column level for a temporal data type when creating a table, and a different format is inferred when loading the table to Lakehouse, the specific format used for the column when creating the table is prioritized and used for loading the column. See About Lakehouse Auto Parallel Load Schema Inference to learn more.

Note

The temporal column-level parameters of `DATE`, `TIME`, `DATETIME`, and `TIMESTAMP` are only available for CSV files. See File Format Options.

```
  EXTERNAL_FORMAT {'date_format' | 'time_format' | 'timestamp_format'}
```

The following example sets the `date_format`, `time_format`, and `timestamp_format` at the column level.

```
mysql> CREATE EXTERNAL TABLE table_1(
        col_1 date EXTERNAL_FORMAT '%W %M %Y',
        col_2 time EXTERNAL_FORMAT '%H:%i:%s',
        col_3 timestamp EXTERNAL_FORMAT '%Y-%m-%d %H:%i:%s',
        col_4 int)
        FILE_FORMAT = (FORMAT csv)
        FILES = (URI = 'oci://mybucket@mynamespace/data_files/data_file_1.csv');
```

The following example sets the `EXTERNAL_FORMAT` option to `auto`, which allows Lakehouse Auto Parallel Load and Guided Load to infer supported formats for temporal data types.

```
mysql> CREATE EXTERNAL TABLE table_1(
        col_1 date EXTERNAL_FORMAT 'auto',
        col_2 time EXTERNAL_FORMAT 'auto',
        col_3 timestamp EXTERNAL_FORMAT 'auto',
        col_4 int)
        FILE_FORMAT = (FORMAT csv)
        FILES = (URI = 'oci://mybucket@mynamespace/data_files/data_file_1.csv');
```

Table Parameters

The following table parameters are optional and specify options at the table level.

- `ALLOW_MISSING_FILES`: Handles missing files based on the selected option. This overrides the table parameter `STRICT_LOAD` for missing files. The supported options are:
  - `1`: An error does not occur for any missing file, and data loading continues with the existing files. An error occurs if all files are not available.
  - `0`: If any file is missing, an error occurs and data is not loaded.
  - If you set no value or `DEFAULT`, the setting is defined by `STRICT_LOAD`. If no value is set for `STRICT_LOAD`, the setting is defined by the `sql_mode`.
  - Review Table 4.6, “Combinations of Settings for ALLOW_MISSING_FILES and STRICT_LOAD”.

  A missing file is defined as:

  - With the `FILE_NAME` parameter: There is no file with that name.
  - With the `FILE_PATTERN` parameter: There are no files that match the pattern.
  - With the `FILE_PREFIX` parameter: There are no files with that prefix.

  The following example sets `ALLOW_MISSING_FILES` to `1` in the table options. If any of the specified files are not available, an error does not occur and data continues loading for the existing files. If all files are missing, an error occurs. If a file-specific setting is set, this may override the `ALLOW_MISSING_FILES` setting.

```
mysql> CREATE EXTERNAL TABLE table_1(
        col_1 int,
        col_2 int,
        col_3 int)
        FILE_FORMAT = (FORMAT csv)
        FILES = (URI = 'oci://mybucket@mynamespace/data_files/data_file_1.csv',
                 URI = 'oci://mybucket@mynamespace/data_files/data_file_2.csv',
```

```
                URI = 'oci://mybucket@mynamespace/data_files/data_file_3.csv')
        ALLOW_MISSING_FILES = 1;
```

- `AUTO_REFRESH_SOURCE`: Supported as of MySQL 9.4.1. Specifies a Stream OCID for event-based incremental load. This string value is verified as a valid Stream resource identifier. Learn more about Resource Identifiers from *Oracle Cloud Infrastructure Documentation*. Learn how to Refresh Data Using Event-Based Incremental Load.

  The following example specifies a Stream OCID to set up event-based incremental load.

```
mysql> CREATE EXTERNAL TABLE table_1(
        col_1 int,
        col_2 int,
        col_3 int)
        FILE_FORMAT = (FORMAT csv)
        FILES = (URI = 'oci://mybucket@mynamespace/data_files/data_file_1.csv')
        AUTO_REFRESH_SOURCE = 'ocid1.stream.oc1.iad...';
```

- `STRICT_LOAD`: Loads data in strict mode or non-strict mode, based on the selected option. By default, `STRICT_LOAD` is set to the `sql_mode` value. See Strict SQL Mode. The `FILES` parameter `STRICT_LOAD` can override this setting. The supported options are:

  - `1`: Loads the data in strict mode. The data loading stops if there is an error due to missing files, empty columns, formatting errors or parsing errors.
  - `0`: Loads the data in non-strict mode. Missing files, empty columns, formatting errors, or parsing errors display a warning, and data is loaded. If there are missing columns, implicit default values are used in the absence of user-provided default values. See Data Type Default Values to learn more.
  - If you set no value or `DEFAULT`, the setting is defined by the `sql_mode`.
  - Review Table 4.6, “Combinations of Settings for ALLOW_MISSING_FILES and STRICT_LOAD”.

  This setting overrides the global `sql_mode` setting for handling missing files. The default is the value of `sql_mode`. See Strict SQL Mode. The `FILES` parameter `STRICT_LOAD` can override this setting.

  The following example sets `STRICT_LOAD` to `0` in the table options.

```
mysql> CREATE EXTERNAL TABLE table_1(
        col_1 int,
        col_2 int,
        col_3 int)
        FILE_FORMAT = (FORMAT csv)
        FILES = (URI = 'oci://mybucket@mynamespace/data_files/data_file_1.csv')
        STRICT_LOAD = 0;
```

- `VERIFY_KEY_CONSTRAINTS`: Lakehouse validates primary key and unique key constraints during the initial load based on the selected option. The supported options are:

  - `1`: Lakehouse validates primary key and unique key constraints only during the initial load of the table. If there are subsequent loads or refreshes of the table, validation does not occur.
  - `0`: Lakehouse does not validate primary key and unique key constraints.
  - `DEFAULT`: Lakehouse does not validate primary key and unique key constraints.

  The following example sets `VERIFY_KEY_CONSTRAINTS` to `0` in the table options.

```
mysql> CREATE EXTERNAL TABLE table_1(
        col_1 int,
        col_2 int,
        col_3 int)
        FILE_FORMAT = (FORMAT csv)
        FILES = (URI = 'oci://mybucket@mynamespace/data_files/data_file_1.csv')
        VERIFY_KEY_CONSTRAINTS = 0;
```

Refer to the following table to review how different combinations of settings for `ALLOW_MISSING_FILES` and `STRICT_LOAD` affect errors for missing files.

Table 4.6 Combinations of Settings for ALLOW_MISSING_FILES and STRICT_LOAD

ALLOW_MISSING_FILES	STRICT_LOAD	Possible Errors
1	1	No error generated unless all files are missing.
1	0	
1	Not set	
Not set	0	
0	1	Error generated for any missing files.
0	0	
0	Not set	
Not set	1	
Not set	Not set	Error generation depends on setting for `sql_mode`.

File Specifications

File specifications are enclosed in `FILES`.

A `FILES` clause is required if you specify any additional external table options. The following parameters are available:

- `FILES`: Defines the Object Storage files. You can define the file locations using uniform resource identifier (URI) or PAR URL.

  As of MySQL 9.4.1, Lakehouse supports a maximum of 164 file locations. Before MySQL 9.4.1, Lakehouse supports a maximum of 170 file locations. To define more than the maximum number of files, store the files under the same bucket or use `FILE_PREFIX` or `FILE_PATTERN`.

- `FILES` parameters for uniform resource identifiers:

  - `URI`: The URL of the URI you specify. Do not provide a pattern, prefix, or name parameter with the `URI`. These details are inferred by the `object_path` you define.

    The following examples use the URI syntax for OCIFS:

    ```
    oci://bucket_name@namespace_name/object_path
    ```

    - Name: If the `object_path` is neither a glob pattern nor prefix.

      The following example specifies a single file with an OCIFS URI.

      ```
      mysql> CREATE EXTERNAL TABLE table_1(
              col_1 int,
              col_2 int,
              col_3 int)
              FILE_FORMAT = (FORMAT csv)
              FILES = (URI = 'oci://mybucket@mynamespace/data_files/data_file_1.csv');
      ```

    - Prefix: If the `object_path` is not a glob pattern and ends with an unencoded `/` character, such as a folder path.

      The following example uses a prefix with an OCIFS URI. The prefix specifies files in an Object Storage folder.

      ```
      mysql> CREATE EXTERNAL TABLE table_1(
              col_1 int,
              col_2 int,
      ```

```
        col_3 int)
        FILE_FORMAT = (FORMAT csv)
        FILES = (URI = 'oci://mybucket@mynamespace/data_files/');
```

- Glob pattern: If the `object_path` contains at least one of the following characters: `?`, `*`, or `[`. To use these characters as literals, you need to escape them or encode them as needed depending on the URI syntax. Regex patterns are not supported. See Glob Patterns from the *Oracle Cloud Infrastructure Documentation* to learn more.

  The following example uses an unencoded `*` character to create a glob pattern.

```
mysql> CREATE EXTERNAL TABLE table_1(
        col_1 int,
        col_2 int,
        col_3 int)
        FILE_FORMAT = (FORMAT csv)
        FILES = (URI = 'oci://mybucket@mynamespace/data_files/data_files/data_file_*.csv');
```

Review additional examples that use URIs to create external tables.

- `FILES` parameters for pre-authenticated requests:
  - `URL`: The PAR URL. Review PAR Recommendations.

    The following example specifies a single file with a PAR.

```
mysql> CREATE EXTERNAL TABLE table_1(
        col_1 int,
        col_2 int,
        col_3 int)
        FILE_FORMAT = (FORMAT csv)
        FILES = (URL = 'https://objectstorage.us-ashburn-1.oraclecloud.com/p/.../n/tenancy_1/b/buck
```

  - `FILE_NAME`: A specific Object Storage file name. For pre-authenticated requests, the file name provided in the PAR URL is considered.

    The following example uses the `FILE_NAME` parameter to specify a file in an Object Storage bucket. The PAR points to the Object Storage bucket.

```
mysql> CREATE EXTERNAL TABLE table_1(
        col_1 int,
        col_2 int,
        col_3 int)
        FILE_FORMAT = (FORMAT csv)
        FILES = (URL = 'https://objectstorage.us-ashburn-1.oraclecloud.com/p/.../n/tenancy_1/b/buck
                 FILE_NAME = 'data_files/data_file_1.csv');
```

  - `FILE_PATTERN`: A regular expression that defines a set of Object Storage files.

    The following example uses a pattern to load files that have a numerical suffix.

```
mysql> CREATE EXTERNAL TABLE table_1(
        col_1 int,
        col_2 int,
        col_3 int)
        FILE_FORMAT = (FORMAT csv)
        FILES = (URL = 'https://objectstorage.us-ashburn-1.oraclecloud.com/p/.../n/tenancy_1/b/buck
                 FILE_PATTERN = 'data_files/data_file_\\d+\\.csv');
```

    See the following to learn more:

    - Regular Expression Syntax
    - Modified ECMAScript regular expression grammar

  - `FILE_PREFIX`: The prefix for a set of Object Storage files. The prefix or bucket name present in the PAR URL is considered.

The following example uses a prefix to specify files to load in an Object Storage folder.

```
mysql> CREATE EXTERNAL TABLE table_1(
        col_1 int,
        col_2 int,
        col_3 int)
        FILE_FORMAT = (FORMAT csv)
        FILES = (URL = 'https://objectstorage.us-ashburn-1.oraclecloud.com/p/.../n/tenancy_1/b/bucket_1
                 FILE_PREFIX = 'data_files/');
```

Review additional examples that use PARs to create external tables.

The following parameters enclosed in `FILES` are optional:

- `ALLOW_MISSING_FILES`: Handles missing files based on the selected option. This overrides the table option parameter `STRICT_LOAD` for missing files. To learn about the default value, see the `ALLOW_MISSING_FILES` table option parameter in Table Parameters. The supported options are:
  - `1`: If any file is missing, no error occurs and data loading continues with the existing files. An error occurs if all the files are not available.
  - `0`: If any file is missing, an error occurs and data is not loaded.

  The following example uses a glob pattern specified with a uniform resource identifier (URI) and sets `ALLOW_MISSING_FILES` to `1` in the `FILES` parameters. If any of the specified files in the pattern are not available, an error does not occur and data continues loading for the existing files. If all files are missing, an error occurs.

  ```
  mysql> CREATE EXTERNAL TABLE table_1(
          col_1 int,
          col_2 int,
          col_3 int)
          FILE_FORMAT = (FORMAT csv)
          FILES = (URI = 'oci://mybucket@mynamespace/data_files/data_file_[1-10].csv'
                   ALLOW_MISSING_FILES = 1);
  ```

  The following example sets `ALLOW_MISSING_FILES` to `0` for the first file and `ALLOW_MISSING_FILES` to `1` for the second file. If the first file is missing, an error occurs. If the second file is missing, the load continues with the loading of the first file.

  ```
  mysql> CREATE EXTERNAL TABLE table_1(
          col_1 int,
          col_2 int,
          col_3 int)
          FILE_FORMAT = (FORMAT csv)
          FILES = (URI = 'oci://mybucket@mynamespace/data_files/data_file_1.csv' ALLOW_MISSING_FILES = 0,
                   URI = 'oci://mybucket@mynamespace/data_files/data_file_2.csv' ALLOW_MISSING_FILES = 1);
  ```

- `STRICT_LOAD`: Loads data in strict mode or non-strict mode, based on the selected option. This overrides the `STRICT_LOAD` table option parameter. To learn about the default value, see the `STRICT_LOAD` table option parameter in Table Parameters. The supported options are:
  - `1`: Loads the data in strict mode. The data loading stops if there is an error due to missing files, empty columns, formatting errors or parsing errors.
  - `0`: Loads the data in non-strict mode. Missing files, empty columns, formatting errors, or parsing errors display a warning, and data is loaded. If there are missing columns, implicit default values are used in the absence of user-provided default values. See Data Type Default Values to learn more.

  The following example sets `STRICT_LOAD` to `0` in the `FILES` parameters.

  ```
  mysql> CREATE EXTERNAL TABLE table_1(
          col_1 int,
          col_2 int,
  ```

```
       col_3 int)
       FILE_FORMAT = (FORMAT csv)
       FILES = (URI = 'oci://mybucket@mynamespace/data_files/data_file_1.csv'
                STRICT_LOAD = 0);
```

File Format Options

File format options are enclosed in `FILE_FORMAT`.

The following file format options are available:

- `FORMAT`: The file format defined in a table. You can define only one file format per table. The supported file formats are:
  - `csv`: The default file format.

    The following example specifies a CSV file.

    ```
    mysql> CREATE EXTERNAL TABLE table_1(
           col_1 int,
           col_2 int,
           col_3 int)
           FILE_FORMAT = (FORMAT csv)
           FILES = (URI = 'oci://mybucket@mynamespace/data_files/data_file_1.csv');
    ```

  - `json`: Only Newline Delimited JSON files are supported. Tables created with `json` format must only have a single column that conforms to the JSON data type.

    The following example specifies a JSON file.

    ```
    mysql> CREATE EXTERNAL TABLE table_1(
           col_1 json)
           FILE_FORMAT = (FORMAT json)
           FILES = (URI = 'oci://mybucket@mynamespace/data_files/data_file_1.json');
    ```

  - `parquet`: The Parquet data type.

    The following example specifies a Parquet file.

    ```
    mysql> CREATE EXTERNAL TABLE table_1(
           col_1 int,
           col_2 int,
           col_3 int)
           FILE_FORMAT = (FORMAT parquet)
           FILES = (URI = 'oci://mybucket@mynamespace/data_files/data_file_1.parquet');
    ```

  - `avro`: The Avro data type.

    The following example specifies an Avro file.

    ```
    mysql> CREATE EXTERNAL TABLE table_1(
           col_1 int,
           col_2 int,
           col_3 int)
           FILE_FORMAT = (FORMAT avro)
           FILES = (URI = 'oci://mybucket@mynamespace/data_files/data_file_1.avro');
    ```

  - `delta`: Supported as of MySQL 9.5.0. Loads delta lake tables. See Delta Lake Tables and Limitations for Delta Lake Tables to learn more.

    The following example specifies a delta lake table.

    ```
    mysql> CREATE EXTERNAL TABLE table_1(
           col_1 int,
           col_2 int,
           col_3 int)
           FILE_FORMAT = (FORMAT delta)
           FILES = (URI = 'oci://mybucket@mynamespace/data_files/');
    ```

- `COMPRESSION`: Use this to load compressed files.

  Compression is supported for all file formats:

  - `csv`: The default value is no compression. You can set `compression` to `zip`, `gzip`, or `bzip2`.

    The following example specifies a CSV file compressed in `zip` format.

    ```
    mysql> CREATE EXTERNAL TABLE table_1(
           col_1 int,
           col_2 int,
           col_3 int)
           FILE_FORMAT = (FORMAT csv COMPRESSION zip)
           FILES = (URI = 'oci://mybucket@mynamespace/data_files/data_file_1.csv.zip');
    ```

  - `json`: The default value is no compression. You can set `compression` to `zip`, `gzip`, or `bzip2`.

    The following example specifies a JSON file compressed in `gzip` format.

    ```
    mysql> CREATE EXTERNAL TABLE table_1(
           col_1 json)
           FILE_FORMAT = (FORMAT json COMPRESSION gzip)
           FILES = (URI = 'oci://mybucket@mynamespace/data_files/data_file_1.json.gz');
    ```

  - `avro`: The default value is `auto`. You cannot set any other options for `COMPRESSION`, as Avro files declare their compression format in the file metadata.

    The following specifies an Avro file with the `auto` compression option.

    ```
    mysql> CREATE EXTERNAL TABLE table_1(
           col_1 int,
           col_2 int,
           col_3 int)
           FILE_FORMAT = (FORMAT avro COMPRESSION auto)
           FILES = (URI = 'oci://mybucket@mynamespace/data_files/data_file_1.avro');
    ```

  - `parquet`: The default value is `auto`. You cannot set any other options for `COMPRESSION`, as Parquet files declare their compression format in the file metadata.

    The following example specifies a Parquet file with the `auto` compression option.

    ```
    mysql> CREATE EXTERNAL TABLE table_1(
           col_1 int,
           col_2 int,
           col_3 int)
           FILE_FORMAT = (FORMAT parquet COMPRESSION auto)
           FILES = (URI = 'oci://mybucket@mynamespace/data_files/data_file_1.parquet');
    ```

  Note the following when loading compressed files:

  - Each compressed file must only contain a single original file. You cannot provide a compressed file with archives or folders with multiple files.
  - Files in `zip` format must be compressed with the Deflate algorithm.
  - If loading multiple files, all the files must be in the same compression format.
  - When loading multiple compressed files, it is best to prepare files of similar size, and in the range of 80-120 MB each. Otherwise, you may experience longer loading times.

- `CHARACTER SET`: Defines the character encoding. The default is `'utf8mb4'`. Only available for CSV and JSON files.

- `HEADER`: Adds a header row to the `CSV` file based on the selected option (`ON` or `OFF`). The default is `OFF`. Only available for CSV files.

  If `HEADER` and `IGNORE` are both specified, Lakehouse first skips the number of rows, and then uses the next row as the header row.

The following field options are available:

- The `FIELDS` and `COLUMNS` clauses define the format for fields and columns. The following parameters are available:

  - `DATE FORMAT`: The date format. See `date_format`. You can also set date formats for each column by using the `EXTERNAL_FORMAT` statement, which overrides the format in the table parameter. Only available for CSV files.

  - `TIME FORMAT`: The time format. See String and Numeric Literals in Date and Time Context. You can also set time formats for each column by using the `EXTERNAL_FORMAT` statement, which overrides the format in the table parameter. Only available for CSV files.

  - `DATETIME FORMAT`: The datetime format. See `date_format`. You can also set datetime formats for each column by using the `EXTERNAL_FORMAT` statement, which overrides the format in the table parameter. Only available for CSV files.

  - As of MySQL 9.5.0, Lakehouse Auto Parallel Load and Guided Load can infer supported formats for temporal data types by setting the appropriate field option to `auto`. The `auto` setting is the default value if no temporal format is specified. See About Lakehouse Auto Parallel Load Schema Inference to learn more.

  - The following example sets the `DATE FORMAT`, `TIME FORMAT`, and `DATETIME FORMAT`.

    ```
    mysql> CREATE EXTERNAL TABLE table_1(
           col_1 date,
           col_2 time,
           col_3 timestamp)
           FILE_FORMAT = (FORMAT csv FIELDS DATE FORMAT '%W %M %Y' TIME FORMAT '%H:%i:%s' DATETIME FOR
           FILES = (URI = 'oci://mybucket@mynamespace/data_files/data_file_1.csv');
    ```

    The following example sets the `DATE FORMAT`, `TIME FORMAT`, and `DATETIME FORMAT` to `auto`, which allows Auto Parallel Load and Guided Load to infer supported formats for temporal data types.

    ```
    mysql> CREATE EXTERNAL TABLE table_1(
           col_1 date,
           col_2 time,
           col_3 timestamp)
           FILE_FORMAT = (FORMAT csv FIELDS DATE FORMAT 'auto' TIME FORMAT 'auto' DATETIME FORMAT 'aut
           FILES = (URI = 'oci://mybucket@mynamespace/data_files/data_file_1.csv');
    ```

- `TERMINATED BY`: Specifies one or more characters used to enclose fields. The maximum field delimiter length is 64 characters. You can add any string as a *`custom_field_delimiter`*. Only available for CSV files.

  When `FIELDS TERMINATED BY` is set to `auto`, Lakehouse automatically detects the field delimiters.

  The following field delimiters are detected automatically by Lakehouse:

  - `|`: Pipe.
  - `,`: Comma
  - `\t`: Tab

- `;`: Semicolon

The following example sets the `FIELDS TERMINATED BY` and `LINES TERMINATED BY` options.

```
mysql> CREATE EXTERNAL TABLE table_1(
        col_1 int,
        col_2 int,
        col_3 int)
        FILE_FORMAT = (FORMAT csv FIELDS TERMINATED BY '\r\n' LINES TERMINATED BY ';')
        FILES = (URI = 'oci://mybucket@mynamespace/data_files/data_file_1.csv');
```

- `[OPTIONALLY] ENCLOSED BY`: Specifies one or more characters used to enclose fields. The default is `'\''`. You can add any string as a *`custom_quotation_marks`*. As of MySQL 9.5.0, you have the option to ignore the quote character by setting its value to an empty string. Only available for CSV files.

- `ESCAPED BY`: Specifies one or more characters used to escape special characters. The default is `'\'`. You might have to use an additional escape character depending on the client you use. You can add any string as a *`custom_escape_character`*. As of MySQL 9.5.0, you have the option to ignore the escape character by setting its value to an empty string. Only available for CSV files.

  Consider the following example for setting empty strings for escape and quotation characters. You have the following data in a CSV file: `"{""doc1"":{\""doc2\"":22}}"`.

  - You create an external table and set an empty string for the escape character: `ESCAPED BY ''`.

    ```
    mysql> CREATE EXTERNAL TABLE table_1(
            col_1 text)
            FILE_FORMAT = (FORMAT csv FIELDS ESCAPED BY '')
            FILES = (URI = 'oci://mybucket@mynamespace/data_files/data_file_1.csv');
    ```

    The value loads as: `{"doc1":{\"doc2\":22}}`.

  - You create an external table and set an empty string for the quotation character: `ENCLOSED BY ''`.

    ```
    mysql> CREATE EXTERNAL TABLE table_1(
            col_1 text)
            FILE_FORMAT = (FORMAT csv FIELDS ENCLOSED BY '')
            FILES = (URI = 'oci://mybucket@mynamespace/data_files/data_file_1.csv');
    ```

    The value loads as: `"{""doc1"":{""doc2"":22}}"`.

  - You create an external table and set empty strings for both the escape character and quotation character.

    ```
    mysql> CREATE EXTERNAL TABLE table_1(
            col_1 text)
            FILE_FORMAT = (FORMAT csv FIELDS ESCAPED BY '' OPTIONALLY ENCLOSED BY '')
            FILES = (URI = 'oci://mybucket@mynamespace/data_files/data_file_1.csv');
    ```

    The value loads as: `"{""doc1"":{\""doc2\"":22}}"`.

  The following example sets the `ESCAPED BY` and `[OPTIONALLY] ENCLOSED BY` options.

```
mysql> CREATE EXTERNAL TABLE table_1(
        col_1 int,
        col_2 int,
        col_3 int)
        FILE_FORMAT = (FORMAT csv FIELDS ESCAPED BY '\\' OPTIONALLY ENCLOSED BY '\"')
        FILES = (URI = 'oci://mybucket@mynamespace/data_files/data_file_1.csv');
```

Additional File Format Options

The following additional file format options are available:

- The `LINES` clause defines the format for each new line. The following parameters are available:
  - `TERMINATED BY`: Specifies one or more characters used to delimit records. The maximum record delimiter length is 64 characters. You can add any string as a delimiter. Only available for CSV and JSON files.

    The default record delimiter for is `'\n'`.

    For `CSV` files, if you set `LINES TERMINATED BY` to `auto`, Lakehouse automatically detects the record delimiters and composite record delimiters with field delimiters as prefixes.

    The following record delimiters detected automatically by Lakehouse:

    - `\r`: Carriage return.
    - `\n`: Line feed. This is the default.
    - `\r\n` : Carriage return and line feed
- `IGNORE`: The number of rows to skip at the start of the file. The default value is `0` and the maximum value is `20`. Only available for CSV files.

  If `HEADER` and `IGNORE` are both specified, Lakehouse first skips the number of rows, and then uses the next row as the header row.

  The following example sets `IGNORE` and `HEADER`. Since both parameters are set, the first row is skipped and the second row is set as the header row.

```
mysql> CREATE EXTERNAL TABLE table_1(
        col_1 int,
        col_2 int,
        col_3 int)
        FILE_FORMAT = (FORMAT csv HEADER ON IGNORE 1 LINES)
        FILES = (URI = 'oci://mybucket@mynamespace/data_files/data_file_1.csv');
```

Additional Syntax Examples

The following examples load tables with additional options and show the proper order of options in the commands.

```
mysql> CREATE EXTERNAL TABLE table_1(
        col_1 int,
        col_2 int,
        col_3 DATE EXTERNAL_FORMAT '%e/%c %Y')
        FILE_FORMAT = (FORMAT csv COMPRESSION zip CHARACTER SET utf8mb4 HEADER OFF
                       FIELDS DATE FORMAT '%Y%M%D' TERMINATED BY ',' LINES TERMINATED BY '\n' IGNORE 2
        FILES = (URI = 'oci://mybucket@mynamespace/data_files/data_file_1.csv.zip' ALLOW_MISSING_FILES
```

```
mysql> CREATE EXTERNAL TABLE table_1(
        col_1 int,
        col_2 int,
        col_3 int)
        FILE_FORMAT = (FORMAT csv HEADER ON FIELDS ESCAPED BY '\\' IGNORE 1 LINES)
        FILES = (URI = 'oci://mybucket@mynamespace/data_files/data_file_[1-5].csv' STRICT_LOAD = 1)
        ALLOW_MISSING_FILES = 1;
```

```
mysql> CREATE EXTERNAL TABLE table_1(
        col_1 int,
        col_2 int,
        col_3 int)
        FILE_FORMAT = (FORMAT csv HEADER ON FIELDS TERMINATED BY ','
                       OPTIONALLY ENCLOSED BY '"' IGNORE 1 LINES)
```

```
        FILES = (URI = 'oci://mybucket@mynamespace/data_files/data_file_1.csv')
        VERIFY_KEY_CONSTRAINTS = 0;
```

What's Next

- Learn about Lakehouse Auto Parallel Load
- Learn how to create external tables using Auto Parallel Load or manually.
- Learn how to Refresh Data Using Event-Based Incremental Load.

4.3.6.2 Lakehouse External Table JSON Syntax

External tables use table-level and column-level `ENGINE_ATTRIBUTE` options of the `CREATE TABLE` statement to specify the parameters needed to process data stored externally in Object Storage. The column-level `ENGINE_ATTRIBUTE` overrides specific `dialect` parameters, if required.

This topic contains the following sections:

- CREATE TABLE Statement
- External Table ENGINE_ATTRIBUTE Options
- Column Definitions
- File Options
- Event-Based Incremental Load Options
- Column Matching Options
- Allow Missing Columns Option
- Dialect Parameters
- Common Dialect Parameters
- Dialect Parameters for CSV and JSON Files
- Dialect Parameters for CSV and Parquet Files
- Dialect Parameters for CSV Files
- What's Next

CREATE TABLE Statement

As of MySQL 9.4.0, you can use the `CREATE EXTERNAL TABLE` statement to specify `ENGINE_ATTRIBUTE` options.

```
CREATE EXTERNAL TABLE table_name
(col_name data_type [ENGINE_ATTRIBUTE='Col_Engine_Attribute_JSON'], ...)
ENGINE_ATTRIBUTE='Tbl_Engine_Attribute_JSON'
[PARTITION BY KEY (partition_column_name)];
```

The `PARTITION BY KEY` clause is optional. To learn more, see Create Tables with Key Partitioning.

If you are on a version earlier than MySQL 9.4.0, you must use the `CREATE TABLE` statement to specify `ENGINE_ATTRIBUTE` options. You must also manually set `ENGINE` to `lakehouse`, and `SECONDARY_ENGINE` to `rapid`.

```
CREATE TABLE table_name
(col_name data_type [ENGINE_ATTRIBUTE='Col_Engine_Attribute_JSON'], ...)
ENGINE=lakehouse
SECONDARY_ENGINE=rapid
```

```
ENGINE_ATTRIBUTE='Tbl_Engine_Attribute_JSON';
```

After creating a table, you can load data into the table in the following ways:

- Load the data automatically with Lakehouse Auto Parallel Load by using the `HEATWAVE_LOAD` command. See how to Load External Data Using Lakehouse Auto Parallel Load with an Existing Table.
- Load the data manually with the `ALTER TABLE` statement and the option to use Guided Load.

You can also use `ENGINE_ATTRIBUTE` options with the `HEATWAVE_LOAD` command to load data using Lakehouse Auto Parallel Load. See HEATWAVE_LOAD and Load Structured Data Using Lakehouse Auto Parallel Load.

External Table ENGINE_ATTRIBUTE Options

The external table `ENGINE_ATTRIBUTE` is a `JSON` object that consists of the dialect parameters and the file parameters. They are represented in the following ways:

- *`file_JSON_array`* is a `JSON` array containing the `JSON` objects representing the file location details and file parameters. You can use the following methods to specify these details:
  - Uniform Resource Identifiers (URI)

    Available as of MySQL 9.3.1 for MySQL HeatWave on OCI. You can provide an OCIFS URI, PAR URI, or Native URI.
  - Resource Principals (RP)

    See Resource Principals in *Oracle Cloud Infrastructure Documentation*.
  - Pre-Authenticated Requests (PAR)

    See Object Storage Pre-Authenticated Requests in *Oracle Cloud Infrastructure Documentation*.

  Refer to File Options.
- *`dialect_JSON_object`* is a `JSON` object containing the dialect parameters to specify the data format in the files.

```
Tbl_Engine_Attribute_JSON: {
  begin-object
    "file": file_JSON_array
    [, "dialect": dialect_JSON_object ]
    [, "auto_refresh_event_source": {"stream_OCID"}
    [, "match_columns_by": {"order" | "name_case_sensitive" | "name_case_insensitive"}]
    [, "allow_missing_columns": {true|false}]
  end-object
}

file_JSON_array: {
  begin-array file_def [, file_def] … end-array
}

file_def: file_RP_JSON | file_PAR_JSON | file_URI_JSON

file_def_RP_JSON: {
  "bucket": "bucket_name",
  "namespace": "namespace_name",
  "region": "region_name",
  {"prefix": "prefix_value" | "name": "filename" | "pattern" : "pattern_value"},
  "is_strict_mode": {true | false},
  "allow_missing_files": {true | false}
}

file_PAR_JSON: {
```

```
  "par": "PAR URL",
  ["prefix": "prefix_value", | "name": "filename", | "pattern" : "pattern_value",]
  "is_strict_mode": {true | false},
  "allow_missing_files": {true | false}
}

file_URI_JSON: {
  ("uri": "OCIFS_URI") | ("uri": "PAR_URI") | ("uri": "NATIVE_URI"),
  "is_strict_mode": {true | false},
  "allow_missing_files": {true | false}
}

dialect_JSON_object: begin-object dialect_option_key_value [, dialect_option_key_value]... end-object

dialect_option_key_value: {
    "format": {"avro" | "csv" | "json" | "parquet" | "delta"}
  | "check_constraints": {true | false}
  | "field_delimiter": {"|" | "," | "\\t" | ";" | "auto" | "custom_field_delimiter"}
  | "record_delimiter": {"\\r" | "\\n" | "\\r\\n" | "auto" | "custom_record_delimiter"}
  | "escape_character": {"\\\\" | "" | "custom_escape_character"}
  | "quotation_marks": {"\\"" | "" | "custom_quotation_marks"}
  | "encoding": "utf8mb4"
  | "date_format": "custom_date_format"
  | "time_format": "custom_time_format"
  | "timestamp_format": "custom_timestamp_or_datetime_format"
  | "datetime_format": "custom_timestamp_or_datetime_format"
  | "trim_spaces": {true | false}
  | "skip_rows": Integer_20
  | "has_header": {true | false}
  | "is_strict_mode": {true | false}
  | "compression": {"zip" | "gzip" | "bzip2" | "auto"}
  | "embed_model_id": {"MySQL HeatWave_in_database_embedding_model"}
}
Integer_20: Represents integer value between 0 and 20.
```

Note

- When you specify `ENGINE_ATTRIBUTE` parameters in `JSON` format, you must escape certain characters using `\ (backslash)`. For example, to specify linefeed as a record delimiter, you need to specify it as `"\\n"` in the `ENGINE_ATTRIBUTE` parameter's `JSON` value.
- In the syntax description, square brackets (`"["` and `"]"`) indicate optional words or clauses.
- The syntax description uses a vertical bar (`"|"`) to separate alternatives. The list of alternatives are enclosed in square brackets `"["` and `"]"`) when you must choose one member. The list of alternatives are enclosed in curly braces `"{"` and `"}"`) when you must choose one member.
- The terms *`begin-array`* and *`end-array`* in the syntax represent a `JSON` array. While writing the sample, replace them with `[]`.
- The terms *`begin-object`* and *`end-object`* in the syntax represent a `JSON` object. While writing the sample, replace them with `{}`.
- The following examples create tables with the `CREATE EXTERNAL TABLE` statement. If you are on a version earlier than MySQL 9.4.0, you must use `CREATE TABLE` and set `ENGINE` to `lakehouse`, and `SECONDARY_ENGINE` to `rapid`. See CREATE TABLE Statement.

Column Definitions

For versions earlier than MySQL 9.5.0, you can override some options at the column level by specifying column-level parameters. For MySQL 9.5.0 and later, you can do this by loading data manually with Guided Load disabled. You can set the `date_format`, `time_format`,

`timestamp_format`, `datetime_format` (as of MySQL 9.5.1), and `model` as optional column-level `ENGINE_ATTRIBUTE` parameters. If you set both `timestamp_format` and `datetime_format`, only the `datetime_format` is used for inference and loading of timestamp/datetime values. We recommend using `datetime_format` instead of `timestamp_format`. For more information, refer to dialect parameters.

As of MySQL 9.5.0, Lakehouse Auto Parallel Load and Guided Load can infer supported formats for temporal data types by setting the appropriate dialect-level temporal formats to `auto`. The `auto` setting is the default value if no temporal format is specified. If you set a specific format at the column level for a temporal data type when creating a table, and a different format is inferred when loading the table to Lakehouse, the specific format used for the column when creating the table is prioritized and used for loading the column. See About Lakehouse Auto Parallel Load Schema Inference to learn more.

Note

The temporal column-level parameters of `date_format`, `time_format`, `timestamp_format`, and `datetime_format` are only available for CSV files. See Dialect Parameters for CSV Files.

```
Col_Engine_Attribute_JSON: begin-object column_option_key_value [, column_option_key_value]... end-obje

column_option_key_value: {
    "date_format": "custom_date_format"
  | "time_format": "custom_time_format"
  | "timestamp_format": "custom_timestamp_or_datetime_format"
  | "datetime_format": "custom_timestamp_or_datetime_format"
  | "model": {MySQL HeatWave_in_database_embedding_model}
}
```

- `model`: Supported as of MySQL 9.3.2. Lets you specify the column-level embedding model for `VECTOR` columns. You must specify a model supported by MySQL HeatWave GenAI. If the vector uses another embedding model, you cannot use it with GenAI features. You can include an unsupported embedding model in the column or table comments for your own reference. Learn more about the Vector Data Type.

  The following example sets the embedding model for the column to `multilingual-e5-small`.

  ```
  mysql> CREATE EXTERNAL TABLE table_1(col_1 VECTOR(5) ENGINE_ATTRIBUTE '{"model": "multilingual-e5-sma
         ENGINE_ATTRIBUTE='{"dialect": {"format": "csv"},
                           "file": [{"uri": "oci://mybucket@mynamespace/data_files/data_file_1.csv"}]
  ```

The following example sets the `date_format`, `time_format`, and `datetime_format` at the column level.

```
mysql> CREATE EXTERNAL TABLE table_1(
        col_1 date ENGINE_ATTRIBUTE '{"date_format": "%d/%m/%Y"}',
        col_2 time ENGINE_ATTRIBUTE '{"time_format": "%I:%i %p"}',
        col_3 timestamp ENGINE_ATTRIBUTE '{"datetime_format": "%Y-%m-%d %H:%i:%s"}',
        col_4 int)
        ENGINE_ATTRIBUTE='{"dialect": {"format": "csv"},
                          "file": [{"uri": "oci://mybucket@mynamespace/data_files/data_file_1.csv"}]}'
```

The following example sets the `date_format`, `time_format`, and `datetime_format` options to `auto`, which allows Lakehouse Auto Parallel Load and Guided Load to infer supported formats for temporal data types.

```
mysql> CREATE EXTERNAL TABLE table_1(
        col_1 date ENGINE_ATTRIBUTE '{"date_format": "auto"}',
        col_2 time ENGINE_ATTRIBUTE '{"time_format": "auto"}',
        col_3 datetime ENGINE_ATTRIBUTE '{"datetime_format": "auto"}',
        col_4 int)
        ENGINE_ATTRIBUTE='{"dialect": {"format": "csv"},
                          "file": [{"uri": "oci://mybucket@mynamespace/data_files/data_file_1.csv"}]}'
```

For more information about other column definitions, see Dialect Parameters for CSV Files.

File Options

The following parameters are required to create an external table:

- `ENGINE`: Set to `lakehouse` if you are on a version earlier than MySQL 9.4.0. For MySQL 9.4.0 and later, you can use the `CREATE EXTERNAL TABLE` statement, which automatically sets `ENGINE` to `lakehouse`.
- `SECONDARY_ENGINE`: Set to `rapid` if you are on a version earlier than MySQL 9.4.0. For MySQL 9.4.0 and later, you can use the `CREATE EXTERNAL TABLE` statement, which automatically sets `SECONDARY_ENGINE` to `rapid`.
- `ENGINE_ATTRIBUTE`: JSON object literal. Specifies the location of files, the file format, and how the file format is handled.

  Use key-value pairs in `JSON` format to specify *`options`*. Lakehouse uses the default setting if there is no defined option. Use `NULL` to specify no arguments.
- `file`: Defines the Object Storage files. You can define the file locations using resource principals, uniform resource identifier (URI) (as of MySQL 9.3.1 for MySQL HeatWave on OCI), or PAR URL.

  As of MySQL 9.4.1, Lakehouse supports a maximum of 164 file locations. As of MySQL 9.3.1, Lakehouse supports a maximum of 170 file locations. Before MySQL 9.3.1, Lakehouse supports a maximum of 256 file locations. To define more than the maximum number of files, store the files under the same bucket or use `prefix` or `pattern`.

  - `file` parameters for uniform resource identifiers (As of MySQL 9.3.1 for MySQL HeatWave on OCI):

    - `uri`: The URL depending on the type of URI you define. Do not provide a pattern, prefix, or name parameter with the `uri`. These details are inferred by the `object_path` you define.

      The following examples use the URI syntax for OCIFS:

      ```
      oci://bucket_name@namespace_name/object_path
      ```

      - Name: If the `object_path` is neither a glob pattern nor prefix.

        The following example specifies a single file by using an OCIFS URI.

        ```
        mysql> CREATE EXTERNAL TABLE table_1(
                col_1 int,
                col_2 int,
                col_3 int)
                ENGINE_ATTRIBUTE='{"dialect": {"format": "csv"},
                                   "file": [{"uri": "oci://mybucket@mynamespace/data_files/data_file_1.csv
        ```

      - Prefix: If the `object_path` is not a glob pattern and ends with an unencoded `/` character, such as a folder path.

        The following example uses a prefix with an OCIFS URI. The prefix specifies files in an Object Storage folder.

        ```
        mysql> CREATE EXTERNAL TABLE table_1(
                col_1 int,
                col_2 int,
                col_3 int)
                ENGINE_ATTRIBUTE='{"dialect": {"format": "csv"},
                                   "file": [{"uri": "oci://mybucket@mynamespace/data_files/"}]}';
        ```

      - Glob pattern: If the `object_path` contains at least one of the following unencoded characters: `?`, `*`, or `[`. To use these characters as literals, you need to escape them or encode them as needed depending on the URI syntax. Regex patterns are not supported. See Glob Patterns from the *Oracle Cloud Infrastructure Documentation* to learn more.

The following example uses an unencoded * character to create a glob pattern.

```
mysql> CREATE EXTERNAL TABLE table_1(
        col_1 int,
        col_2 int,
        col_3 int)
        ENGINE_ATTRIBUTE='{"dialect": {"format": "csv"},
                           "file": [{"uri": "oci://mybucket@mynamespace/data_files/data_files/d
```

Review additional examples that use URIs to create external tables using Auto Parallel Load or manually.

- `file` parameters for resource principals:
  - `bucket_name`: Buckets allow you to storage objects in a compartment. To learn more about buckets, see Object Storage Buckets in *Oracle Cloud Infrastructure Documentation*.
  - `namespace_name`: This is the top-level container for all buckets and objects. To learn how to view your namespace, see Object Storage Namespaces in *Oracle Cloud Infrastructure Documentation*.
  - `region_name`: You need to define the region identifier for your OCI tenancy. To view the list of available regions and region identifiers, see Regions and Availability Domains in *Oracle Cloud Infrastructure Documentation*.

    As of MySQL 9.2.1, if the region is not specified, the instance's associated region is used as the default region. Otherwise, the specified region is used.

  The following example specifies a single file by using resource principals.

```
mysql> CREATE EXTERNAL TABLE table_1(
        col_1 int,
        col_2 int,
        col_3 int)
        ENGINE_ATTRIBUTE='{"dialect": {"format": "csv"},
                           "file": [{"region": "us-ashburn-1",
                                     "namespace": "tenant_1",
                                     "bucket": "bucket_1",
                                     "name": "data_file_1.csv"}]}';
```

Review additional examples that use resource principals to create external tables using Lakehouse Auto Parallel Load or manually.

- `file` parameters for pre-authenticated requests:
  - `par`: The PAR URL.

    Do not specify a `region`, `namespace` or `bucket` with `par`. These values are included in the PAR URL and generate an error if defined as separate parameters. For more information, refer to PAR Recommendations.

  The following example specifies a single file with a PAR.

```
mysql> CREATE EXTERNAL TABLE table_1(
        col_1 int,
        col_2 int,
        col_3 int)
        ENGINE_ATTRIBUTE='{"dialect": {"format": "csv"},
                           "file": [{"par": "https://objectstorage.us-ashburn-1.oraclecloud.com/p/.
```

Review additional examples that use PARs to create external tables using Lakehouse Auto Parallel Load or manually.

- The following `file` parameters apply only to PAR and resource principals. Use one of the parameters, unless the target defines a specific file.

  - `name`: A specific Object Storage file name. For pre-authenticated requests, the file name provided in the PAR URL is considered.

    The following example uses a PAR and the `name` parameter to specify a file in an Object Storage bucket. The PAR points to the Object Storage bucket.

    ```
    mysql> CREATE EXTERNAL TABLE table_1(
            col_1 int,
            col_2 int,
            col_3 int)
            ENGINE_ATTRIBUTE='{"dialect": {"format": "csv"},
                              "file": [{"par": "https://objectstorage.us-ashburn-1.oraclecloud.com/p/.../n
                                        "name": "data_files/data_file_1.csv"}]}';
    ```

  - `pattern`: A regular expression that defines a set of Object Storage files.

    The following example uses resource principals and a pattern to load files that have a numerical suffix.

    ```
    mysql> CREATE EXTERNAL TABLE table_1(
            col_1 int,
            col_2 int,
            col_3 int)
            ENGINE_ATTRIBUTE='{"dialect": {"format": "csv"},
                              "file": [{"region": "us-ashburn-1",
                                        "namespace": "tenant_1",
                                        "bucket": "bucket_1",
                                        "pattern": "data_files/data_file_\\\\d+\\\\.csv"}]}';
    ```

    See the following to learn more:

    - Regular Expression Syntax
    - Modified ECMAScript regular expression grammar

  - `prefix`: The prefix for a set of Object Storage files. For pre-authenticated requests, prefix or bucket name present in the PAR URL is considered.

    The following example uses resource principals and a prefix to specify files to load in an Object Storage folder.

    ```
    mysql> CREATE EXTERNAL TABLE table_1(
            col_1 int,
            col_2 int,
            col_3 int)
            ENGINE_ATTRIBUTE='{"dialect": {"format": "csv"},
                              "file": [{"region": "us-ashburn-1",
                                        "namespace": "tenant_1",
                                        "bucket": "bucket_1",
                                        "prefix": "data_files/"}]}';
    ```

  - Review examples that use PARs to create external tables using Lakehouse Auto Parallel Load or manually.

  - Review examples that use resource principals to create external tables using Lakehouse Auto Parallel Load or manually.

The following file parameters are optional:

- `compression`: Supported as of MySQL 9.3.2. Use this to load compressed files.

  Compression is supported for all file formats:

- `CSV`: The default value is no compression. You can set `compression` to `zip`, `gzip`, or `bzip2`.

  The following example specifies a CSV file compressed in `zip` format.

```
mysql> CREATE EXTERNAL TABLE table_1(
        col_1 int,
        col_2 int,
        col_3 int)
        ENGINE_ATTRIBUTE='{"dialect": {"format": "csv", "compression": "zip"},
                           "file": [{"uri": "oci://mybucket@mynamespace/data_files/data_file_1.csv.
```

- `JSON`: The default value is no compression. You can set `compression` to `zip`, `gzip`, or `bzip2`.

  The following example specifies a JSON file compressed in `gzip` format.

```
mysql> CREATE EXTERNAL TABLE table_1(
        col_1 int,
        col_2 int,
        col_3 int)
        ENGINE_ATTRIBUTE='{"dialect": {"format": "json", "compression": "gzip"},
                           "file": [{"uri": "oci://mybucket@mynamespace/data_files/data_file_1.jso
```

- `Avro`: The default value is `auto`. You cannot set any other options for `compression`, as Avro files declare their compression format in the file metadata.

  The following example specifies an Avro file with the `auto` compression option.

```
mysql> CREATE EXTERNAL TABLE table_1(
        col_1 int,
        col_2 int,
        col_3 int)
        ENGINE_ATTRIBUTE='{"dialect": {"format": "avro", "compression": "auto"},
                           "file": [{"uri": "oci://mybucket@mynamespace/data_files/data_file_1.avro
```

- `Parquet`: The default value is `auto`. You cannot set any other options for `compression`, as Parquet files declare their compression format in the file metadata.

  The following example specifies a Parquet file with the `auto` compression option.

```
mysql> CREATE EXTERNAL TABLE table_1(
        col_1 int,
        col_2 int,
        col_3 int)
        ENGINE_ATTRIBUTE='{"dialect": {"format": "parquet", "compression": "auto"},
                           "file": [{"uri": "oci://mybucket@mynamespace/data_files/data_file_1.parq
```

Note the following when loading compressed files:

- Each compressed file must only contain a single original file. You cannot provide a compressed file with archives or folders with multiple files.
- Files in `zip` format must be compressed with the Deflate algorithm.
- If loading multiple files, all the files must be in the same compression format.
- When loading multiple compressed files, it is best to prepare files of similar size, and in the range of 80-120 MB each. Otherwise, you may experience longer loading times.

- `is_strict_mode`: Loads data in strict mode or non-strict mode, based on the selected option. This overrides the `is_strict_mode` dialect parameter. To learn about the default value, see the `is_strict_mode` dialect parameter in Common Dialect Parameters. The supported options are:

  - `true`: Loads the data in strict mode. The data loading stops if there is an error due to missing files, empty columns, formatting errors or parsing errors.

  - `false`: Loads the data in non-strict mode. Missing files, empty columns, formatting errors, or parsing errors display a warning, and data is loaded. If there are missing columns, implicit default values are used in the absence of user-provided default values. See Data Type Default Values to learn more.

  The following example sets `is_strict_mode` to `false` in the `file` parameters.

```
mysql> CREATE EXTERNAL TABLE table_1(
        col_1 int,
        col_2 int,
        col_3 int)
        ENGINE_ATTRIBUTE='{"dialect": "format": "csv"},
                          "file": [{"oci://mybucket@mynamespace/data_files/data_file_1.csv",
                                    "is_strict_mode": false}]}';
```

- `allow_missing_files`: Supported as of MySQL 8.4.0. Handles missing files based on the selected option. This overrides the dialect parameter `is_strict_mode` for missing files. To learn about the default value, see the `allow_missing_files` dialect parameter in Common Dialect Parameters. The supported options are:

  - `true`: If any file is missing, no error occurs and data loading continues with the existing files. An error occurs if all the files are not available.

  - `false`: If any file is missing, an error occurs and data is not loaded.

  The following example uses a glob pattern specified with a uniform resource identifier (URI) and sets `allow_missing_files` to `true` in the `file` parameters. If any of the specified files in the pattern are not available, an error does not occur and data continues loading for the existing files. If all files are missing, an error occurs.

```
mysql> CREATE EXTERNAL TABLE table_1(
        col_1 int,
        col_2 int,
        col_3 int)
        ENGINE_ATTRIBUTE='{"dialect": {"format": "csv"},
                          "file": [{"uri": "oci://mybucket@mynamespace/data_files/data_file_[1-10].csv",
                                    "allow_missing_files": true}]}';
```

Event-Based Incremental Load Options

- `auto_refresh_event_source`: Supported as of MySQL 9.4.1. Specifies a Stream OCID for event-based incremental load. This string value is verified as a valid Stream resource identifier. Learn more about Resource Identifiers from *Oracle Cloud Infrastructure Documentation*. Learn how to Refresh Data Using Event-Based Incremental Load.

  The following example specifies a Stream OCID to set up event-based incremental load.

```
mysql> CREATE EXTERNAL TABLE table_1(
        col_1 int,
        col_2 int,
        col_3 int)
        ENGINE_ATTRIBUTE='{"dialect": {"format": "csv"},
                          "auto_refresh_event_source": "ocid1.stream.oc1.iad...",
                          "file": [{"uri": "oci://mybucket@mynamespace/data_files/data_file_1.csv"}]}';
```

Column Matching Options

- `match_columns_by`: Supported as of MySQL 9.5.0. Allows you to map columns from Parquet files and Avro files (as of MySQL 9.7.0) to external tables based on column order, column name using case sensitivity, or column name using case insensitivity.

  `match_columns_by` has the following requirements and specifications:

  - As of MySQL 9.7.0, the external file can be in `avro` or `parquet` format. In earlier versions, the external file must be in `parquet` format.
  - When Guided Load is enabled, you can only set `match_columns_by` to `order`. When Guided Load is disabled, you can set `match_columns_by` to `order`, `name_case_sensitive`, or `name_case insensitive`. To enable and disable Guided Load, see Load and Validate Tables Statement.
  - If there is a difference between the number of columns in the external table and the external file, you must set `is_strict_mode` to `false` (non-strict mode).
  - If the external file has more columns than the external file, implicit default values are used for the missing columns in the absence of user-provided default values. See Data Type Default Values to learn more.
  - If the external file has columns with the same name but different upppercase and lowercase charcters (for example, `col_1` and `Col_1`), Auto Parallel Load and Guided Load generate errors for duplicate column names even if you use the `name_case_sensitive` option.
  - If you create a new table with Auto Parallel Load and use the `name_case_insensitive` option, column names in the external table are created in all lowercase characters.

  `match_columns_by` has the following options:

  - `order`: The external table schema maps to the external file schema by column order. For example, the first column in the table schema maps to the first column in the file schema, and so on. This is the default value.
  - `name_case_sensitive`: The external table schema maps to the external file by column name using case sensitivity.
  - `name_case_insensitive`: The external table schema maps to the external file by column name using case insensitivity.

  The following example loads columns from a Parquet file based on name using case sensitivity. Lakehouse maps columns in the Parquet file to the columns in the external table (col_1, col_2, col_3) with the same name and case sensitivity.

```
mysql> CREATE EXTERNAL TABLE table_1(
        col_1 int,
        col_2 int,
        col_3 int)
        ENGINE_ATTRIBUTE='{"dialect": {"format": "parquet"},
                           "match_columns_by": "name_case_sensitive",
                           "file": [{"uri": "oci://mybucket@mynamespace/data_files/data_file_1.parque
```

Allow Missing Columns Option

- `allow_missing_columns`: Supported as of MySQL 9.5.2. Provides more flexibility when loading external tables by allowing some columns to be missing (in some files, or in the table as compared to files) even when `is_strict_mode` is `true`, but still having other strict mode properties such as data checks. During the initial load of a table using Auto Parallel Load, this option infers the table structure from a union of all column names from all files.

- As of MySQL 9.7.0, the external file can be in `avro` or `parquet` format. In earlier versions, the external file must be in `parquet` format.
- To enable this, you must also set `match_columns_by` to `name_case_sensitive` or `name_case_insensitive`. If you set `match_columns_by` to `name_case_sensitive` or `name_case_insensitive`, `allow_missing_columns` is set to `true` by default.
- Keep in mind that when the `ENGINE_ATTRIBUTE` `sampling` option for `HEATWAVE_LOAD` is enabled and set to `true`, Lakehouse tries to correctly create a unified schema from the data that it samples. To guarantee a correctly configured schema, you need to set `sampling` to `false`. To learn more about `HEATWAVE_LOAD` options, see HEATWAVE_LOAD.
- If you do not manually create a table before loading data, Auto Parallel Load creates a table structure from a union of all column names found across files. If a column is present in one file but not another, `NULL` or implicit default values are used in the absence of user-provided default values. See Data Type Default Values to learn more.
- If you manually create a table before loading data, Auto Parallel Load and Guided Load use the provided table definition, and do not add or remove extra columns. You can check warnings for any differences in columns found during the load.
- When Lakehouse loads data, if there are any columns that are in the table but not in a file, `NULL` or implicit default values are used in the absence of user-provided default values with applicable warning messages. See Data Type Default Values to learn more.

The following example manually creates a table, specifies files in Parquet format, enables `allow_missing_columns`, matches columns by name using case sensitivity, disables sampling, specifies all files in to load from the `data_files` directory, and then loads the table with the `HEATWAVE_LOAD` command. When you load the table with the files specified, `NULL` and implicit default values are used for any columns in the table but not in the files. Any columns that are present other than `col_1`, `col_2`, and `col_3` in the files are skipped.

```
mysql> CREATE EXTERNAL TABLE table_1(
        col_1 int,
        col_2 int,
        col_3 int)
        ENGINE_ATTRIBUTE='{"dialect": {"format": "parquet"},
                           "allow_missing_columns": true,
                           "match_columns_by": "name_case_sensitive",
                           "sampling": false,
                           "file": [{"uri": "oci://mybucket@mynamespace/data_files/"}]}';
```

Dialect Parameters

The dialect parameters are optional and are used to define the data configuration.

Common Dialect Parameters

The following dialect parameters are applicable to all file formats (`CSV`, `JSON`, `Parquet` and `Avro`).

- `format`: The file format defined in a table. You can define only one file format per table. The supported file formats are:
  - `csv`: The default file format.

    The following example specifies a CSV file.

    ```
    mysql> CREATE EXTERNAL TABLE table_1(
            col_1 int,
            col_2 int,
            col_3 int)
            ENGINE_ATTRIBUTE='{"dialect": {"format": "csv"},
                               "file": [{"uri": "oci://mybucket@mynamespace/data_files/data_file_1.csv"}]}
    ```

- `json`: Supported as of MySQL 8.4.0. Only Newline Delimited JSON files are supported. Tables created with `json` format must only have a single column that conforms to the JSON data type.

  The following example specifies a JSON file.

```
mysql> CREATE EXTERNAL TABLE table_1(
        col_1 int,
        col_2 int,
        col_3 int)
        ENGINE_ATTRIBUTE='{"dialect": {"format": "json"},
                           "file": [{"uri": "oci://mybucket@mynamespace/data_files/data_file_1.jso
```

- `parquet`: The Parquet data type.

  The following example specifies a Parquet file.

```
mysql> CREATE EXTERNAL TABLE table_1(
        col_1 int,
        col_2 int,
        col_3 int)
        ENGINE_ATTRIBUTE='{"dialect": {"format": "parquet"},
                           "file": [{"uri": "oci://mybucket@mynamespace/data_files/data_file_1.parq
```

- `avro`: The Avro data type.

  The following example specifies an Avro file.

```
mysql> CREATE EXTERNAL TABLE table_1(
        col_1 int,
        col_2 int,
        col_3 int)
        ENGINE_ATTRIBUTE='{"dialect": {"format": "avro"},
                           "file": [{"uri": "oci://mybucket@mynamespace/data_files/data_file_1.avro
```

- `delta`: Supported as of MySQL 9.5.0. Loads delta lake tables. See Delta Lake Tables and Limitations for Delta Lake Tables to learn more.

  The following example specifies a delta lake table.

```
mysql> CREATE EXTERNAL TABLE table_1(
        col_1 int,
        col_2 int,
        col_3 int)
        ENGINE_ATTRIBUTE='{"dialect": {"format": "delta"},
                           "file": [{"uri": "oci://mybucket@mynamespace/data_files/"}]}';
```

- `check_constraints`: Supported as of MySQL 8.4.0. Lakehouse validates primary key and unique key constraints during the initial load based on the selected option. The supported options are:

  - `true`: The default value. Lakehouse validates primary key and unique key constraints only during the initial load of the table. If there are subsequent loads or refreshes of the table, validation does not occur.

  - `false`: Lakehouse does not validate primary key and unique key constraints.

  The following example sets `check_constraints` to `false`.

```
mysql> CREATE EXTERNAL TABLE table_1(
        col_1 int,
        col_2 int,
        col_3 int)
        ENGINE_ATTRIBUTE='{"dialect": {"format": "csv", "check_constraints": false},
                           "file": [{"uri": "oci://mybucket@mynamespace/data_files/data_file_1.csv"}]
```

- `is_strict_mode`: Loads data in strict mode or non-strict mode, based on the selected option. By default, `is_strict_mode` is set to the `sql_mode` value. See Strict SQL Mode. The `file` common parameter `is_strict_mode` can override this setting. The supported options are:

  - `true`: Loads the data in strict mode. The data loading stops if there is an error due to missing files, empty columns, formatting errors or parsing errors.

    As of MySQL 8.4.0, the dialect parameter `is_strict_mode` applies to all file formats. Before MySQL 8.4.0, it only applies to the `CSV` file format. For `Avro` and `Parquet` file formats, use the `file` parameter `is_strict_mode` to define strict mode before MySQL 8.4.0.

  - `false`: Loads the data in non-strict mode. Missing files, empty columns, formatting errors, or parsing errors display a warning, and data is loaded. If there are missing columns, implicit default values are used in the absence of user-provided default values. See Data Type Default Values to learn more.

  - If no value is set, the setting is defined by the `sql_mode`.

  - Review Table 4.7, “Combinations of Settings for allow_missing_files and is_strict_mode”.

  This setting overrides the global `sql_mode` setting for handling missing files. The default is the value of `sql_mode`. See Strict SQL Mode. The `file` common parameter `is_strict_mode` can override this setting.

  The following example sets `is_strict_mode` to `false` in the `dialect` parameters.

```
mysql> CREATE EXTERNAL TABLE table_1(
        col_1 int,
        col_2 int,
        col_3 int)
        ENGINE_ATTRIBUTE='{"dialect": {"format": "csv", "is_strict_mode": false},
                           "file": [{"uri": "oci://mybucket@mynamespace/data_files/data_file_1.csv"}]}';
```

- `allow_missing_files`: Supported as of MySQL 8.4.0. Handles missing files based on the selected option. This overrides the dialect parameter `is_strict_mode` for missing files. The supported options are:

  - `true`: An error does not occur for any missing file, and data loading continues with the existing files. An error occurs if all files are not available.

  - `false`: If any file is missing, an error occurs and data is not loaded.

  - If no value is set, the setting is defined by `is_strict_mode`. If no value is set for `is_strict_mode`, the setting is defined by the `sql_mode`.

  - Review Table 4.7, “Combinations of Settings for allow_missing_files and is_strict_mode”.

  A missing file is defined as:

  - With the `name` parameter: There is no file with that name.

  - With the `pattern` parameter: There are no files that match the pattern.

  - With the `prefix` parameter: There are no files with that prefix.

  The following example uses a glob pattern specified with a uniform resource identifier (URI). It sets `allow_missing_files` to `true` in the `dialect` parameters. If any of the specified files in the pattern are not available, an error does not occur and data continues loading for the existing files. If all files are missing, an error occurs. If a file-specific setting is set, this may override the `allow_missing_files` setting.

```
mysql> CREATE EXTERNAL TABLE table_1(
        col_1 int,
```

```
        col_2 int,
        col_3 int)
        ENGINE_ATTRIBUTE='{"dialect": {"format": "csv", "allow_missing_files": true},
                           "file": [{"uri": "oci://mybucket@mynamespace/data_files/data_file_[1-10].c
```

Refer to the following table to review how different combinations of settings for `allow_missing_files` and `is_strict_mode` affect errors for missing files.

Table 4.7 Combinations of Settings for allow_missing_files and is_strict_mode

allow_missing_files	is_strict_mode	Possible Errors
T	T	No error generated unless all files are missing.
T	F	
T	Not set	
Not set	F	
F	T	Error generated for any missing files.
F	F	
F	Not set	
Not set	T	
Not set	Not set	Error generation depends on setting for `sql_mode`.

Dialect Parameters for CSV and JSON Files

The following dialect parameters applicable to `CSV` and `JSON` file formats.

Note

If these parameters are used with `Avro` or `Parquet` file formats, an error occurs.

- `encoding`: Defines the character encoding. The default is `"utf8mb4"`.
- `record_delimiter`: Specifies one or more characters used to delimit records. The maximum record delimiter length is 64 characters. You can add any string as a *`custom_record_delimiter`*.

  The default record delimiter for `json` is `"\n"`. The only alternative for `json` is `"\r\n"`.

  For `CSV` files, if you set the `record_delimiter` to `auto`, Auto Parallel Load automatically detects the record delimiters and composite record delimiters with field delimiters as prefixes. Supported as of MySQL 9.2.0 and set as the default value.

  The following record delimiters detected automatically by Auto Parallel Load:

  - `\r`: Carriage return.
  - `\n`: Line feed. This is the default for all MySQL versions prior to 9.2.0.
  - `\r\n` : Carriage return and line feed

Dialect Parameters for CSV and Parquet Files

The following dialect parameters are applicable to `CSV` and `Parquet` file formats.

- `embed_model_id`: Supported as of MySQL 9.3.2. Lets you specify the embedding model for `VECTOR` columns in the table. You must specify a model supported by MySQL HeatWave GenAI. To specify an embedding model for a table, you must use Auto Parallel Load to automatically create the table and load the data. See the example below. If you want to use a `CREATE EXTERNAL TABLE` statement to create the table manually and specify an embedding model, provide the embedding model as a column definition. Learn more about the Vector Data Type.

The following example uses Auto Parallel Load to set the embedding model for all `VECTOR` columns detected in the table to `minilm`. Keep in mind that if the table already exists and there are `VECTOR` columns with an embedding model already set, the existing model is not replaced with the one specified in this example.

```
mysql> SET @input_list = '[{
  "db_name": "data_db",
  "tables": [{
    "table_name": "table_1",
    "engine_attribute": {
      "dialect": {"format": "csv", "embed_model_id": "minilm"},
      "file": [{"uri": "oci://mybucket@mynamespace/data_files/data_file_1.csv"}]
    }
  }]
}]';
```

Dialect Parameters for CSV Files

The following dialect parameters are applicable only to `CSV` file formats.

- `field_delimiter`: Specifies one or more characters used to enclose fields. The maximum field delimiter length is 64 characters. You can add any string as a *`custom_field_delimiter`*.

  When the `field_delimiter` is set to `auto`, Auto Parallel Load automatically detects the field delimiters. Supported as of MySQL 9.2.0 and set as the default value.

  The following field delimiters are detected automatically by Auto Parallel Load:

  - `|`: Pipe. Default for all MySQL versions prior to 9.2.0.
  - `,`: Comma
  - `\t`: Tab
  - `;`: Semicolon

  The following example sets the `record_delimiter`, and `field_delimiter`. The backslash character, `\`, needs to be escaped twice since it is inside an JSON object.

  ```
  mysql> CREATE EXTERNAL TABLE table_1(
         col_1 int,
         col_2 int,
         col_3 int)
         ENGINE_ATTRIBUTE='{"dialect": {"format": "csv",
                                        "record_delimiter": ";",
                                        "field_delimiter": "\\r\\n"},
                                        "file": [{"uri": "oci://mybucket@mynamespace/data_files/data_file_
  ```

- `escape_character`: Specifies one or more characters used to escape special characters. The default is `"\\"`. You can add any string as a *`custom_escape_character`*. As of MySQL 9.5.0, you have the option to ignore the escape character by setting its value to an empty string.

- `quotation_marks`: Specifies one or more characters used to enclose fields. The default is `"\'"`. You can add any string as *`custom_quotation_marks`*. As of MySQL 9.5.0, you have the option to ignore the quote character by setting its value to an empty string.

  Consider the following example for setting empty strings for escape and quotation characters. You have the following data in a CSV file: `"{""doc1"":{\""doc2\"":22}}"`.

  - You create an external table and set an empty string for the escape character: `"dialect": {"escape_character": ""}`.

    ```
    mysql> CREATE EXTERNAL TABLE table_1(
           col_1 int)
           ENGINE_ATTRIBUTE='{"dialect": {"format": "csv",
    ```

```
                                        "escape_character": ""},
                                        "file": [{"uri": "oci://mybucket@mynamespace/data_files/data
```

The value loads as: `{"doc1":{\"doc2\":22}}`.

- You create an external table and set an empty string for the quotation character: `"dialect": {"quotation_marks": ""}`.

```
mysql> CREATE EXTERNAL TABLE table_1(
       col_1 int)
       ENGINE_ATTRIBUTE='{"dialect": {"format": "csv",
                                      "quotation_marks": ""},
                                      "file": [{"uri": "oci://mybucket@mynamespace/data_files/data
```

The value loads as: `"{""doc1"":{""doc2"":22}}"`.

- You create an external table and set empty strings for both the escape character and quotation character: `"dialect": {"quotation_marks": "", "escape_character": ""}`.

```
mysql> CREATE EXTERNAL TABLE table_1(
       col_1 int)
       ENGINE_ATTRIBUTE='{"dialect": {"format": "csv",
                                      "escape_character": "",
                                      "quotation_marks": ""},
                                      "file": [{"uri": "oci://mybucket@mynamespace/data_files/data
```

The value loads as: `"{""doc1"":{\""doc2\"":22}}"`.

The following example sets the `escape_character`, and `quotation_marks`. The backslash character, `\`, needs to be escaped twice since it is inside an JSON object.

```
mysql> CREATE EXTERNAL TABLE table_1(
       col_1 int,
       col_2 int,
       col_3 int)
       ENGINE_ATTRIBUTE='{"dialect": {"format": "csv",
                                      "escape_character": "\\\\",
                                      "quotation_marks": "\\""},
                                      "file": [{"uri": "oci://mybucket@mynamespace/data_files/data_f
```

- `date_format`: The date format. See `date_format`. You can also set date formats for each column by using the column `ENGINE_ATTRIBUTE` option, which overrides the format in the dialect parameter. For versions earlier than MySQL 9.5.0, this format is ignored if you load data using Lakehouse Auto Parallel Load or manually with Guided Load. As of MySQL 9.5.0, the format specified at the dialect level is used to infer the temporal data if you load data using Lakehouse Auto Parallel Load or manually with Guided Load.

- `time_format`: The time format. See String and Numeric Literals in Date and Time Context. You can also set time formats for each column by using the column `ENGINE_ATTRIBUTE` option, which overrides the format in the dialect parameter. For versions earlier than MySQL 9.5.0, this format is ignored if you load data using Lakehouse Auto Parallel Load or manually with Guided Load. As of MySQL 9.5.0, the format specified at the dialect level is used to infer the temporal data if you load data using Lakehouse Auto Parallel Load or manually with Guided Load.

- `timestamp_format`: Supported as of MySQL 9.0.1. The timestamp format. See `date_format`. You can also set timestamp formats for each column by using the column `ENGINE_ATTRIBUTE` option, which overrides the format in the dialect parameter. For versions earlier than MySQL 9.5.0, this format is ignored if you load data using Lakehouse Auto Parallel Load or manually with Guided Load. As of MySQL 9.5.0, the format specified at the dialect level is used to infer the temporal data if you load data using Lakehouse Auto Parallel Load or manually with Guided Load.

- `datetime_format`: Supported as of MySQL 9.5.1. The datetime format. You can also set datetime formats for each column by using the column `ENGINE_ATTRIBUTE` option, which overrides the format in the dialect parameter. The format specified at the dialect level is used to infer the temporal data if you load data using Lakehouse Auto Parallel Load or manually with Guided Load.

- If you set both `timestamp_format` and `datetime_format`, only the `datetime_format` is used for inference and loading of timestamp/datetime values. We recommend using `datetime_format` instead of `timestamp_format`.
- As of MySQL 9.5.0, Lakehouse Auto Parallel Load and Guided Load can infer supported formats for temporal data types by setting the appropriate dialect option to `auto`. The `auto` setting is the default value if no temporal format is specified. See About Lakehouse Auto Parallel Load Schema Inference to learn more.
- The following example sets the `date_format`, `time_format`, and `datetime_format`.

```
mysql> CREATE EXTERNAL TABLE table_1(
        col_1 date,
        col_2 time,
        col_3 datetime,
        col_4 int)
        ENGINE_ATTRIBUTE='{"dialect": {"format": "csv",
                                       "date_format": "%d/%m/%Y",
                                       "time_format": "%I:%i %p",
                                       "datetime_format": "%Y-%m-%d %H:%i:%s"},
                                       "file": [{"uri": "oci://mybucket@mynamespace/data_files/data_file_
```

  The following example sets the `date_format`, `time_format`, and `datetime_format` options to `auto`, which allows Auto Parallel Load and Guided Load to infer supported formats for temporal data types.

```
mysql> CREATE EXTERNAL TABLE table_1(
        col_1 date,
        col_2 time,
        col_3 datetime,
        col_4 int)
        ENGINE_ATTRIBUTE='{"dialect": {"format": "csv",
                                       "date_format": "auto",
                                       "time_format": "auto",
                                       "datetime_format": "auto"},
                                       "file": [{"uri": "oci://mybucket@mynamespace/data_files/data_file_
```

- `trim_spaces`: Removes/retains the leading and trailing spaces, based on the set option. The default value is `false`.

  The following example sets `trim_spaces` to `true`.

```
mysql> CREATE EXTERNAL TABLE table_1(
        col_1 int,
        col_2 int,
        col_3 int)
        ENGINE_ATTRIBUTE='{"dialect": {"format": "csv", "trim_spaces": true},
                                       "file": [{"uri": "oci://mybucket@mynamespace/data_files/data_file_
```

- `skip_rows`: The number of rows to skip at the start of the file. The default value is `0` and the maximum value is `20`.
- `has_header`: Adds a header row to the `CSV` file based on the selected option. The default is `false`.

  If `has_header` and `skip_rows` are both specified, Lakehouse first skips the number of rows, and then uses the next row as the header row.

  The following example creates a table with Auto Parallel Load and sets `skip_rows` and `has_header`. Since both parameters are set, the first row is skipped and the second row is set as the header row.

```
mysql> CREATE EXTERNAL TABLE table_1(
        col_1 int,
        col_2 int,
        col_3 int)
        ENGINE_ATTRIBUTE='{"dialect": {"format": "csv", "skip_rows": 1, "has_header": true},
                                       "file": [{"uri": "oci://mybucket@mynamespace/data_files/data_file_
```

What's Next

- Learn about Lakehouse Auto Parallel Load
- Learn how to create external tables using Auto Parallel Load or manually.
- Learn how to Refresh Data Using Event-Based Incremental Load.

4.3.7 About Lakehouse Auto Parallel Load

Lakehouse Auto Parallel Load, which extends the Auto Parallel Load feature of MySQL HeatWave, facilitates the process of loading data from Object Storage into MySQL HeatWave by automating many of the steps involved, including:

- Excluding schemas, tables, and columns that Auto Parallel Load cannot load.
- Verifying that there is sufficient memory available for the data.
- Optimizing load parallelism based on machine learning models.
- Loading data into MySQL HeatWave.
- Defining `LAKEHOUSE` as the engine for tables that MySQL HeatWave loads.
- Defining the `ENGINE_ATTRIBUTE` for tables that MySQL HeatWave loads.

Lakehouse Auto Parallel Load also includes Lakehouse Incremental Load, which can refresh tables after an initial load.

Requirements and Privileges for Auto Parallel Load

To run Auto Parallel Load in `normal` mode, the MySQL HeatWave Cluster must be active.

To use Auto Parallel Load, ask the admin user to grant you the following privileges. Replace *`user_name`* and *`database_name`* in the commands with the appropriate user name and database name.

The `PROCESS` privilege on the appropriate database in the DB System.

```
mysql> GRANT PROCESS ON database_name.* TO 'user_name'@'%';
```

The `EXECUTE` privilege on the `sys` schema.

```
mysql> GRANT EXECUTE ON sys.* TO 'user_name'@'%';
```

The `SELECT` privilege.

```
mysql> GRANT SELECT ON performance_schema.* TO 'user_name'@'%';
```

About Lakehouse Auto Parallel Load Schema Inference

Lakehouse Auto Parallel Load includes schema inference, and uses it in one of two ways:

- If the external tables do not already exist, Lakehouse Auto Parallel Load analyzes the data, infers the table structure, and creates the database and all tables. This only requires the name of the database, the names of each table, the external file parameters, and then Lakehouse Auto Parallel Load generates the `CREATE DATABASE` and `CREATE TABLE` statements.

  Lakehouse Auto Parallel Load uses header information from the external files to define the column names. If this is not available, Lakehouse Auto Parallel Load defines the column names sequentially: `col_1`, `col_2`, `col_3` ...

  As of MySQL 9.5.2, when you create a table with JSON syntax and with Auto Parallel Load, the `allow_missing_columns` table-level parameter infers the table structure from a union of all column names from all files. As of MySQL 9.7.0, this is supported for Parquet and Avro files. In earlier versions, the external file must be in Parquet format. By default, `allow_missing_columns`

is set to `true`. If there are missing columns in the files specified to load data, Lakehouse Auto Parallel Load uses default values instead of generating errors. To enable this, you must also set the `match_columns_by` table-level parameter to `name_case_sensitive` or `name_case_insensitive`. Keep in mind that when the `ENGINE_ATTRIBUTE` `sampling` option for `HEATWAVE_LOAD` is enabled and set to `true`, Lakehouse tries to correctly create a unified schema from the data that it samples. To guarantee a correctly configured schema, you need to set `sampling` to `false`. To learn more about `HEATWAVE_LOAD` options, see HEATWAVE_LOAD. To learn more about `allow_missing_columns`, see Allow Missing Columns Option.

- If the external tables already exist, Lakehouse Auto Parallel Load analyzes the data, infers the table structure, and then modifies the existing column definitions to avoid errors during data load. For example, if a table defines a column with `TINYINT`, but Lakehouse Auto Parallel Load infers that the data requires `SMALLINT` `MEDIUMINT`, `INT`, or `BIGINT`, then Lakehouse Auto Parallel Load modifies the column definitions accordingly. If `allow_missing_columns` is set to `true`, Lakehouse Auto Parallel Load does not add or drop any columns based on the inferred schema. If the inferred data type is incompatible with the table definition, Lakehouse Auto Parallel Load raises an error, and specifies the column as `NOT SECONDARY`. To learn more, see Load External Data Using Lakehouse Auto Parallel Load with an Existing Table.

  Note

  If you are on a version earlier than MySQL 8.4.0, refer to Lakehouse Auto Parallel Load with The external_tables Option for more information on the appropriate syntax to use.

- As of MySQL 9.5.0, Lakehouse Auto Parallel Load is able to infer the format of temporal data types. To do this, specify the appropriate external table option (`EXTERNAL_FORMAT`, `DATE FORMAT`, `DATETIME FORMAT`, and `TIME FORMAT` for SQL syntax, and `date_format`, `time_format`, `timestamp_format`, and `datetime_format` (as of MySQL 9.5.1) for JSON syntax) and set the option to `auto`. The `auto` setting is the default value if no temporal format is specified. To review examples, see Lakehouse External Table SQL Syntax and External Table JSON Syntax.

  Inferring formats of temporal data types has the following requirements and specifications:

  - This is only available for files in CSV format.
  - Only the temporal formats specified at the dialect level are considered during inference. Column level formats, if any, are ignored during inference.
  - To review supported values for format specifiers of all temporal formats, see `date_format`. Not including at least one supported format specifier generates an error.
  - If you use an ambiguous format to specify temporal data types (for example, `date_format` as `%m:%d:%y` and `time_format` as `%h:%i:%s`), and the data matches both formats, the following order of precedence is used for inferring the data type: `DATE` > `TIME` > `TIMESTAMP` > `DATETIME`.
  - If you use JSON syntax and set both `timestamp_format` and `datetime_format`, only the `datetime_format` is used for inference and loading of timestamp/datetime values. We recommend using `datetime_format` instead of `timestamp_format`.
  - Any extra characters in the temporal format that are not format specifiers must match exactly with the corresponding characters in the data for the temporal data type to be recognized. For example, the specified format `"timestamp_format": "abc%m:%d:%yabc%h:%i:%s"` would match `"abc3:21:2022abc8:22:44"`, but not `"3:21:2022 8:22:44"`.
  - Any setting for the strict mode option (`STRICT_LOAD` or `is_strict_mode`) does not affect inference for temporal data types. This also includes MySQL modes such as `NO_ZERO_DATE` and `NO_ZERO_IN_DATE`. Invalid and zero dates are still considered as valid temporal values during inference. The impact of the `sql_mode` is considered during the loading of tables into Lakehouse.

- If you set a specific format at the column level for a temporal data type when creating a table, and a different format is inferred when loading the table to Lakehouse, the specific format used for the column when creating the table is prioritized and used for loading the column.

Review the following table of supported formats for temporal data types. If multiple formats are valid for the inferred data, the first valid format listed in the following table is used as the inferred temporal format.

Table 4.8 Supported formats for inferred temporal data types

Temporal Data Type	Format Pattern	Example Values
`DATE`	`%Y-%m-%d`	2025-08-09 2025-08-9 2025-8-09
`DATE`	`%m/%d/%Y`	08/09/2025 08/9/2025 8/09/2025
`DATE`	`%d/%m/%Y`	09/08/2025 09/8/2025 9/08/2025
`TIME`	`%H:%i:%s.%f` (24-hour format)	22:30:05.100200
`TIME`	`%H:%i` (24-hour format)	22:30
`TIME`	`%I:%i %p` (12-hour format)	10:30 PM
`TIME`	`%I:%i:%S %p` (12-hour format)	10:30:05 PM
`TIMESTAMP`/`DATETIME`	`%Y-%m-%d %H:%i:%s.%f`	2025-08-20 14:55:30.123456
`TIMESTAMP`/`DATETIME`	`%Y-%m-%d %H:%i`	2025-08-20 14:55
`TIMESTAMP`/`DATETIME`	`%Y-%m-%dT%H:%i:%s.%f`	22025-08-20T14:55:01.000000
`TIMESTAMP`/`DATETIME`	`%Y-%m-%dT%H:%i`	2025-08-20T14:55

What's Next

- Review the following:
  - The Auto Parallel Load HEATWAVE_LOAD command
  - Lakehouse External Table Syntax
- Learn how to Create External Tables Using Lakehouse Auto Parallel Load.
- Learn how to Refresh Data Using Event-Based Incremental Load.

4.3.8 Create External Tables Using Auto Parallel Load

This topics in this section describe how to create external tables using Auto Parallel Load.

4.3.8.1 Create an External Table

You can use Auto Parallel Load to set up the automatic creation of an external table along with the loading of data from files in Object Storage. We recommend setting up Auto Parallel Load with an

`input_list` variable, and then using the `HEATWAVE_LOAD` command, which creates the external table and loads the data in one operation. To learn more about loading data, see Load Structured Data Using Lakehouse Auto Parallel Load.

This topic contains the following sections:

- Before You Begin
- Lakehouse Auto Parallel Load Syntax
- Create External Table Example
- What's Next

Before You Begin

- Prepare to create external tables by reviewing the following:
  - Additional MySQL HeatWave Lakehouse Requirements
  - MySQL HeatWave Lakehouse Privileges
  - Supported File Formats
  - Access Object Storage
- Review the relevant syntax for creating external tables and loading data:
  - Auto Parallel Load Syntax for the HEATWAVE_LOAD command
  - Lakehouse External Table Syntax

Lakehouse Auto Parallel Load Syntax

The following example shows how to create an external table with Auto Parallel Load in one command using an OCIFS URI. Since no dialect parameters are included, Lakehouse specifies a CSV file by default with all default settings. Options are set to `NULL`.

```
mysql> CALL sys.HEATWAVE_LOAD(
  '[{"db_name": "db_1",
  "tables": [{
    "table_name": "table_1",
    "engine_attribute": {
      "file": [{"uri": "oci://bucket_1@tenant_1/data_file_1.csv"}]
    }
  }]
}]', NULL);
```

While you can specify all parameters in one command, it might be easier for you to first create the `input_list` and store it in a variable. You can then pass the variable in the `HEATWAVE_LOAD` command when you load the data.

The following example uses the same command to create the external table, but creates the variable with the `input_list`, and then passes that in the `HEATWAVE_LOAD` command to load the data.

```
mysql> SET @input_list = '[{
  "db_name": "db_1",
  "tables": [{
    "table_name": "table_1",
    "engine_attribute": {
      "file": [{"uri": "oci://bucket_1@tenant_1/data_file_1.csv"}]
    }
  }]
}]';
mysql> CALL sys.HEATWAVE_LOAD(CAST(@input_list AS JSON), NULL);
```

Create External Table Example

To demonstrate how to create an external table using Lakehouse Auto Parallel Load, the following example specifies a single CSV file by using an OCIFS URI.

The CSV file in this example is from Bank Marketing. To use this file, visit Bank Marketing and download the `bank+marketing.zip` file. Unzip the file, and then unzip the `bank.zip` file. Refer to the `bank.csv` file.

To load external data using Lakehouse Auto Parallel Load:

1. Prepare the files to load in the proper format. See Supported File Formats.
2. Upload the files to load into Object Storage. See Uploading an Object Storage Object to a Bucket in *Oracle Cloud Infrastructure Documentation*.
3. Select the method to specify the files that will load the data in the external table: uniform resource identifier (URI), resource principals, or PAR. To learn more about each method, see the following:
   - Access Object Storage with Uniform Resource Identifiers and URI examples.
   - Access Object Storage with Resource Principals and resource principal examples.
   - Access Object Storage with Pre-Authenticated Requests (PAR) and PAR examples.
4. Connect to your MySQL HeatWave Database System.
5. Create a session variable with the characteristics of the `input_list` to create the external table and specify the file to use to load data into the table. To review all syntax options, see HEATWAVE_LOAD.

```
mysql> SET @input_list = '[{
  "db_name": "bank_marketing",
  "tables": [{
    "table_name": "bank_train",
    "engine_attribute": {
      "dialect": {"format": "csv", "has_header": true},
      "file": [{"uri": "oci://mybucket@mynamespace/data_files/data_file_1.csv"}]
    }
  }]
}]';
```

Where:

- `@input_list` is the name of the session variable.
- `db_name` identifies the database name to store the table: `bank_marketing`. The database is automatically created if it does not exist.
- `table_name` sets the table name to store the data: `bank_train`. The table is automatically created if it does not exist.
- `engine_attribute` defines the parameters of the external file.
- `format` defines the format of the external file: `csv`.
- `has_header` identifies a header in the external file. Auto Parallel Load then infers the column names from the first row in the file.
- `uri` sets the URI link to access the file. Replace `mybucket@mynamespace/data_files/data_file_1.csv` with your own link.

If you are on MySQL 9.1.2 and earlier, you need to update `dialect` with the `field delimiter` and `record delimiter` parameters. As of MySQL 9.2.0, MySQL HeatWave Lakehouse can automatically detect these values. See Lakehouse External Table Syntax to learn more.

6. Run the `HEATWAVE_LOAD` command with the characteristics of the `input_list` in `normal` mode to automatically create the external table and load data into the table. See Load Structured Data Using Lakehouse Auto Parallel Load for more information about loading data in `dryrun` mode to test out the creation of the table and the loading of data beforehand. Keep in mind that the `input_list` variable is valid for the current session. If you end the session, you can no longer use the variable.

```
mysql> CALL sys.HEATWAVE_LOAD(CAST(@input_list AS JSON), JSON_OBJECT('mode', 'normal'));
```

What's Next

- Review different ways to specify files when creating external tables by using Lakehouse Auto Parallel Load:
  - Using Uniform Resource Identifiers
  - Using Resource Principals
  - Using PARs
- Review how to Load Structured Data Using Lakehouse Auto Parallel Load.

4.3.8.2 Use URI to Create External Tables

As of MySQL 9.3.1, and only on MySQL HeatWave on OCI, you can use uniform resource identifiers (URI) to specify files for loading data when creating external tables. Review the following examples for different ways to create external tables.

These examples use an `input_list` variable to set up the creation of the external table and the files to load the table. To create the external table, you need to run the `HEATWAVE_LOAD` command with the `input_list`. To learn more, see Load Structured Data Using Lakehouse Auto Parallel Load.

This topic contains the following sections:

- Before You Begin
- Types of URIs
- URI Parameters
- OCIFS URI
- PAR URI
- Native URI
- What's Next

Before You Begin

- To use URIs, you require the same privileges to use pre-authenticated requests and resource prinicipals.
  - To use PAR URIs, make sure you have the required PAR Privileges.
  - To use Native URIs and OCIFS URIs, make sure you have the required Resource Principal Privileges.
- Review how to Create an External Table.

Types of URIs

There are three types of URIs supported: OCIFS, pre-authenticated request (PAR), and Native URI, and you can specify a name of a file as your source in Object Storage, or a prefix, or a pattern.

When you define the `object_path` of a URI, the name, prefix, or glob pattern is inferred by the following.

- Name: If the `object_path` is neither a glob pattern nor prefix.
- Prefix: If the `object_path` is not a glob pattern and ends with an unencoded `/` character, such as a folder path.
- Glob pattern: If the `object_path` contains at least one of the following unencoded characters: `?`, `*`, or `[`. To use these characters as literals, you need to escape them or encode them as needed depending on the URI syntax. Regex patterns are not supported. See Glob Patterns from the *Oracle Cloud Infrastructure Documentation* to learn more.

URI Parameters

Depending on the type of URI you use, you need to ensure that the following parameters are present in the URI.

- `bucket_name`: Buckets allow you to store objects in a compartment. This parameter is needed for all three types of URIs. To learn more about buckets, see Object Storage Buckets in the *Oracle Cloud Infrastructure Documentation*.
- `namespace_name`: This is the top-level container for all buckets and objects. This parameter is needed for all three types of URIs. To learn how to view your namespace, see Object Storage Namespaces in the *Oracle Cloud Infrastructure Documentation*.
- `region_name`: The region identifier for your OCI tenancy. This parameter is needed for PAR URI and Native URI. For OCIFS URI, it is set to the region of the instance running Lakehouse. To view the list of available regions and region identifiers, see Regions and Availability Domains in the *Oracle Cloud Infrastructure Documentation*.

Note

If you are on MySQL 9.1.2 and earlier, you need to update `dialect` with the `field delimiter` and `record delimiter` parameters. As of MySQL 9.2.0, MySQL HeatWave Lakehouse can automatically detect these values. See Lakehouse External Table Syntax to learn more.

To review all syntax options for creating external tables and loading external files, see HEATWAVE_LOAD and Lakehouse External Table Syntax.

OCIFS URI

The format of an OCIFS URI is the following:

```
oci://bucket_name@namespace_name/object_path
```

The `object_path` in an OCIFS URI cannot be empty. It also does not have a region parameter. The instance's associated region is used as the default region. See OCIFS Documentation to learn more about OCIFS and its specifications.

The following example specifies a single file as a name since the `object_path` is not a glob pattern or prefix. The example specifies the `data_file_1.csv` file.

```
mysql> SET @input_list = '[{
  "db_name": "lakehouse_db",
  "tables": [{
    "table_name": "table_1",
```

```
    "engine_attribute": {
      "dialect": {"format": "csv"},
      "file": [{"uri": "oci://mybucket@mynamespace/data_files/data_file_1.csv"}]
    }
  }]
}]';
```

In the following example the `object_path` is encoded, so it is treated as a name instead of a prefix. The original file name is `data_file_[1].csv`.

```
 mysql> SET @input_list = '[{
  "db_name": "lakehouse_db",
  "tables": [{
    "table_name": "table_1",
    "engine_attribute": {
      "dialect": {"format": "csv"},
      "file": [{"uri": "oci://mybucket@mynamespace/data_files/data_file_%5B1%5D.csv"}]
    }
  }]
}]';
```

The following example uses a prefix to specify files because the `object_path` ends with a `/` character and does not contain any of the following characters: `*`, `?`, or `[`. The example uses all files in the `data_files/` Object Storage folder to load the external table.

```
mysql> SET @input_list = '[{
  "db_name": "lakehouse_db",
  "tables": [{
    "table_name": "table_1",
    "engine_attribute": {
      "dialect": {"format": "csv"},
      "file": [{"uri": "oci://mybucket@mynamespace/data_files/"}]
    }
  }]
}]';
```

The following example uses as a glob pattern and specifies two files, `data_file_1.csv` and `data_file_2.csv`. The `object_path` is a pattern because it uses a `[` character.

```
mysql> SET @input_list = '[{
  "db_name": "lakehouse_db",
  "tables": [{
    "table_name": "table_1",
    "engine_attribute": {
      "dialect": {"format": "csv"},
      "file": [{"uri": "oci://mybucket@mynamespace/data_files/data_file_[1-2].csv"}]
    }
  }]
}]';
```

The following example uses a glob pattern because it uses an unencoded `*` character. The example specifies all CSV files that start with `data_file_` to load the external table. For example, it uses the files `data_file_1.csv`, `data_file_2.csv`, and `data_file_3.csv`.

```
mysql> SET @input_list = '[{
  "db_name": "lakehouse_db",
  "tables": [{
    "table_name": "table_1",
    "engine_attribute": {
      "dialect": {"format": "csv"},
      "file": [{"uri": "oci://mybucket@mynamespace/data_files/data_files/data_file_*.csv"}]
    }
  }]
}]';
```

The following example uses a glob pattern because it has an unencoded `?` character. The example specifies all CSV files that start with `data_file_` and have any single character followed by `0` to load the external table. For example, it uses the files `data_file_10.csv`, `data_file_20.csv`, and `data_file_30.csv`.

```
mysql> SET @input_list = '[{
  "db_name": "lakehouse_db",
  "tables": [{
    "table_name": "table_1",
    "engine_attribute": {
      "dialect": {"format": "csv"},
      "file": [{"uri": "oci://mybucket@mynamespace/data_files/data_files/data_file_?0.csv"}]
    }
  }]
}]';
```

The following example creates two external tables and specifies one file for each table.

```
mysql> SET @input_list = '[{
  "db_name": "lakehouse_db",
  "tables": [{
    "table_name": "table_1",
    "engine_attribute": {
      "dialect": {"format": "csv"},
      "file": [{"uri": "oci://mybucket@mynamespace/data_files/data_file_1.csv"}]
    }
  },
  {
    "table_name": "table_2",
    "engine_attribute": {
      "dialect": {"format": "csv"},
      "file": [{"uri": "oci://mybucket@mynamespace/data_files/data_file_2.csv"}]
    }
  }]
}]';
```

PAR URI

The format of a PAR URI is the following if using dedicated endpoints:

```
https://namespace_name.objectstorage.region_name.oci.customer-oci.com/p/PAR_token/n/namespace_name/b/bu
```

The format of a PAR URI can also be the following:

```
https://objectstorage.region_name.oraclecloud.com/p/PAR_token/n/namespace_name/b/bucket_name/o/object_p
```

The `object_path` in a PAR URI can be empty. If the `object_path` is empty, the PAR URI is treated as a PAR with no additional pattern, prefix, or name target.

For more information about pre-authenticated requests, see Object Storage Pre-Authenticated Requests in *Oracle Cloud Infrastructure Documentation*.

The following example specifies a single file as a name since the `object_path` is not a glob pattern or prefix. The file `data_file_2.csv` is used to load the external table.

```
mysql> SET @input_list = '[{
  "db_name": "lakehouse_db",
  "tables": [{
    "table_name": "table_1",
    "engine_attribute": {
      "dialect": {"format": "csv"},
      "file": [{"uri": "https://mynamespace.objectstorage.myregion.oci.customer-oci.com/p/.../n/mynames
    }
  }]
}]';
```

In the following example the `object_path` is encoded, so it is treated as a name instead of a prefix. The original file name is `data_file_[1].csv`.

```
mysql> SET @input_list = '[{
  "db_name": "lakehouse_db",
  "tables": [{
    "table_name": "table_1",
    "engine_attribute": {
```

```
      "dialect": {"format": "csv"},
      "file": [{"uri": "https://mynamespace.objectstorage.myregion.oci.customer-oci.com/p/.../n/mynamespace
    }
  }]
}]';
```

In the following example, the `object_path` is empty, which means that no additional target is specified. Therefore, the example uses all the files under the PAR to load the external table.

```
mysql> SET @input_list = '[{
  "db_name": "lakehouse_db",
  "tables": [{
    "table_name": "table_1",
    "engine_attribute": {
      "dialect": {"format": "csv"},
      "file": [{"uri": "https://mynamespace.objectstorage.myregion.oci.customer-oci.com/p/.../n/mynamespace
    }
  }]
}]';
```

The following example uses a prefix to specify files because the `object_path` ends with a `/` character and does not contain any of the following characters: `*`, `?`, or `[`. The example uses all files in the `data_files/` Object Storage folder to load the external table.

```
mysql> SET @input_list = '[{
  "db_name": "lakehouse_db",
  "tables": [{
    "table_name": "table_1",
    "engine_attribute": {
      "dialect": {"format": "csv"},
      "file": [{"uri": "https://mynamespace.objectstorage.myregion.oci.customer-oci.com/p/.../n/mynamespace
    }
  }]
}]';
```

The following example uses a glob pattern and specifies two files, `data_file_2.csv` and `data_file_3.csv`, to load the external table. The `object_path` is a pattern because it has a `[` character.

```
mysql> SET @input_list = '[{
  "db_name": "lakehouse_db",
  "tables": [{
    "table_name": "table_1",
    "engine_attribute": {
      "dialect": {"format": "csv"},
      "file": [{"uri": "https://mynamespace.objectstorage.myregion.oci.customer-oci.com/p/.../n/mynamespace
    }
  }]
}]';
```

The following example uses a glob pattern because it has an unencoded `*` character. The example specifies all CSV files that start with `data_file_` to load the external table. For example, it uses the files `data_file_1.csv`, `data_file_2.csv`, and `data_file_3.csv`.

```
mysql> SET @input_list = '[{
  "db_name": "lakehouse_db",
  "tables": [{
    "table_name": "table_1",
    "engine_attribute": {
      "dialect": {"format": "csv"},
      "file": [{"uri": "https://mynamespace.objectstorage.myregion.oci.customer-oci.com/p/.../n/mynamespace
    }
  }]
}]';
```

The following example uses a glob pattern because it has an unencoded `?` character. The example specifies all CSV files that start with `data_file_` and have any single character followed by `0` to load the external table. For example, it uses the files `data_file_10.csv`, `data_file_20.csv`, and `data_file_30.csv`.

```
mysql> SET @input_list = '[{
  "db_name": "lakehouse_db",
  "tables": [{
    "table_name": "table_1",
    "engine_attribute": {
      "dialect": {"format": "csv"},
      "file": [{"uri": "https://mynamespace.objectstorage.myregion.oci.customer-oci.com/p/.../n/mynames
    }
  }]
}]';
```

The following example creates two external tables and specifies one file for each table.

```
mysql> SET @input_list = '[{
  "db_name": "lakehouse_db",
  "tables": [{
    "table_name": "table_1",
    "engine_attribute": {
      "dialect": {"format": "csv"},
      "file": [{"uri": "https://mynamespace.objectstorage.myregion.oci.customer-oci.com/p/.../n/mynames
    }
  },
  {
    "table_name": "table_2",
    "engine_attribute": {
      "dialect": {"format": "csv"},
      "file": [{"uri": "https://mynamespace.objectstorage.myregion.oci.customer-oci.com/p/.../n/mynames
    }
  }]
}]';
```

Native URI

The format of a Native URI is the following if using dedicated endpoints:

```
https://namespace_name.objectstorage.region_name.oci.customer-oci.com/n/namespace_name/b/bucket_name/o/
```

The format of a Native URI can also be the following:

```
https://objectstorage.region_name.oraclecloud.com/n/namespace_name/b/bucket_name/o/object_path
```

The `object_path` cannot be empty.

In the following example specifies a single file as a name since the `object_path` is not a glob pattern or prefix. The example uses the `data_file_1.csv` file to load the external table.

```
mysql> SET @input_list = '[{
  "db_name": "lakehouse_db",
  "tables": [{
    "table_name": "table_1",
    "engine_attribute": {
      "dialect": {"format": "csv"},
      "file": [{"uri": "https://mynamespace.objectstorage.myregion.oci.customer-oci.com/n/mynamespace/b
    }
  }]
}]';
```

In the following example the `object_path` is encoded, so it is treated as a name instead of a prefix. The original file name is `data_file_[1].csv`.

```
mysql> SET @input_list = '[{
  "db_name": "lakehouse_db",
  "tables": [{
    "table_name": "table_1",
    "engine_attribute": {
      "dialect": {"format": "csv"},
      "file": [{"uri": "https://mynamespace.objectstorage.myregion.oci.customer-oci.com/n/mynamespace/b
    }
  }]
}]';
```

The following example uses a prefix to specify files because the `object_path` ends with a `/` character and does not contain any of the following characters: `*`, `?`, or `[`. The example uses all files in the `data_files/` Object Storage folder to load the external table.

```
mysql> SET @input_list = '[{
  "db_name": "lakehouse_db",
  "tables": [{
    "table_name": "table_1",
    "engine_attribute": {
      "dialect": {"format": "csv"},
      "file": [{"uri": "https://mynamespace.objectstorage.myregion.oci.customer-oci.com/n/mynamespace/b/myb
    }
  }]
}]';
```

The following example uses a glob pattern and specifies two files, `data_file_1.csv` and `data_file_2.csv`, to load the external table. The `object_path` is a pattern because it has a `[` character.

```
mysql> SET @input_list = '[{
  "db_name": "lakehouse_db",
  "tables": [{
    "table_name": "table_1",
    "engine_attribute": {
      "dialect": {"format": "csv"},
      "file": [{"uri": "https://mynamespace.objectstorage.myregion.oci.customer-oci.com/n/mynamespace/b/myb
    }
  }]
}]';
```

The following example uses a glob pattern because it has an unencoded `*` character. The example specifies all CSV files that start with `data_file_` to load the external table. For example, it uses the files `data_file_1.csv`, `data_file_2.csv`, and `data_file_3.csv`.

```
mysql> SET @input_list = '[{
  "db_name": "lakehouse_db",
  "tables": [{
    "table_name": "table_1",
    "engine_attribute": {
      "dialect": {"format": "csv"},
      "file": [{"uri": "https://mynamespace.objectstorage.myregion.oci.customer-oci.com/n/mynamespace/b/myb
    }
  }]
}]';
```

The following example uses a glob pattern because it has an unencoded `?` character. The example specifies all CSV files that start with `data_file_` and have any single character followed by `0`. For example, it uses the files `data_file_10.csv`, `data_file_20.csv`, and `data_file_30.csv`.

```
mysql> SET @input_list = '[{
  "db_name": "lakehouse_db",
  "tables": [{
    "table_name": "table_1",
    "engine_attribute": {
      "dialect": {"format": "csv"},
      "file": [{"uri": "https://mynamespace.objectstorage.myregion.oci.customer-oci.com/n/mynamespace/b/myb
    }
  }]
}]';
```

The following example creates two external tables and specifies one file for each table.

```
mysql> SET @input_list = '[{
  "db_name": "lakehouse_db",
  "tables": [{
    "table_name": "table_1",
    "engine_attribute": {
      "dialect": {"format": "csv"},
      "file": [{"uri": "https://mynamespace.objectstorage.myregion.oci.customer-oci.com/n/mynamespace/b/myb
    }
```

```
  },
  {
    "table_name": "table_2",
    "engine_attribute": {
      "dialect": {"format": "csv"},
      "file": [{"uri": "https://mynamespace.objectstorage.myregion.oci.customer-oci.com/n/mynamespace/b
    }
  }]
}]';
```

What's Next

After successfully creating external tables and specifying the files to load data into the table, learn how to Load Structured Data Using Auto Parallel Load.

4.3.8.3 Use Resource Principals to Create External Tables

Review the following examples to see the different ways to use resource principals and Auto Parallel Load to specify files when creating external tables.

These examples use an `input_list` variable to set up the creation of the external table and the files to load the table. To create the external table, you need to run the `HEATWAVE_LOAD` command with the `input_list`. To learn more, see Load Structured Data Using Lakehouse Auto Parallel Load.

This topic contains the following sections:

- Before You Begin
- Resource Principals Parameters
- Resource Principals Examples
- Specify a Single File
- Specify Multiple Files for One External Table
- Specify Multiple Files for Multiple Tables
- Specify Files with a Pattern
- What's Next

Before You Begin

- Review the requirements to Access Object Storage with Resource Principals.
- Review how to Create an External Table.

Resource Principals Parameters

To set up resource principals, you configure the following parameters:

- `bucket_name`: Buckets allow you to storage objects in a compartment. To learn more about buckets, see Object Storage Buckets in *Oracle Cloud Infrastructure Documentation*.
- `namespace_name`: This is the top-level container for all buckets and objects. To learn how to view your namespace, see Object Storage Namespaces in *Oracle Cloud Infrastructure Documentation*.
- `region_name`: You need to define the region identifier for your OCI tenancy. To view the list of available regions and region identifiers, see Regions and Availability Domains in *Oracle Cloud Infrastructure Documentation*.

  As of MySQL 9.2.1, if the region is not specified, the instance's associated region is used as the default region. Otherwise, the specified region is used.
- `name`: Use this to specify an Object Storage file name.

- `pattern`: Use this to set a regular expression that defines a set of Object Storage files. The pattern follows the modified Modified ECMAScript regular expression grammar.
- `prefix`: Use this to define a set of Object Storage folders and files.

Resource Principals Examples

The following examples use these parameters to create the external tables and specify the files to load the tables with. Replace the values in the examples with your own.

- `SET @input_list` creates a session variable that stores all the parameters for creating the external file and loading the table with the specified files.
- `db_name` identifies the database name to store the table. MySQL HeatWave automatically creates the database if it does not exist.
- `table_name` sets the table name to store the data. MySQL HeatWave automatically creates the table if it does not exist.
- `engine_attribute` defines the parameters of the specified files.
- `dialect` defines the format options of the specified files.
- The `file` parameters define the resource principal details.
  - `region`: The region the tenant resides in.
  - `namespace`: The name of the tenancy.
  - `bucket`: The name of the Object Storage bucket.
  - `name`: The name of the external file to load.
  - `prefix`: The prefix for a set of Object Storage folders and files.
  - `pattern:` The regular expression that defines a set of Object Storage files.

Note

If you are on MySQL 9.1.2 and earlier, you need to update `dialect` with the `field delimiter` and `record delimiter` parameters. As of MySQL 9.2.0, MySQL HeatWave Lakehouse can automatically detect these values. See Lakehouse External Table Syntax to learn more.

To review all syntax options for creating external tables and specifying files, see HEATWAVE_LOAD and Lakehouse External Table Syntax.

Specify a Single File

The following example specifies a single file.

```
mysql> SET @input_list = '[{
  "db_name": "lakehouse_db",
  "tables": [{
    "table_name": "table_1",
    "engine_attribute": {
      "dialect": {"format": "csv"},
      "file": [{
        "region": "us-ashburn-1",
        "namespace": "tenant_1",
        "bucket": "bucket_1",
        "name": "data_file_1.csv"}]
    }
  }]
}]';
```

Specify Multiple Files for One External Table

To load multiple external files into one table, you can do one of the following:

- Create a folder in the Object Storage bucket and upload the required files into that folder. See Managing Folders in an Object Storage Bucket in *Oracle Cloud Infrastructure Documentation*.
- Create `file` items as needed to identify the files to load into the table.

The following example specifies all the files uploaded to a folder in the Object Storage bucket for one table.

```
mysql> SET @input_list = '[{
  "db_name": "lakehouse_db",
  "tables": [{
    "table_name": "table_1",
    "engine_attribute": {
      "dialect": {"format": "csv"},
      "file": [{
        "region": "us-ashburn-1",
        "namespace": "tenant_1",
        "bucket": "bucket_1",
        "prefix": "data_files/"}]
    }
  }]
}]';
```

The following example specifies two files for one table.

```
mysql> SET @input_list = '[{
  "db_name": "lakehouse_db",
  "tables": [{
    "table_name": "table_1",
    "engine_attribute": {
      "dialect": {"format": "csv"},
      "file": [{
        "region": "us-ashburn-1",
        "namespace": "tenant_1",
        "bucket": "bucket_1",
        "name": "data_file_1.csv"
    },
    {
        "region": "us-ashburn-1",
        "namespace": "tenant_1",
        "bucket": "bucket_1",
        "name": "data_file_2.csv"}]
    }
  }]
}]';
```

Specify Multiple Files for Multiple Tables

To specify multiple external files for multiple tables with one command, you can create the necessary number of `tables` items and do one of the following:

- Create folders as necessary in the Object Storage bucket and upload the required files into that folder. See Managing Folders in an Object Storage Bucket in *Oracle Cloud Infrastructure Documentation*.
- Create `file` items as needed to specify the files for the table.

The following example specifies two files for two tables.

```
mysql> SET @input_list = '[{
  "db_name": "lakehouse_db",
  "tables": [{
    "table_name": "table_1",
    "engine_attribute": {
      "dialect": {"format": "csv"},
```

```
      "file": [{
        "region": "us-ashburn-1",
        "namespace": "tenant_1",
        "bucket": "bucket_1",
        "name": "data_file_1.csv"}]
    }
  },
  {
    "table_name": "table_2",
    "engine_attribute": {
      "dialect": {"format": "csv"},
      "file": [{
        "region": "us-ashburn-1",
        "namespace": "tenant_1",
        "bucket": "bucket_1",
        "name": "data_file_2.csv"}]
    }
  }]
}]';
```

The following example specifies all the files in the identified folders for the tables.

```
mysql> SET @input_list = '[{
  "db_name": "lakehouse_db",
  "tables": [{
    "table_name": "table_1",
    "engine_attribute": {
      "dialect": {"format": "csv"},
      "file": [{
        "region": "us-ashburn-1",
        "namespace": "tenant_1",
        "bucket": "bucket_1",
        "prefix": "data_files_1/"}]
    }
  },
  {
    "table_name": "table_2",
    "engine_attribute": {
      "dialect": {"format": "csv"},
      "file": [{
        "region": "us-ashburn-1",
        "namespace": "tenant_1",
        "bucket": "bucket_1",
        "prefix": "data_files_2/"}]
    }
  }]
}]';
```

Specify Files with a Pattern

You can add a `pattern` parameter to use regular expression to specify specific files.

The regular expression syntax requires certain characters to have an escape character.

The escape character is the backslash character, and it is a reserved character in both JSON and MySQL. Therefore, it is necessary to escape the backslash character twice, and specify `\\` for both JSON and MySQL.

However, the regular expression escape sequence depends upon the `NO_BACKSLASH_ESCAPES` SQL mode:

- Use `\\.` to escape a period if `NO_BACKSLASH_ESCAPES` is enabled.
- Use `\\\\.` to escape a period if `NO_BACKSLASH_ESCAPES` is not enabled. The following examples use this sequence because it is the default mode.

See the following to learn more:

- Regular Expression Syntax

- Modified ECMAScript regular expression grammar

The following example specifies all files in the `data_files` folder that have a numerical suffix of one or more digits. For example, it uses the files `data_file_1.csv`, `data_file_2.csv`, and `data_file_3.csv` to load the external table.

```
mysql> SET @input_list = '[{
  "db_name": "lakehouse_db",
  "tables": [{
    "table_name": "table_1",
    "engine_attribute": {
      "dialect": {"format": "csv"},
      "file": [{
        "region": "us-ashburn-1",
        "namespace": "tenant_1",
        "bucket": "bucket_1",
        "pattern": "data_files/data_file_\\\\d+\\\\.csv"}]
    }
  }]
}]';
```

The following examples specifies all files in the `data_files` folder that have an alphabetical suffix of one or more lowercase characters. For example, it uses the files `data_file_a.csv`, `data_file_b.csv`, and `data_file_c.csv` to load the external table.

```
mysql> SET @input_list = '[{
  "db_name": "lakehouse_db",
  "tables": [{
    "table_name": "table_1",
    "engine_attribute": {
      "dialect": {"format": "csv"},
      "file": [{
        "region": "us-ashburn-1",
        "namespace": "tenant_1",
        "bucket": "bucket_1",
        "pattern": "data_files/data_file_[a-z]+\\\\.csv"}]
    }
  }]
}]';
```

To update the previous example to include uppercase and lowercase characters, replace `[a-z]` with `[A-Za-z]`.

The following example specifies all files in the `data_files` folder that have a numerical suffix that end in `0` with one preceding digit. For example, it uses the files `data_file_10.csv`, `data_file_20.csv`, and `data_file_30.csv` to load the external table.

```
mysql> SET @input_list = '[{
  "db_name": "lakehouse_db",
  "tables": [{
    "table_name": "table_1",
    "engine_attribute": {
      "dialect": {"format": "csv"},
      "file": [{
        "region": "us-ashburn-1",
        "namespace": "tenant_1",
        "bucket": "bucket_1",
        "pattern": "data_files/data_file_\\\\d0\\\\.csv"}]
    }
  }]
}]';
```

To update the previous example to load files with one or more digits preceding the 0, update `\\\\d0\\\\` with `\\\\d+0\\\\`.

What's Next

After successfully creating external tables and specifying the files to load data into the table, learn how to Load Structured Data Using Auto Parallel Load.

4.3.8.4 Use PAR to Create External Tables

Review the following examples to see the different ways to use pre-authenticated requests (PAR) and Auto Parallel Load to specify files when creating external tables.

These examples use an `input_list` variable to set up the creation of the external table and the files to load the table. To create the external table, you need to run the `HEATWAVE_LOAD` command with the `input_list`. To learn more, see Load Structured Data Using Lakehouse Auto Parallel Load.

This topic contains the following sections:

- Before You Begin
- Types of PARs
- PAR Recommendations
- PAR Parameters
- PAR Examples
- Specify a Single File
- Specify Multiple Files for One External Table
- Specify Multiple Files for Multiple Tables
- Specify Files with a Prefix
- Specify Files with a Pattern
- What's Next

Before You Begin

- Review the requirements and recommendations to Access Object Storage with Pre-Authenticated Requests.
- Review how to Create an External Table.

Types of PARs

You can create the following types of PARs:

- Bucket or prefix PAR: This type of PAR ends with `/o/`. You can specify multiple files with this PAR. You can also use `name`, `prefix`, or `pattern` parameters to specify files in the bucket or Object Storage folder.
- Object PAR: This type of PAR ends with `/o/object_name`. This PAR specifies individual files. You cannot use `name`, `prefix`, or `pattern` parameters with this PAR.

PAR Recommendations

When creating PARs consider the following recommendations:

- Only use read-only PARs.
- Set a short expiration date for the PAR URL that matches the data loading plan.
- Do not make a PAR URL publicly accessible.
- If the target defines a bucket or uses a prefix or pattern:
  - Use **Enable Object Listing** when creating the PAR in the Oracle Cloud Infrastructure (OCI) console.

- When creating the PAR from the command line, include the `--access-type AnyObjectRead` parameter.

Note

Use a resource principal for access to more sensitive data in Object Storage as it is more secure. See Access Object Storage with Resource Principals.

PAR Parameters

To set up PARs, you configure the following parameters:

- `par`: Provide the PAR URL.
- `name`: Use this to specify a file for an object or bucket PAR.
- `pattern`: Use this to set a regular expression that defines a set of Object Storage files. The pattern follows the modified Modified ECMAScript regular expression grammar.
- `prefix`: Use this to specify a set of Object Storage folders and files.

PAR Examples

The following examples use these parameters to create the external tables and specify the files to load the tables with. Replace the values in the examples with your own.

- `SET @input_list` creates a session variable that stores all the parameters for creating the external file and loading the table with the specified files.
- `db_name` identifies the database name to store the table. MySQL HeatWave automatically creates the database if it does not exist.
- `table_name` sets the table name to store the data. MySQL HeatWave automatically creates the table if it does not exist.
- `engine_attribute` defines the parameters of the specified files.
- `dialect` defines the format options of the specified files.
- `file` defines the file or files to load.

Note

If you are on MySQL 9.1.2 and earlier, you need to update `dialect` with the `field delimiter` and `record delimiter` parameters. As of MySQL 9.2.0, MySQL HeatWave Lakehouse can automatically detect these values. See Lakehouse External Table Syntax to learn more.

To review all syntax options for creating external tables and specifying files, see HEATWAVE_LOAD and Lakehouse External Table Syntax.

Specify a Single File

To specify a single file with a PAR, you can create a pre-authenticated request for that individual file or specify the file in the command.

The following example specifies a single file by using a PAR for that file.

```
mysql> SET @input_list = '[{
  "db_name": "lakehouse_db",
  "tables": [{
    "table_name": "table_1",
    "engine_attribute": {
      "dialect": {"format": "csv"},
```

```
      "file": [{"par": "https://objectstorage.us-ashburn-1.oraclecloud.com/p/.../n/tenancy_1/b/bucket_1/o/d
    }
  }]
}]';
```

The following example specifies a single file by using a PAR for a bucket and naming the file in the `name` parameter. The file is in the `data_files` Object Storage folder.

```
mysql> SET @input_list = '[{
  "db_name": "lakehouse_db",
  "tables": [{
    "table_name": "table_1",
    "engine_attribute": {
      "dialect": {"format": "csv"},
      "file": [{
        "par": "https://objectstorage.us-ashburn-1.oraclecloud.com/p/.../n/tenancy_1/b/bucket_1/o/",
        "name": "data_files/data_file_1.csv"}]
    }
  }]
}]';
```

Specify Multiple Files for One External Table

If you want to specify multiple files for one external table, you can do one of the following:

- Create a folder in the Object Storage bucket and upload the required files into that folder. See Managing Folders in an Object Storage Bucket in *Oracle Cloud Infrastructure Documentation*.
- Create a PAR for a bucket or folder, and then specify each file to load with the `name` parameter.
- Create a PAR for each file to load, and then specify the files as separate items in the `ENGINE_ATTRIBUTE` parameter.

To specify multiple external files for one external table, you can create a folder in the Object Storage bucket and upload the required files into that folder. See Managing Folders in an Object Storage Bucket in *Oracle Cloud Infrastructure Documentation*.

If you create a folder with the files for the external table, create a PAR for that folder and do the following:

- For the `Pre-Authenticated Request Target`, select **Objects with prefix**.
- Use **Enable Object Listing**.

The following example specifies all the files uploaded to the folder in the Object Storage bucket for one table.

```
mysql> SET @input_list = '[{
  "db_name": "lakehouse_db",
  "tables": [{
    "table_name": "table_1",
    "engine_attribute": {
      "dialect": {"format": "csv"},
      "file": [{"par": "https://objectstorage.us-ashburn-1.oraclecloud.com/p/.../n/tenancy_1/b/bucket_1/o/'
    }
  }]
}]';
```

The following example uses one PAR for a bucket and specifies two files with the `name` parameter.

```
mysql> SET @input_list = '[{
  "db_name": "lakehouse_db",
  "tables": [{
    "table_name": "table_1",
    "engine_attribute": {
      "dialect": {"format": "csv"},
      "file": [{
        "par": "https://objectstorage.us-ashburn-1.oraclecloud.com/p/.../n/tenant_1/b/bucket_1/o/",
        "name": "data_files/data_file_1.csv"
```

```
    },
    {
        "par": "https://objectstorage.us-ashburn-1.oraclecloud.com/p/.../n/tenant_1/b/bucket_1/o/",
        "name": "data_files/data_file_2.csv"}]
    }
  }]
}]';
```

The following example specifies two separate object PARs to load two files.

```
mysql> SET @input_list = '[{
  "db_name": "lakehouse_db",
  "tables": [{
    "table_name": "table_1",
    "engine_attribute": {
      "dialect": {"format": "csv"},
      "file": [{"par": "https://objectstorage.us-ashburn-1.oraclecloud.com/p/.../n/tenant_1/b/bucket_1/
               {"par": "https://objectstorage.us-ashburn-1.oraclecloud.com/p/.../n/tenant_1/b/bucket_1/
    }
  }]
}]';
```

Specify Multiple Files for Multiple Tables

To specify multiple files for multiple external tables with one command, create a pre-authenticated for each required file and create the necessary number of `tables` items.

The following example specifies two files for two tables.

```
mysql> SET @input_list = '[{
  "db_name": "lakehouse_db",
  "tables": [{
    "table_name": "table_1",
    "engine_attribute": {
      "dialect": {"format": "csv"},
      "file": [{"par": "https://objectstorage.us-ashburn-1.oraclecloud.com/p/.../n/tenant_1/b/bucket_1/
    }
  },
  {
    "table_name": "table_2",
    "engine_attribute": {
      "dialect": {"format": "csv"},
      "file": [{"par": "https://objectstorage.us-ashburn-1.oraclecloud.com/p/.../n/tenant_1/b/bucket_1/
    }
  }]
}]';
```

Specify Files with a Prefix

You can add a `prefix` parameter to a PAR to specify specific files. To do this, create a PAR for the folder that stores the files to load.

The following example specifies all files in the `data_files` folder that begin with `data_file_`. For example, it uses the files `data_file_1`, `data_file_2`, and `data_file_3` to load the external table.

```
mysql> SET @input_list = '[{
  "db_name": "lakehouse_db",
  "tables": [{
    "table_name": "table_1",
    "engine_attribute": {
      "dialect": {"format": "csv"},
      "file": [{"par": "https://objectstorage.us-ashburn-1.oraclecloud.com/p/.../n/tenancy_1/b/bucket_1
               "prefix": "data_files/data_file_"}]
    }
  }]
}]';
```

Specify Files with a Pattern

You can add a `pattern` parameter to a PAR and use regular expression to specify files.

The regular expression syntax requires certain characters to have an escape character.

The escape character is the backslash character, and it is a reserved character in both JSON and MySQL. Therefore, it is necessary to escape the backslash character twice, and specify `\\` for both JSON and MySQL.

However, the regular expression escape sequence depends upon the `NO_BACKSLASH_ESCAPES` SQL mode:

- Use `\\.` to escape a period if `NO_BACKSLASH_ESCAPES` is enabled.
- Use `\\\\.` to escape a period if `NO_BACKSLASH_ESCAPES` is not enabled. The following examples use this sequence because it is the default mode.

See the following to learn more:

- Regular Expression Syntax
- Modified ECMAScript regular expression grammar

The following example specifies all files in the `data_files` folder that have a numerical suffix of one or more digits. For example, it uses the files `data_file_1.csv`, `data_file_2.csv`, and `data_file_3.csv` to load the external table.

```
mysql> SET @input_list = '[{
  "db_name": "lakehouse_db",
  "tables": [{
    "table_name": "table_1",
    "engine_attribute": {
      "dialect": {"format": "csv"},
      "file": [{"par": "https://objectstorage.us-ashburn-1.oraclecloud.com/p/.../n/tenancy_1/b/bucket_1/o/'
               "pattern": "data_files/data_file_\\\\d+\\\\.csv"}]
    }
  }]
}]';
```

The following examples specifies all files in the `data_files` folder that have an alphabetical suffix of one or more lowercase characters. For example, it uses the files `data_file_a.csv`, `data_file_b.csv`, and `data_file_c.csv` to load the external table.

```
mysql> SET @input_list = '[{
  "db_name": "lakehouse_db",
  "tables": [{
    "table_name": "table_1",
    "engine_attribute": {
      "dialect": {"format": "csv"},
      "file": [{"par": "https://objectstorage.us-ashburn-1.oraclecloud.com/p/.../n/tenancy_1/b/bucket_1/o/'
               "pattern": "data_files/data_file_[a-z]+\\\\.csv"}]
    }
  }]
}]';
```

To update the previous example to include uppercase and lowercase characters, replace `[a-z]` with `[A-Za-z]`.

The following example specifies all files in the `data_files` folder that have a numerical suffix that end in `0` with one preceding digit. For example, it loads the files `data_file_10.csv`, `data_file_20.csv`, and `data_file_30.csv` to load the external table.

```
mysql> SET @input_list = '[{
  "db_name": "lakehouse_db",
  "tables": [{
    "table_name": "table_1",
    "engine_attribute": {
      "dialect": {"format": "csv"},
      "file": [{"par": "https://objectstorage.us-ashburn-1.oraclecloud.com/p/.../n/tenancy_1/b/bucket_1/o/'
```

```
                "pattern": "data_files/data_file_\\\\d0\\\\.csv"}]
    }
  }]
}]';
```

To update the previous example to load files with one or more digits preceding the 0, update `\\\\d0\\\\` with `\\\\d+0\\\\`.

What's Next

After successfully creating external tables and specifying the files to load data into the table, learn how to Load Structured Data Using Auto Parallel Load.

4.3.8.5 Use Lakehouse Auto Parallel Load with external_tables Option

Note

Do not use the `external_tables` option as of MySQL 8.4.0. This option will be deprecated in a future release. Use the `input_list` JSON array instead. See HEATWAVE_LOAD and full descriptions for `input_list`.

The Auto Parallel Load `external_tables` option is a JSON array that includes one or more `db_object` items. It allows you to set the details for schemas and tables to load.

Syntax Examples for external_tables Option

While it is possible to define the entire load command on a single line, for readability the configuration is divided into option definitions using SET.

Define the following option sets:

- Define the name of the database which will store the data.

```
mysql> SET @db_list = '["tpch"]';
```

  This assumes that Lakehouse Auto Parallel Load will analyze the data, infer the table structure, and create the database and all tables.

- Define the `db_object` parameters that will load data from three external sources with Avro, CSV and Parquet format files:

```
mysql> SET @ext_tables = '[{
  "db_name": "tpch",
  "tables": [{
    "table_name": "supplier_pq",
    "dialect": {"format": "parquet"},
    "file": [{
      "prefix": "src_data/parquet/tpch/supplier/",
      "bucket": "myBucket",
      "namespace": "myNamespace",
      "region": "myRegion"}]
  },
  {
    "table_name": "nation_csv",
    "dialect": {
      "format": "csv",
      "field_delimiter": "|",
      "record_delimiter": "|\\n",
      "has_header": true
    },
    "file": [{
      "par": "https://objectstorage.../nation.csv"}]
  },
  {
    "table_name": "region_avro",
    "dialect": {"format": "avro"},
```

```
    "file": [{
      "par": "https://objectstorage.../region.avro"}]
  }]
}]';
```

- Define the `@options` variable with `SET`:

```
mysql> SET @options = JSON_OBJECT('external_tables', CAST(@ext_tables AS JSON));
```

Setting `mode` to `dryrun` generates the load script but does not create or load the external tables. For example:

```
mysql> SET @options = JSON_OBJECT('mode', 'dryrun', 'external_tables', CAST(@ext_tables AS JSON));
```

To implement the changes as part of the load command, set `mode` to `normal`. This is the default, and it is not necessary to add it to the command.

Exclude columns from the loading process with the `exclude_list` option. See HEATWAVE_LOAD.

Lakehouse Auto Parallel Load infers the column names for Avro and Parquet files, and also for CSV files if `has_header` is `true`. For these situations, use the column names with the `exclude_list` option.

If the table already exists, but has no data, use the existing column names with the `exclude_list` option.

For CSV files if `has_header` is `false`, use the generated schema names with the `exclude_list` option. These are: `col_1`, `col_2`, `col_3` ...

4.3.9 Create External Tables Manually

The topics in this section describe how to create external tables manually.

4.3.9.1 Create an External Table Manually

If you choose not to Create an External Table Using Lakehouse Auto Parallel Load, you can create the table manually.

This topic contains the following sections:

- Before You Begin
- Create External Table Manually Example
- What's Next

Before You Begin

- Prepare to create external tables by reviewing the following:
  - Additional MySQL HeatWave Lakehouse Requirements
  - MySQL HeatWave Lakehouse Privileges
  - Supported File Format
  - Access Object Storage
- Create the external tables that will ingest the loaded data by either using SQL syntax or JSON syntax.
  - CREATE TABLE Statement for SQL syntax. Also review relevant examples showing how to create external tables.

- CREATE TABLE Statement for JSON syntax. Also review relevant examples showing how to create external tables.

Create External Table Manually Example

To demonstrate how to create an external table manually, the following example specifies a single file for the table by using an OCIFS URI.

The CSV file in this example is from Bank Marketing. To use this file, visit Bank Marketing and download the `bank+marketing.zip` file. Unzip the file, and then unzip the `bank.zip` file. Refer to the `bank.csv` file.

1. Prepare the files to load in the proper format. See Supported File Formats.
2. Upload the files to load into Object Storage. See Uploading an Object Storage Object to a Bucket in *Oracle Cloud Infrastructure Documentation*.
3. Select the method to specify the files for the external table: uniform resource identifier (URI), resource principal, or PAR. To learn more about each method, see the following:
   - Access Object Storage with Uniform Resource Identifiers and URI load examples.
   - Access Object Storage with Resource Principals and resource principal load examples.
   - Access Object Storage with Pre-Authenticated Requests and PAR load examples.
4. Connect to your MySQL HeatWave Database System.
5. In the terminal window, create and use the database to store the table.

```
mysql> CREATE DATABASE bank_data;
mysql> USE DATABASE bank_data;
```

6. Set up the `CREATE EXTERNAL TABLE` statement and the `ENGINE_ATTRIBUTE` options to specify the parameters needed to process the external files. In versions earlier than MySQL 9.4.0, you must use the `CREATE TABLE` statement and set `ENGINE` to `lakehouse`, and `SECONDARY_ENGINE` to `rapid`.

   Ensure that the table has the correct data type for each column. For this example, columns are defined according to the data in the `bank.csv` file. See the following to learn more:

   - CREATE TABLE
   - Data Types
   - Lakehouse External Table Syntax

   As of MySQL 9.4.0, you can use SQL syntax to set options for external tables. For earlier versions, you must set options using JSON syntax. To learn more, see Lakehouse External Table JSON Syntax and Lakehouse External Table SQL Syntax.

   The following example uses SQL syntax:

```
mysql> CREATE EXTERNAL TABLE bank_marketing(
        age int,
        job varchar(255),
        marital varchar(255),
        education varchar(255),
        default1 varchar(255),
        balance float,
        housing varchar(255),
        loan varchar(255),
        contact varchar(255),
        day int,
```

```
        month varchar(255),
        duration float,
        campaign int,
        pdays float,
        previous float,
        poutcome varchar(255),
        y varchar(255)
    )
        FILE_FORMAT = (FORMAT csv HEADER ON)
        FILES = (URI = 'oci://mybucket@mynamespace/bucket_1/bank.csv');
```

If you are on MySQL 9.1.2 and earlier, you need to update `dialect` with the `field delimiter` and `record delimiter` parameters. As of MySQL 9.2.0, MySQL HeatWave Lakehouse can automatically detect these values. See Lakehouse External Table Syntax to learn more.

Where:

- The `CREATE EXTERNAL TABLE` statement creates the table `bank_marketing` and automatically sets `ENGINE` to `lakehouse`, and `SECONDARY_ENGINE` to `rapid`.
- Each column for the table is defined according to the `bank.csv` file.
- `FORMAT` defines the format of the external file: `csv`.
- `HEADER` identifies a header in the external file.
- `URI` sets the URI link to access the file. Replace the link in the example with your own.

If you are on a version earlier than MySQL 9.4.0, you must use the following JSON syntax:

```
mysql> CREATE EXTERNAL TABLE bank_marketing(
        age int,
        job varchar(255),
        marital varchar(255),
        education varchar(255),
        default1 varchar(255),
        balance float,
        housing varchar(255),
        loan varchar(255),
        contact varchar(255),
        day int,
        month varchar(255),
        duration float,
        campaign int,
        pdays float,
        previous float,
        poutcome varchar(255),
        y varchar(255)
    )
        ENGINE_ATTRIBUTE='{"dialect": {"format": "csv", "has_header": true},
                          "file": [{ "uri": "oci://mybucket@mynamespace/bucket_1/bank.csv"}]}';
```

7. After creating the external table, do the following:

   - Before loading the external table with data from the specified files, you have the option to validate the load operation (as of MySQL 9.6.0) by using the `SECONDARY_LOAD` and `VALIDATE ONLY` clauses in an `ALTER TABLE` statement. Review any error and warning messages before proceeding.
   - When you are ready to load the external table, use the `SECONDARY_LOAD` clause in an `ALTER TABLE` statement.
   - To learn more, see Load Structured Data Manually.

What's Next

- Review different ways to specify files when creating external tables manually:

- Using Uniform Resource Identifiers
- Using Resource Principals
- Using Pre-Authenticated Requests

- Review how to Load Structured Data Manually.

4.3.9.2 Create Tables with Metadata Columns

As of MySQL 9.6.1, MySQL HeatWave Lakehouse supports the `_metadata_filename` column, which is a reserved column name that stores file name information for each row of data.

Before You Begin

Review how to Create an External Table Manually.

Overview

- You can use metadata columns for all supported MySQL HeatWave Lakehouse file formats for structured and semi-structured data. See Supported File Formats for Structured and Semi-Structured Data. For unstructured data, the `document_name` column serves as the metadata filename column. See Section 4.3.4, “Supported File Formats and Data Types”.
- The `_metadata_filename` metadata column is treated as a normal table column and must be part of the table definition.
- Specifying an invalid data type for a metadata column generates an error if `STRICT_LOAD` or `is_strict_mode` is enabled, or a warning message if `STRICT_LOAD` or `is_strict_mode` is disabled. We recommend using the data type `VARCHAR(1024)`.
- Guided Load is disabled for tables with metadata columns.
- Specifying a generated column as a metadata column generates an error.
- When you use the `match_columns_by` option and specify the `_metadata_filename` column in the schema, the specified files must not contain a `_metadata_filename` column, as it conflicts with the metadata column. In that case the file column loads and not the metadata column.

For example, you create a table that includes the `_metadata_filename` column:

```
mysql> CREATE EXTERNAL TABLE table_1 (
 col_1 int,
 col_2 int,
 _metadata_filename varchar(1024)
)
FILE_FORMAT = (FORMAT csv)
FILES = (URI = 'oci://mybucket@mynamespace/data_files/data_file_[1-3].csv');
```

The command loads `data_file_1`, `data_file_2`, and `data_file_3` into the external table.

The files load the following values:

- `data_file_1`: 1, 2
- `data_file_2`: 2, 3
- `data_file_3`: 3, 4

The output of the table looks like this:

```
mysql> SELECT * FROM table_1;
```

```
+-------+-------+-----------------------------------------------------------------------------------------
| col_1 | col_2 | _metadata_filename
+-------+-------+-----------------------------------------------------------------------------------------
| 1     | 2     | https://objectstorage.us-ashburn-1.oraclecloud.com/n/mynamespace/b/mybucket/o/data_file_1
| 2     | 3     | https://objectstorage.us-ashburn-1.oraclecloud.com/n/mynamespace/b/mybucket/o/data_file_2
| 3     | 4     | https://objectstorage.us-ashburn-1.oraclecloud.com/n/mynamespace/b/mybucket/o/data_file_3
+-------+-------+-----------------------------------------------------------------------------------------
```

The table shows the file name associated with each row of data.

What's Next

- Review different ways to specify files when creating external tables manually:
  - Using Uniform Resource Identifiers
  - Using Resource Principals
  - Using Pre-Authenticated Requests
- Review how to Load Structured Data Manually.

4.3.9.3 Create Tables with Key Partitioning

As of MySQL 9.6.1, you can specify the `PARTITION BY KEY` option in the `CREATE EXTERNAL TABLE` statement.

Learn more about KEY Partitioning from the MySQL Reference Manual.

Before You Begin

Review how to Create an External Table Manually.

Overview

- For structured and semi-structured data, the column specified for key partitioning must be a metadata column, or a generated column over a metadata column. For unstructured data, the column specified must be the `document_id` column. See Metadata Columns.
- You can use key partitioning for all supported MySQL HeatWave Lakehouse file formats. See Supported File Formats for Structured and Semi-Structured Data and Supported File Formats for Unstructured Data.
- The partitioning type must be `KEY`, otherwise an error is generated.
- Once MySQL HeatWave Lakehouse loads a table, you cannot modify the partitioning.

Examples with SQL Syntax

The following example specifies key partitioning. The required metadata column `_metadata_filename` is specified.

```
mysql> CREATE EXTERNAL TABLE table_1 (
 col_1 int,
 col_2 int,
 _metadata_filename varchar(1024)
)
FILE_FORMAT = (FORMAT csv)
FILES = (URI = 'oci://mybucket@mynamespace/data_files/data_file_[1-3].csv')
PARTITION BY KEY (_metadata_filename);
```

The following example uses a generated column over the metadata column. It extracts only the name of the files from the specified URI.

```
mysql> CREATE EXTERNAL TABLE table_1 (
 col_1 int,
 col_2 int,
 _metadata_filename varchar(1024),
 partition_col VARCHAR(128) generated always as (substring_index(_metadata_filename, '/', -1)) stored
)
FILE_FORMAT = (FORMAT csv)
FILES = (URI = 'oci://mybucket@mynamespace/data_files/data_file_[1-3].csv')
PARTITION BY KEY (partition_col);
```

To learn more, see Create Tables with Stored Generated Columns.

Examples with JSON Syntax

The following example specifies key partitioning. The required metadata column `_metadata_filename` is specified.

```
mysql> CREATE EXTERNAL TABLE table_1 (
 col_1 int,
 col_2 int,
 _metadata_filename varchar(1024)
)
ENGINE_ATTRIBUTE='{"dialect": {"format": "csv"},
"file": [{"uri": "oci://mybucket@mynamespace/data_files/data_file_[1-3].csv"}]}'
PARTITION BY KEY (_metadata_filename);
```

The following example uses a generated column over the metadata column. It extracts only the name of the files from the specified URI.

```
mysql> CREATE EXTERNAL TABLE table_1 (
 col_1 int,
 col_2 int,
 _metadata_filename varchar(1024),
 partition_col VARCHAR(128) generated always as (substring_index(_metadata_filename, '/', -1)) stored
)
ENGINE_ATTRIBUTE='{"dialect": {"format": "csv"},
"file": [{"uri": "oci://mybucket@mynamespace/data_files/data_file_[1-3].csv"}]}'
PARTITION BY KEY (partition_col);
```

What's Next

- Review different ways to specify files when creating external tables manually:
  - Using Uniform Resource Identifiers
  - Using Resource Principals
  - Using Pre-Authenticated Requests
- Review how to Load Structured Data Manually.

4.3.9.4 Use URI to Create External Tables Manually

As of MySQL 9.3.1, and only on MySQL HeatWave on OCI, you can use uniform resource identifiers (URI) to specify files for loading data when creating external tables manually. Review the following examples for different ways to create external tables.

This topic contains the following sections:

- Before You Begin
- Types of URIs
- URI Parameters
- OCIFS URI Examples

- PAR URI Examples
- Native URI Examples
- What's Next

Before You Begin

- To use URIs, you require the same privileges to use pre-authenticated requests and resource prinicipals.
  - To use PAR URIs, make sure you have the required PAR Privileges.
  - To use Native URIs and OCIFS URIs, make sure you have the required Resource Principal Privileges.
- Create the external tables that will ingest the loaded data by either using SQL syntax or JSON syntax.
  - CREATE TABLE Statement for SQL syntax. Also review relevant examples showing how to create external tables.
  - CREATE TABLE Statement for JSON syntax. Also review relevant examples showing how to create external tables.
- Review how to Create an External Table Manually.

Types of URIs

There are three types of URIs supported: OCIFS, pre-authenticated request (PAR), and Native URI, and you can specify a name of a file as your source in Object Storage, or a prefix, or a pattern.

When you define the `object_path` of a URI, the name, prefix, or glob pattern is inferred by the following.

- Name: If the `object_path` is neither a glob pattern nor prefix.
- Prefix: If the `object_path` is not a glob pattern and ends with an unencoded `/` character, such as a folder path.
- Glob pattern: If the `object_path` contains at least one of the following unencoded characters: `?`, `*`, or `[`. To use these characters as literals, you need to escape them or encode them as needed depending on the URI syntax. Regex patterns are not supported. See Glob Patterns from the *Oracle Cloud Infrastructure Documentation* to learn more.

URI Parameters

Depending on the type of URI you use, you need to ensure that the following parameters are present in the URI.

- `bucket_name`: Buckets allow you to store objects in a compartment. This parameter is needed for all three types of URIs. To learn more about buckets, see Object Storage Buckets in *Oracle Cloud Infrastructure Documentation*.
- `namespace_name`: This is the top-level container for all buckets and objects. This parameter is needed for all three types of URIs. To learn how to view your namespace, see Object Storage Namespaces in *Oracle Cloud Infrastructure Documentation*.
- `region_name`: The region identifier for your OCI tenancy. This parameter is needed for PAR URI and Native URI. For OCIFS URI, it is set to the region of the instance running Lakehouse. To view the list of available regions and region identifiers, see Regions and Availability Domains in *Oracle Cloud Infrastructure Documentation*.

Note

If you are on MySQL 9.1.2 and earlier, you need to update `dialect` with the `field delimiter` and `record delimiter` parameters. As of MySQL 9.2.0, MySQL HeatWave Lakehouse can automatically detect these values. See Lakehouse External Table Syntax to learn more.

To review all syntax options for loading external files, see Lakehouse External Table Syntax.

OCIFS URI Examples

The format of an OCIFS URI is the following:

```
oci://bucket_name@namespace_name/object_path
```

The `object_path` in an OCIFS URI cannot be empty. It also does not have a region parameter. The instance's associated region is used as the default region. See OCIFS Documentation to learn more about OCIFS and its specifications.

The following examples specify a single file as a name since the `object_path` is not a glob pattern or prefix. The example uses the `data_file_1.csv` file to load the external table.

SQL syntax example:

```
mysql> CREATE EXTERNAL TABLE table_1(col_1 int, col_2 int, col_3 int)
       FILE_FORMAT = (FORMAT csv)
       FILES = (URI = 'oci://mybucket@mynamespace/data_files/data_file_1.csv');
```

JSON syntax example:

```
mysql> CREATE EXTERNAL TABLE table_1(col_1 int, col_2 int, col_3 int)
       ENGINE_ATTRIBUTE='{"dialect": {"format": "csv"},
                         "file": [{"uri": "oci://mybucket@mynamespace/data_files/data_file_1.csv"}]}'
```

In the following examples the `object_path` is encoded, so it is treated as a name instead of a prefix. The original file name is `data_file_[1].csv`.

SQL syntax example:

```
mysql> CREATE EXTERNAL TABLE table_1(col_1 int, col_2 int, col_3 int)
       FILE_FORMAT = (FORMAT csv)
       FILES = (URI = 'oci://mybucket@mynamespace/data_files/data_file_%5B1%5D.csv');
```

JSON syntax example:

```
mysql> CREATE EXTERNAL TABLE table_1(col_1 int, col_2 int, col_3 int)
       ENGINE_ATTRIBUTE='{"dialect": {"format": "csv"},
                         "file": [{"uri": "oci://mybucket@mynamespace/data_files/data_file_%5B1%5D.cs
```

The following examples specify files as a prefix because the `object_path` ends with a `/` character and does not contain any of the following characters: `*`, `?`, or `[`. The examples use all files in the `data_files/` Object Storage folder.

SQL syntax example:

```
mysql> CREATE EXTERNAL TABLE table_1(col_1 int, col_2 int, col_3 int)
       FILE_FORMAT = (FORMAT csv)
       FILES = (URI = 'oci://mybucket@mynamespace/data_files/');
```

JSON syntax example:

```
mysql> CREATE EXTERNAL TABLE table_1(col_1 int, col_2 int, col_3 int)
       ENGINE_ATTRIBUTE='{"dialect": {"format": "csv"},
```

```
                          "file": [{"uri": "oci://mybucket@mynamespace/data_files/"}]}';
```

The following examples use a glob pattern and specify two files, `data_file_1.csv` and `data_file_2.csv`, to load the external table. The `object_path` is a pattern because it has a `[` character.

SQL syntax example:

```
mysql> CREATE EXTERNAL TABLE table_1(col_1 int, col_2 int, col_3 int)
       FILE_FORMAT = (FORMAT csv)
       FILES = (URI = 'oci://mybucket@mynamespace/data_files/data_file_[1-2].csv');
```

JSON syntax example:

```
mysql> CREATE EXTERNAL TABLE table_1(col_1 int, col_2 int, col_3 int)
       ENGINE_ATTRIBUTE='{"dialect": {"format": "csv"},
                         "file": [{"uri": "oci://mybucket@mynamespace/data_files/data_file_[1-2].csv"}]}'
```

The following examples use a glob pattern because they have an unencoded `*` character. The examples specify all CSV files that start with `data_file_` to load the external table. For example, they use the files `data_file_1.csv`, `data_file_2.csv`, and `data_file_3.csv`.

SQL syntax example:

```
mysql> CREATE EXTERNAL TABLE table_1(col_1 int, col_2 int, col_3 int)
       FILE_FORMAT = (FORMAT csv)
       FILES = (URI = 'oci://mybucket@mynamespace/data_files/data_file_*.csv');
```

JSON syntax example:

```
mysql> CREATE EXTERNAL TABLE table_1(col_1 int, col_2 int, col_3 int)
       ENGINE_ATTRIBUTE='{"dialect": {"format": "csv"},
                         "file": [{"uri": "oci://mybucket@mynamespace/data_files/data_file_*.csv"}]}';
```

The following examples use a glob pattern because they have an unencoded `?` character. The examples specify all CSV files that start with `data_file_` and have any single character followed by `0` to load the external table. For example, they use the files `data_file_10.csv`, `data_file_20.csv`, and `data_file_30.csv`.

SQL syntax example:

```
mysql> CREATE EXTERNAL TABLE table_1(col_1 int, col_2 int, col_3 int)
       FILE_FORMAT = (FORMAT csv)
       FILES = (URI = 'oci://mybucket@mynamespace/data_files/data_file_?0.csv');
```

JSON syntax example:

```
mysql> CREATE EXTERNAL TABLE table_1(col_1 int, col_2 int, col_3 int)
       ENGINE_ATTRIBUTE='{"dialect": {"format": "csv"},
                         "file": [{"uri": "oci://mybucket@mynamespace/data_files/data_file_?0.csv"}]}';
```

PAR URI Examples

The format of a PAR URI is the following if using dedicated endpoints:

```
https://namespace_name.objectstorage.region_name.oci.customer-oci.com/p/PAR_token/n/namespace_name/b/bucket
```

The format of a PAR URI can also be the following:

```
https://objectstorage.region_name.oraclecloud.com/p/PAR_token/n/namespace_name/b/bucket_name/o/object_path
```

The `object_path` in a PAR URI can be empty. If the `object_path` is empty, the PAR URI is treated as a PAR with no additional pattern, prefix, or name target.

For more information about pre-authenticated requests, see Object Storage Pre-Authenticated Requests in *Oracle Cloud Infrastructure Documentation*.

The following examples specify a single file as a name since the `object_path` is not a glob pattern or prefix. The examples use the file `data_file_2.csv`.

SQL syntax example:

```
mysql> CREATE EXTERNAL TABLE table_1(col_1 int, col_2 int, col_3 int)
       FILE_FORMAT = (FORMAT csv)
       FILES = (URI = 'https://mynamespace.objectstorage.myregion.oci.customer-oci.com/p/.../n/mynames
```

JSON syntax example:

```
mysql> CREATE EXTERNAL TABLE table_1(col_1 int, col_2 int, col_3 int)
       ENGINE_ATTRIBUTE='{"dialect": {"format": "csv"},
                         "file": [{"uri": "https://mynamespace.objectstorage.myregion.oci.customer-oc
```

In the following examples the `object_path` is encoded, so it is treated as a name instead of a prefix. The original file name is `data_file_[1].csv`.

SQL syntax example:

```
mysql> CREATE EXTERNAL TABLE table_1(col_1 int, col_2 int, col_3 int)
       FILE_FORMAT = (FORMAT csv)
       FILES = (URI = 'https://mynamespace.objectstorage.myregion.oci.customer-oci.com/p/.../n/mynames
```

JSON syntax example:

```
mysql> CREATE EXTERNAL TABLE table_1(col_1 int, col_2 int, col_3 int)
       ENGINE_ATTRIBUTE='{"dialect": {"format": "csv"},
                         "file": [{"uri": "https://mynamespace.objectstorage.myregion.oci.customer-oc
```

In the following examples, the `object_path` is empty, which means that no additional target is specified. Therefore, the examples load all the files under the PAR.

SQL syntax example:

```
mysql> CREATE EXTERNAL TABLE table_1(col_1 int, col_2 int, col_3 int)
       FILE_FORMAT = (FORMAT csv)
       FILES = (URI = 'https://mynamespace.objectstorage.myregion.oci.customer-oci.com/p/.../n/mynames
```

JSON syntax example:

```
mysql> CREATE EXTERNAL TABLE table_1(col_1 int, col_2 int, col_3 int)
       ENGINE_ATTRIBUTE='{"dialect": {"format": "csv"},
                         "file": [{"uri": "https://mynamespace.objectstorage.myregion.oci.customer-oc
```

The following examples specify files as a prefix because the `object_path` ends with a `/` character and does not contain any of the following characters: `*`, `?`, or `[`. The examples use any files in the `data_files/` Object Storage folder to load the external table.

SQL syntax example:

```
mysql> CREATE EXTERNAL TABLE table_1(col_1 int, col_2 int, col_3 int)
       FILE_FORMAT = (FORMAT csv)
       FILES = (URI = 'https://mynamespace.objectstorage.myregion.oci.customer-oci.com/p/.../n/mynames
```

JSON syntax example:

```
mysql> CREATE EXTERNAL TABLE table_1(col_1 int, col_2 int, col_3 int)
       ENGINE_ATTRIBUTE='{"dialect": {"format": "csv"},
                         "file": [{"uri": "https://mynamespace.objectstorage.myregion.oci.customer-oc
```

The following examples use a glob pattern and load two files, `data_file_2` and `data_file_3.csv`, to load the external table. The `object_path` is a pattern because it has a `[` character.

SQL syntax example:

```
mysql> CREATE EXTERNAL TABLE table_1(col_1 int, col_2 int, col_3 int)
       FILE_FORMAT = (FORMAT csv)
       FILES = (URI = 'https://mynamespace.objectstorage.myregion.oci.customer-oci.com/p/.../n/mynamespace
```

JSON syntax example:

```
mysql> CREATE EXTERNAL TABLE table_1(col_1 int, col_2 int, col_3 int)
       ENGINE_ATTRIBUTE='{"dialect": {"format": "csv"},
                         "file": [{"uri": "https://mynamespace.objectstorage.myregion.oci.customer-oci.co
```

The following examples use a glob pattern because they have an unencoded `*` character. The examples specify all CSV files that start with `data_file_` to load the external table. For example, they use the files `data_file_1.csv`, `data_file_2.csv`, and `data_file_3.csv`.

SQL syntax example:

```
mysql> CREATE EXTERNAL TABLE table_1(col_1 int, col_2 int, col_3 int)
       FILE_FORMAT = (FORMAT csv)
       FILES = (URI = 'https://mynamespace.objectstorage.myregion.oci.customer-oci.com/p/.../n/mynamespace
```

JSON syntax example:

```
mysql> CREATE EXTERNAL TABLE table_1(col_1 int, col_2 int, col_3 int)
       ENGINE_ATTRIBUTE='{"dialect": {"format": "csv"},
                         "file": [{"uri": "https://mynamespace.objectstorage.myregion.oci.customer-oci.co
```

The following examples use a glob pattern because they have an unencoded `?` character. The examples specify all CSV files that start with `data_file_` and have any single character followed by `0` to load the external table. For example, they use the files `data_file_10.csv`, `data_file_20.csv`, and `data_file_30.csv`.

SQL syntax example:

```
mysql> CREATE EXTERNAL TABLE table_1(col_1 int, col_2 int, col_3 int)
       FILE_FORMAT = (FORMAT csv)
       FILES = (URI = 'https://mynamespace.objectstorage.myregion.oci.customer-oci.com/p/.../n/mynamespace
```

JSON syntax example:

```
mysql> CREATE EXTERNAL TABLE table_1(col_1 int, col_2 int, col_3 int)
       ENGINE_ATTRIBUTE='{"dialect": {"format": "csv"},
                         "file": [{"uri": "https://mynamespace.objectstorage.myregion.oci.customer-oci.co
```

Native URI Examples

The format of a Native URI is the following if using dedicated endpoints:

```
https://namespace_name.objectstorage.region_name.oci.customer-oci.com/n/namespace_name/b/bucket_name/o/obje
```

The format of a Native URI can also be the following:

```
https://objectstorage.region_name.oraclecloud.com/n/namespace_name/b/bucket_name/o/object_path
```

The `object_path` cannot be empty.

The following examples specify a single file as a name since the `object_path` is not a glob pattern or prefix. The examples use the `data_file_1.csv` file.

SQL syntax example:

```
mysql> CREATE EXTERNAL TABLE table_1(col_1 int, col_2 int, col_3 int)
```

```
        FILE_FORMAT = (FORMAT csv)
        FILES = (URI = 'https://mynamespace.objectstorage.myregion.oci.customer-oci.com/n/mynamespace/b
```

JSON syntax example:

```
mysql> CREATE EXTERNAL TABLE table_1(col_1 int, col_2 int, col_3 int)
        ENGINE_ATTRIBUTE='{"dialect": {"format": "csv"},
                          "file": [{"uri": "https://mynamespace.objectstorage.myregion.oci.customer-oc
```

In the following examples the `object_path` is encoded, so it is treated as a name instead of a prefix. The original file name is `data_file_[1].csv`.

SQL syntax example:

```
mysql> CREATE EXTERNAL TABLE table_1(col_1 int, col_2 int, col_3 int)
        FILE_FORMAT = (FORMAT csv)
        FILES = (URI = 'https://mynamespace.objectstorage.myregion.oci.customer-oci.com/n/mynamespace/b
```

JSON syntax example:

```
mysql> CREATE EXTERNAL TABLE table_1(col_1 int, col_2 int, col_3 int)
        ENGINE_ATTRIBUTE='{"dialect": {"format": "csv"},
                          "file": [{"uri": "https://mynamespace.objectstorage.myregion.oci.customer-oc
```

The following examples specify files as a prefix because the `object_path` ends with a `/` character and does not contain any of the following characters: `*`, `?`, or `[`. The examples use any files in the `data_files/` Object Storage folder to load the external table.

SQL syntax example:

```
mysql> CREATE EXTERNAL TABLE table_1(col_1 int, col_2 int, col_3 int)
        FILE_FORMAT = (FORMAT csv)
        FILES = (URI = 'https://mynamespace.objectstorage.myregion.oci.customer-oci.com/n/mynamespace/b
```

JSON syntax example:

```
mysql> CREATE EXTERNAL TABLE table_1(col_1 int, col_2 int, col_3 int)
        ENGINE_ATTRIBUTE='{"dialect": {"format": "csv"},
                          "file": [{"uri": "https://mynamespace.objectstorage.myregion.oci.customer-oc
```

The following examples use a glob pattern and specify two files, `data_file_1.csv` and `data_file_2.csv`, to load the external table. The `object_path` is a pattern because it has a `[` character.

SQL syntax example:

```
mysql> CREATE EXTERNAL TABLE table_1(col_1 int, col_2 int, col_3 int)
        FILE_FORMAT = (FORMAT csv)
        FILES = (URI = 'https://mynamespace.objectstorage.myregion.oci.customer-oci.com/n/mynamespace/b
```

JSON syntax example:

```
mysql> CREATE EXTERNAL TABLE table_1(col_1 int, col_2 int, col_3 int)
        ENGINE_ATTRIBUTE='{"dialect": {"format": "csv"},
                          "file": [{"uri": "https://mynamespace.objectstorage.myregion.oci.customer-oc
```

The following examples use a glob pattern because they have an unencoded `*` character. The examples specify all CSV files that start with `data_file_` to load the external table. For example, they use the files `data_file_1.csv`, `data_file_2.csv`, and `data_file_3.csv`.

SQL syntax example:

```
mysql> CREATE EXTERNAL TABLE table_1(col_1 int, col_2 int, col_3 int)
        FILE_FORMAT = (FORMAT csv)
```

```
        FILES = (URI = 'https://mynamespace.objectstorage.myregion.oci.customer-oci.com/n/mynamespace/b/myb
```

JSON syntax example:

```
mysql> CREATE EXTERNAL TABLE table_1(col_1 int, col_2 int, col_3 int)
        ENGINE_ATTRIBUTE='{"dialect": {"format": "csv"},
                          "file": [{"uri": "https://mynamespace.objectstorage.myregion.oci.customer-oci.co
```

The following examples use a glob pattern because they have an unencoded `?` character. The examples specify all CSV files that start with `data_file_` and have any single character followed by `0` to load the external table. For example, they use the files `data_file_10.csv`, `data_file_20.csv`, and `data_file_30.csv`.

SQL syntax example:

```
mysql> CREATE EXTERNAL TABLE table_1(col_1 int, col_2 int, col_3 int)
        FILE_FORMAT = (FORMAT csv)
        FILES = (URI = 'https://mynamespace.objectstorage.myregion.oci.customer-oci.com/n/mynamespace/b/myb
```

JSON syntax example:

```
mysql> CREATE EXTERNAL TABLE table_1(col_1 int, col_2 int, col_3 int)
        ENGINE_ATTRIBUTE='{"dialect": {"format": "csv"},
                          "file": [{"uri": "https://mynamespace.objectstorage.myregion.oci.customer-oci.co
```

What's Next

After successfully creating external tables manually and specifying the files to load data into the table, learn how to Load Structured Data Manually.

4.3.9.5 Use Resource Principals to Create External Tables Manually

Review the following examples to see the different ways to use resource principals to specify files when creating external tables manually.

This topic contains the following sections:

- Before You Begin
- Resource Principals Parameters
- Resource Principals Examples
- Specify a Single File
- Specify Multiple Files for One External Table
- Specify Files with a Pattern
- What's Next

Before You Begin

- Review the requirements to Access Object Storage with Resource Principals.
- Review how to Create an External Table Manually.

Resource Principals Parameters

To set up resource principals, you configure the following parameters:

- `bucket_name`: Buckets allow you to storage objects in a compartment. To learn more about buckets, see Object Storage Buckets in *Oracle Cloud Infrastructure Documentation*.

- `namespace_name`: This is the top-level container for all buckets and objects. To learn how to view your namespace, see Object Storage Namespaces in *Oracle Cloud Infrastructure Documentation*.
- `region_name`: You need to define the region identifier for your OCI tenancy. To view the list of available regions and region identifiers, see Regions and Availability Domains in *Oracle Cloud Infrastructure Documentation*.

  As of MySQL 9.2.1, if the region is not specified, the instance's associated region is used as the default region. Otherwise, the specified region is used.
- `name`: Use this to specify an Object Storage file name.
- `pattern`: Use this to set a regular expression that defines a set of Object Storage files. The pattern follows the modified Modified ECMAScript regular expression grammar.
- `prefix`: Use this to define a set of Object Storage folders and files.

Resource Principals Examples

The following examples use these parameters to create the external tables and specify the files to load the tables with. Replace the values in the examples with your own.

- `dialect` defines the format options of the specified files.
- The `file` parameters define the resource principal details.
  - `region`: The region the tenant resides in.
  - `namespace`: The name of the tenancy.
  - `bucket`: The name of the Object Storage bucket.
  - `name`: The name of the external file to load.
  - `prefix`: The prefix for a set of Object Storage folders and files.
  - `pattern:` The regular expression that defines a set of Object Storage files.

To review all syntax options for loading external files, see Lakehouse External Table Syntax.

Specify a Single File

The following example specifies a single file.

```
mysql> CREATE EXTERNAL TABLE table_1(col_1 int, col_2 int, col_3 int)
       ENGINE_ATTRIBUTE='{"dialect": {"format": "csv"},
                          "file": [{"region": "regionName",
                                    "namespace": "tenant_1",
                                    "bucket": "bucket_1",
                                    "name": "data_file_1.csv"}]}';
```

Specify Multiple Files for One External Table

To specify multiple external files for one external table, you can do one of the following:

- Create a folder in the Object Storage bucket and upload the required files into that folder. See Managing Folders in an Object Storage Bucket in *Oracle Cloud Infrastructure Documentation*.
- Create `file` items as needed to specify the files to load into the table.

The following example specifies all the files uploaded to a folder in the Object Storage bucket for one external table.

```
mysql> CREATE EXTERNAL TABLE table_1(col_1 int, col_2 int, col_3 int)
       ENGINE_ATTRIBUTE='{"dialect": {"format": "csv"},
                          "file": [{"region": "regionName",
                                    "namespace": "tenant_1",
                                    "bucket": "bucket_1",
                                    "prefix": "data_files/"}]}';
```

The following example specifies two files for one external table.

```
mysql> CREATE EXTERNAL TABLE table_1(col_1 int, col_2 int, col_3 int)
       ENGINE_ATTRIBUTE='{"dialect": {"format": "csv"},
                          "file": [{"region": "regionName,
                                    "namespace": "tenant_1",
                                    "bucket": "bucket_1",
                                    "name": "data_files/data_file_1.csv"},
                                   {"region": "regionName",
                                    "namespace": "tenant_1",
                                    "bucket": "bucket_1",
                                    "name": "data_files/data_file_2.csv"}]}';
```

Specify Files with a Pattern

You can add a `pattern` parameter to use regular expression to specify files.

The regular expression syntax requires certain characters to have an escape character.

The escape character is the backslash character, and it is a reserved character in both JSON and MySQL. Therefore, it is necessary to escape the backslash character twice, and specify `\\` for both JSON and MySQL.

However, the regular expression escape sequence depends upon the `NO_BACKSLASH_ESCAPES` SQL mode:

- Use `\\.` to escape a period if `NO_BACKSLASH_ESCAPES` is enabled.
- Use `\\\\.` to escape a period if `NO_BACKSLASH_ESCAPES` is not enabled. The following examples use this sequence because it is the default mode.

See the following to learn more:

- Regular Expression Syntax
- Modified ECMAScript regular expression grammar

The following example specifies all files in the `data_files` folder that have a numerical suffix of one or more digits to load into the external table. For example, it uses the files `data_file_1.csv`, `data_file_2.csv`, and `data_file_3.csv`.

```
mysql> CREATE EXTERNAL TABLE table_1(col_1 int, col_2 int, col_3 int)
       ENGINE_ATTRIBUTE='{"dialect": {"format": "csv"},
                          "file": [{"region": "regionName",
                                    "namespace": "tenant_1",
                                    "bucket": "bucket_1",
                                    "pattern": "data_files/data_file_\\\\d+\\\\.csv"}]}';
```

The following examples specifies all files in the `data_files` folder that have an alphabetical suffix of one or more lowercase characters to load into the external table. For example, it uses the files `data_file_a.csv`, `data_file_b.csv`, and `data_file_c.csv`.

```
mysql> CREATE EXTERNAL TABLE table_1(col_1 int, col_2 int, col_3 int)
       ENGINE_ATTRIBUTE='{"dialect": {"format": "csv"},
                          "file": [{"region": "regionName",
                                    "namespace": "tenant_1",
                                    "bucket": "bucket_1",
                                    "pattern": "data_files/data_file_[a-z]+\\\\.csv"}]}';
```

To update the previous example to include uppercase and lowercase characters, replace `[a-z]` with `[A-Za-z]`.

The following example specifies all files in the `data_files` folder that have a numerical suffix that end in `0` with one preceding digit. For example, it uses the files `data_file_10.csv`, `data_file_20.csv`, and `data_file_30.csv`.

```
mysql> CREATE EXTERNAL TABLE table_1(col_1 int, col_2 int, col_3 int)
       ENGINE_ATTRIBUTE='{"dialect": {"format": "csv"},
                          "file": [{"region": "regionName",
                                    "namespace": "tenant_1",
                                    "bucket": "bucket_1",
                                    "pattern": "data_files/data_file_\\\\d0\\\\.csv"}]}';
```

To update the previous example to load files with one or more digits preceding the 0, update `\\\\d0\\\\` with `\\\\d+0\\\\`.

What's Next

After successfully creating external tables manually and specifying the files to load data into the table, learn how to Load Structured Data Manually.

4.3.9.6 Use PAR to Create External Tables Manually

Review the following examples to see the different ways to use pre-authenticated requests (PAR) to specify files when creating external tables manually.

This topic contains the following sections:

- Before You Begin
- Types of PARs
- PAR Recommendations
- PAR Parameters
- PAR Examples
- Specify a Single File
- Specify Multiple Files for One External Table
- Specify Files with a Prefix
- Specify Files with a Pattern
- What's Next

Before You Begin

- Review the requirements and recommendations to Access Object Storage with Pre-Authenticated Requests.
- Create the external tables that will ingest the loaded data by either using SQL syntax or JSON syntax.
  - CREATE TABLE Statement for SQL syntax. Also review relevant examples showing how to create external tables.
  - CREATE TABLE Statement for JSON syntax. Also review relevant examples showing how to create external tables.

- Review how to Create an External Table Manually.

Types of PARs

You can create the following types of PARs:

- Bucket or prefix PAR: This type of PAR ends with `/o/`. You can specify multiple files with this PAR. You can also use `name`, `prefix`, or `pattern` parameters to specify files in the bucket or Object Storage folder.
- Object PAR: This type of PAR ends with `/o/object_name`. This PAR specifies individual files. You cannot use `name`, `prefix`, or `pattern` parameters with this PAR.

PAR Recommendations

When creating PARs consider the following recommendations:

- Only use read-only PARs.
- Set a short expiration date for the PAR URL that matches the data loading plan.
- Do not make a PAR URL publicly accessible.
- If the target defines a bucket or uses a prefix or pattern:
  - Use **Enable Object Listing** when creating the PAR in the Oracle Cloud Infrastructure (OCI) console.
  - When creating the PAR from the command line, include the `--access-type AnyObjectRead` parameter.

Note

Use a resource principal for access to more sensitive data in Object Storage as it is more secure. See Access Object Storage with Resource Principals.

PAR Parameters

To set up PARs, you configure the following parameters:

- `URL` or `par`: Provide the PAR URL.
- `FILE_NAME` or `name`: Use this to specify a file for an object or bucket PAR.
- `FILE_PATTERN` or `pattern`: Use this to set a regular expression that defines a set of Object Storage files. The pattern follows the modified Modified ECMAScript regular expression grammar.
- `FILE_PREFIX` or `prefix`: Use this to define a set of Object Storage folders and files.

PAR Examples

The following examples use these parameters to create the external tables manually and specify the files to load the tables with. Replace the values in the examples with your own.

- `FILE_FORMAT` or `dialect` defines the format options of the specified files.
- `FILES` or `file` defines the file or files to load.

Note

If you are on MySQL 9.1.2 and earlier, you need to update `dialect` with the `field delimiter` and `record delimiter` parameters. As of MySQL 9.2.0,

> MySQL HeatWave Lakehouse can automatically detect these values. See Lakehouse External Table Syntax to learn more.

To review all syntax options for loading external files, see Lakehouse External Table Syntax.

Specify a Single File

To specify a single file with a pre-authenticated request, you can create a PAR for that individual file or specify the file in the command.

The following examples specify a single file by using a PAR for that file.

SQL syntax example:

```
mysql> CREATE EXTERNAL TABLE table_1(col_1 int, col_2 int, col_3 int)
       FILE_FORMAT = (FORMAT csv)
       FILES = (URL = 'https://objectstorage.us-ashburn-1.oraclecloud.com/p/.../n/tenancy_1/b/bucket_1
```

JSON syntax example:

```
mysql> CREATE EXTERNAL TABLE table_1(col_1 int, col_2 int, col_3 int)
       ENGINE_ATTRIBUTE='{"dialect": {"format": "csv"},
                         "file": [{"par": "https://objectstorage.us-ashburn-1.oraclecloud.com/p/.../n
```

The following examples specify a single file by using a PAR for a bucket and naming the file in the `name` parameter. The file is in the `data_files` folder.

SQL syntax example:

```
mysql> CREATE EXTERNAL TABLE table_1(col_1 int, col_2 int, col_3 int)
       FILE_FORMAT = (FORMAT csv)
       FILES = (URL = 'https://objectstorage.us-ashburn-1.oraclecloud.com/p/.../n/tenant_1/b/bucket_1/
                FILE_NAME = 'data_files/data_file_1.csv');
```

JSON syntax example:

```
mysql> CREATE EXTERNAL TABLE table_1(col_1 int, col_2 int, col_3 int)
       ENGINE_ATTRIBUTE='{"dialect": {"format": "csv"},
                         "file": [{"par": "https://objectstorage.us-ashburn-1.oraclecloud.com/p/.../n
                         "name": "data_files/data_file_1.csv"}]}';
```

Specify Multiple Files for One External Table

If you want to specify multiple files for one external table, you can do one of the following:

- Create a folder in the Object Storage bucket and upload the required files into that folder. See Managing Folders in an Object Storage Bucket in *Oracle Cloud Infrastructure Documentation*.
- Create a PAR for a bucket or folder, and then specify each file to load with the `name` parameter.
- Create a PAR for each file to load, and then specify the files as separate items in the `ENGINE_ATTRIBUTE` parameter.

To specify multiple external files into one external table, you can create a folder in the Object Storage bucket and upload the required files into that folder. See Managing Folders in an Object Storage Bucket in *Oracle Cloud Infrastructure Documentation*.

If you create a folder with the files to load, create a PAR for that folder and do the following:

- For the `Pre-Authenticated Request Target`, select **Objects with prefix**.
- Use **Enable Object Listing**.

The following examples specify all the files uploaded to the folder in the Object Storage bucket for one external table.

SQL syntax example:

```
mysql> CREATE EXTERNAL TABLE table_1(col_1 int, col_2 int, col_3 int)
        FILE_FORMAT = (FORMAT csv)
        FILES = (URL = 'https://objectstorage.us-ashburn-1.oraclecloud.com/p/.../n/tenant_1/b/bucket_1/o/'
```

JSON syntax example:

```
mysql> CREATE EXTERNAL TABLE table_1(col_1 int, col_2 int, col_3 int)
        ENGINE_ATTRIBUTE='{"dialect": {"format": "csv"},
                          "file": [{"par": "https://objectstorage.us-ashburn-1.oraclecloud.com/p/.../n/ten
```

The following examples use one PAR for a bucket and specify two files to load with the `name` parameter.

SQL syntax example:

```
mysql> CREATE EXTERNAL TABLE table_1(col_1 int, col_2 int, col_3 int)
        FILE_FORMAT = (FORMAT csv)
        FILES = (URL = 'https://objectstorage.us-ashburn-1.oraclecloud.com/p/.../n/tenant_1/b/bucket_1/o/'
                 URL = 'https://objectstorage.us-ashburn-1.oraclecloud.com/p/.../n/tenant_1/b/bucket_1/o/'
```

JSON syntax example:

```
CREATE EXTERNAL TABLE table_1(col_1 int, col_2 int, col_3 int)
        ENGINE_ATTRIBUTE='{"dialect": {"format": "csv"},
                          "file": [{"par": "https://objectstorage.us-ashburn-1.oraclecloud.com/p/.../n/ten
                                   {"par": "https://objectstorage.us-ashburn-1.oraclecloud.com/p/.../n/ten
```

The following examples specify two separate PARs for two files.

SQL syntax example:

```
mysql> CREATE EXTERNAL TABLE table_1(col_1 int, col_2 int, col_3 int)
        FILE_FORMAT = (FORMAT csv)
        FILES = (URL = 'https://objectstorage.us-ashburn-1.oraclecloud.com/p/.../n/tenant_1/b/bucket_1/o/da
                 URL = 'https://objectstorage.us-ashburn-1.oraclecloud.com/p/.../n/tenant_1/b/bucket_1/o/da
```

JSON syntax example:

```
CREATE EXTERNAL TABLE table_1(col_1 int, col_2 int, col_3 int)
        ENGINE_ATTRIBUTE='{"dialect": {"format": "csv"},
                          "file": [{"par": "https://objectstorage.us-ashburn-1.oraclecloud.com/p/.../n/ten
                                   {"par": "https://objectstorage.us-ashburn-1.oraclecloud.com/p/.../n/ten
```

Specify Files with a Prefix

You can add a `prefix` parameter to a PAR to specify files. To do this, create a PAR for the folder that stores the files to load.

The following examples specify all files in the `data_files` folder that begin with `data_file_`. For example, it uses the files `data_file_1`, `data_file_2`, and `data_file_3`.

SQL syntax example:

```
mysql> CREATE EXTERNAL TABLE table_1(col_1 int, col_2 int, col_3 int)
        FILE_FORMAT = (FORMAT csv)
        FILES = (URL = 'https://objectstorage.us-ashburn-1.oraclecloud.com/p/.../n/tenant_1/b/bucket_1/o/'
                 FILE_PREFIX = 'data_files/data_file_');
```

JSON syntax example:

```
mysql> CREATE EXTERNAL TABLE table_1(col_1 int, col_2 int, col_3 int)
        ENGINE_ATTRIBUTE='{"dialect": {"format": "csv"},
                          "file": [{"par": "https://objectstorage.us-ashburn-1.oraclecloud.com/p/.../n/ten
```

```
                            "prefix": "data_files/data_file_"}]}';
```

Specify Files with a Pattern

You can add a `pattern` parameter to a PAR to use regular expression to specify files.

The regular expression syntax requires certain characters to have an escape character.

Note

- When creating the table with JSON syntax, the escape character is the backslash character, and it is a reserved character in both JSON and MySQL. Therefore, it is necessary to escape the backslash character twice, and specify `\\` for both JSON and MySQL. However, the regular expression escape sequence depends upon the `NO_BACKSLASH_ESCAPES` SQL mode:
  - Use `\\.` to escape a period if `NO_BACKSLASH_ESCAPES` is enabled.
  - Use `\\\\.` to escape a period if `NO_BACKSLASH_ESCAPES` is not enabled. The following examples use this sequence because it is the default mode.
- When creating the table with SQL syntax, it is not necessary to escape the backslash character twice.

See the following to learn more:

- Regular Expression Syntax
- Modified ECMAScript regular expression grammar

The following examples specify all files in the `data_files` folder that have a numerical suffix of one or more digits to load into the external table. For example, it uses the files `data_file_1.csv`, `data_file_2.csv`, and `data_file_3.csv`.

SQL syntax example:

```
mysql> CREATE EXTERNAL TABLE table_1(col_1 int, col_2 int, col_3 int)
        FILE_FORMAT = (FORMAT csv)
        FILES = (URL = 'https://objectstorage.us-ashburn-1.oraclecloud.com/p/.../n/tenant_1/b/bucket_1/
                 FILE_PATTERN = 'data_files/data_file_\\d+\\.csv');
```

JSON syntax example:

```
mysql> CREATE EXTERNAL TABLE table_1(col_1 int, col_2 int, col_3 int)
        ENGINE_ATTRIBUTE='{"dialect": {"format": "csv"},
                          "file": [{"par": "https://objectstorage.us-ashburn-1.oraclecloud.com/p/.../n
                          "pattern": "data_files/data_file_\\\\d+\\\\.csv"}]}';
```

The following examples specify all files in the `data_files` folder that have an alphabetical suffix of one or more lowercase characters. For example, it uses the files `data_file_a.csv`, `data_file_b.csv`, and `data_file_c.csv`.

SQL syntax example:

```
mysql> CREATE EXTERNAL TABLE table_1(col_1 int, col_2 int, col_3 int)
        FILE_FORMAT = (FORMAT csv)
        FILES = (URL = 'https://objectstorage.us-ashburn-1.oraclecloud.com/p/.../n/tenant_1/b/bucket_1/
                 FILE_PATTERN = 'data_files/data_file_[a-z]+\\.csv');
```

JSON syntax example:

```
mysql> CREATE EXTERNAL TABLE table_1(col_1 int, col_2 int, col_3 int)
        ENGINE_ATTRIBUTE='{"dialect": {"format": "csv"},
```

```
                            "file": [{"par": "https://objectstorage.us-ashburn-1.oraclecloud.com/p/.../n/ten
                            "pattern": "data_files/data_file_[a-z]+\\\\.csv"}]}';
```

To update the previous examples to include uppercase and lowercase characters, replace `[a-z]` to `[A-Za-z]`.

The following examples specify all files in the `data_files` folder that have a numerical suffix that end in `0` with one preceding digit. For example, it uses the files `data_file_10.csv`, `data_file_20.csv`, and `data_file_30.csv`.

SQL syntax example:

```
mysql> CREATE EXTERNAL TABLE table_1(col_1 int, col_2 int, col_3 int)
        FILE_FORMAT = (FORMAT csv)
        FILES = (URL = 'https://objectstorage.us-ashburn-1.oraclecloud.com/p/.../n/tenant_1/b/bucket_1/o/'
                 FILE_PATTERN = 'data_files/data_file_\\d0\\.csv');
```

JSON syntax example:

```
mysql> CREATE EXTERNAL TABLE table_1(col_1 int, col_2 int, col_3 int)
        ENGINE_ATTRIBUTE='{"dialect": {"format": "csv"},
                            "file": [{"par": "https://objectstorage.us-ashburn-1.oraclecloud.com/p/.../n/ten
                            "pattern": "data_files/data_file_\\\\d0\\\\.csv"}]}';
```

To update the previous example to load files with one or more digits preceding the 0, update `d0` with `d+0`.

What's Next

After successfully creating external tables manually and specifying the files to load data into the table, learn how to Load Structured Data Manually.

4.3.10 Create Tables with Stored Generated Columns

As of MySQL 9.5.1, you have the option to include stored generated columns when creating external Lakehouse tables. To learn more about generated columns, see CREATE TABLE and Generated Columns.

Before You Begin

- Prepare to create external tables by reviewing the following:
  - Additional MySQL HeatWave Lakehouse Requirements
  - MySQL HeatWave Lakehouse Privileges
  - Supported File Format
  - Access Object Storage
- Review how to create external tables using Auto Parallel Load or manually.

Overview of Stored Generated Columns

The syntax for creating external tables with generated columns is the following:

```
CREATE EXTERNAL TABLE table_name
(col_name data_type ['Col_Options_SQL'], ...
col_name data_type [GENERATED ALWAYS] AS (expr)
  STORED [NOT NULL | NULL]
  [UNIQUE [KEY]] [[PRIMARY] KEY]
  [COMMENT 'string'], ...
)Tbl_Options_SQL;
```

Learn more about Lakehouse External Table SQL Syntax and Lakehouse External Table JSON Syntax.

- You can include stored generated columns for CSV, Parquet, and Avro file formats. Stored generated columns are not supported for the JSON file format, unstructured format, Delta Lake tables, or the `VECTOR` data type.
- You must use the `STORED` keyword, otherwise a `VIRTUAL` generated column is assumed, which is not supported for external tables.
- The expression (`expr`) for stored generated columns can include any of the following functions:
  - `ABS()`
  - `CEILING()`
  - `CONCAT()`
  - `DATEDIFF()`
  - `DAY()`
  - `DAYOFMONTH()`
  - `DAYOFWEEK()`
  - `DAYOFYEAR()`
  - `EXTRACT()`
  - `FLOOR()`
  - `HOUR()`
  - `JSON_EXTRACT()`
  - `MICROSECOND()`
  - `MINUTE()`
  - `MOD()`
  - `MONTH()`
  - `QUARTER()`
  - `SECOND()`
  - `SUBSTR()`
  - `SUBSTRING_INDEX()`
  - `TIME_TO_SEC()`
  - `TO_DAYS()`
  - `TO_SECONDS()`
  - `WEEKDAY()`
  - `YEAR()`
  - `YEARWEEK()`

Any combination of the previous list of supported functions that is supported by MySQL HeatWave is also supported for stored generated columns. See Supported Functions and Operators to learn more.

- Generated column expressions can be in any of the following forms:
  - A function over an existing column of the table. For example:

```
CREATE EXTERNAL TABLE customer (
     date_col DATE,
     month_col INT GENERATED ALWAYS AS (MONTH(`date_col`)) STORED
);
```

  - A function over a constant value. For example:

```
CREATE EXTERNAL TABLE customer (
    date_col DATE,
    month_col INT GENERATED ALWAYS AS (MONTH('2008-02-03')) STORED
);
```

  - The expression refers directly to another column of the table. For example:

```
CREATE EXTERNAL TABLE customer (
    date_col DATE,
    genColDate DATE GENERATED ALWAYS AS (`date_col`) STORED
);
```

  - The expression refers to a constant value. For example:

```
CREATE EXTERNAL TABLE customer (
    date_col DATE,
    genColDate DATE GENERATED ALWAYS AS ('2008-02-03') STORED
);
```

- If you load an external table with generated columns, either with Auto Parallel Load by using `HEATWAVE_LOAD` or manually with Guided Load, the table must be prevously created. See Load External Data Using Lakehouse Auto Parallel Load with an Existing Table and Guided Load.
- You can include the `AS` keyword before `GENERATED ALWAYS` to make the generated nature of the column more explicit.
- Stored generated columns are evaluated when loading data into external tables using the `ALTER TABLE` *`table_name`* `SECONDARY_LOAD` statement. After data is loaded, subsequent queries on the table display resulting values of the generated columns.
- Due to the fact that the query for stored generated columns runs at the same time that data is loaded to an external table, statistics are not available beforehand, and therefore MySQL HeatWave might overestimate the required size of `VARLEN` columns. This can lead to predictions that the load exceeds 4MB and an error.
- For additional limitations, see Lakehouse Limitations for all File Formats.

Stored Generated Columns Example

The following example creates an external table with stored generated columns:

```
CREATE EXTERNAL TABLE customer (
    customer_id INT PRIMARY KEY AUTO_INCREMENT,
    first_name VARCHAR(100) NOT NULL,
    last_name VARCHAR(100) NOT NULL,
    email VARCHAR(200) NOT NULL,
    birth_date DATE,
    registration_date DATETIME DEFAULT CURRENT_TIMESTAMP,
    full_name VARCHAR(201) GENERATED ALWAYS AS (CONCAT(first_name, ' ', last_name)) STORED,
    birth_year INT GENERATED ALWAYS AS (YEAR(birth_date)) STORED
```

```
);
```

The first stored generated column, `full_name`, combines the first and last name into a single string, and the second stored generated column, `birth_year`, extracts the year portion of the `birth_date`.

What's Next

- Review how to load data manually or using Auto Parallel Load with an existing table.

4.3.11 Load Structured Data Using Lakehouse Auto Parallel Load

After setting up an external table with the files to load into the table, you run `HEATWAVE_LOAD` command to automatically create the table and load the data.

This topic contains the following sections:

- Before You Begin
- Load External Data Using Lakehouse Auto Parallel Load with a New Table
- Load External Data Using Lakehouse Auto Parallel Load with an Existing Table
- What's Next

Before You Begin

Complete the steps to Create an External Table before running the `HEATWAVE_LOAD` command.

Load External Data Using Lakehouse Auto Parallel Load with a New Table

To demonstrate how to load data using Lakehouse Auto Parallel Load, the following example uses the same file in the Create External Table Example and runs the `HEATWAVE_LOAD` command in `dryrun` mode to test out the creation of the table and the loading of data beforehand.

1. Create a session variable with the characteristics of the `input_list` to create the external table and specify the files to use to load data into the table. To review all syntax options, see HEATWAVE_LOAD.

```
mysql> SET @input_list = '[{
  "db_name": "bank_marketing",
  "tables": [{
    "table_name": "bank_train",
    "engine_attribute": {
      "dialect": {"format": "csv", "has_header": true},
      "file": [{"uri": "oci://mybucket@mynamespace/data_files/data_file_1.csv"}]
    }
  }]
}]';
```

Where:

- `@input_list` is the name of the session variable.
- `db_name` identifies the database name to store the table: `bank_marketing`. The database is automatically created if it does not exist.
- `table_name` sets the table name to store the data: `bank_train`. The table is automatically created if it does not exist.
- `engine_attribute` defines the parameters of the external file.
- `format` defines the format of the external file: `csv`.

- `has_header` identifies a header in the external file. Auto Parallel Load then infers the column names from the first row in the file.
- `uri` sets the URI link to access the file. Replace `mybucket@mynamespace/data_files/data_file_1.csv` with your own link.

If you are on MySQL 9.1.2 and earlier, you need to update `dialect` with the `field delimiter` and `record delimiter` parameters. As of MySQL 9.2.0, MySQL HeatWave Lakehouse can automatically detect these values. See Lakehouse External Table Syntax to learn more.

2. Use the `HEATWAVE_LOAD` command and use the `dryrun` mode option to generate a summary, so you can review the details and address any issues before loading the data. To skip this step, run the command in `normal` mode. Insert the `input_list` variable previously created into the command.

```
mysql> CALL sys.HEATWAVE_LOAD(CAST(@input_list AS JSON), JSON_OBJECT('mode', 'dryrun'));
+-----------------------------------------+
| INITIALIZING HEATWAVE AUTO PARALLEL LOAD |
+-----------------------------------------+
| Version: 4.31                           |
|                                         |
| Load Mode: dryrun                       |
| Load Policy: disable_unsupported_columns |
| Output Mode: normal                     |
|                                         |
+-----------------------------------------+

+-----------------------------------------------------------------------------------------------------------
| LAKEHOUSE AUTO SCHEMA INFERENCE
+-----------------------------------------------------------------------------------------------------------
| Verifying external lakehouse tables: 1
|
| SCHEMA                   TABLE                    TABLE IS           RAW     NUM. OF      ESTIMATED
| NAME                     NAME                     CREATED      FILE SIZE     COLUMNS      ROW COUNT
| ------                   -----                    --------     ---------     -------      ---------
| `bank_marketing`         `bank_train`             NO          450.66 KiB          17         4.52 K
|
| New schemas to be created: 1
| External lakehouse tables to be created: 1
|
+-----------------------------------------------------------------------------------------------------------

+-------------------------------------------------------------------+
| OFFLOAD ANALYSIS                                                  |
+-------------------------------------------------------------------+
| Verifying input schemas: 1                                        |
| User excluded items: 0                                            |
|                                                                   |
| SCHEMA                        OFFLOADABLE    OFFLOADABLE     SUMMARY OF |
| NAME                               TABLES        COLUMNS     ISSUES     |
| ------                        -----------    -----------     ---------- |
| `bank_marketing`                        1             17                |
|                                                                   |
| Total offloadable schemas: 1                                      |
|                                                                   |
+-------------------------------------------------------------------+

+-----------------------------------------------------------------------------------------------------------
| CAPACITY ESTIMATION
+-----------------------------------------------------------------------------------------------------------
| Default encoding for string columns: VARLEN (unless specified in the schema)
| Estimating memory footprint for 1 schema(s)
|
|                                TOTAL       ESTIMATED       ESTIMATED       TOTAL     DICTIONARY      V
| SCHEMA                    OFFLOADABLE   HEATWAVE NODE      MYSQL NODE      STRING        ENCODED     E
| NAME                           TABLES       FOOTPRINT       FOOTPRINT     COLUMNS        COLUMNS     C
| ------                    -----------       ---------       ---------     -------     ----------     -
| `bank_marketing`                    1        4.65 MiB        1.12 MiB          17             0
|
```

```
| Sufficient MySQL host memory available to load all tables.
| Sufficient HeatWave cluster memory available to load all tables.
|
+-----------------------------------------------------------------------------------------------------

+-----------------------------------------------------------------------------------------------------
| LOAD SCRIPT GENERATION
+-----------------------------------------------------------------------------------------------------
| Dryrun mode only generates the load script
| Set mode to "normal" in options to load tables
|
| Retrieve load script containing 3 generated DDL commands using the query below:
| Deprecation Notice: "heatwave_load_report" will be deprecated, please switch to "heatwave_autopilo
|   SELECT log->>"$.sql" AS "Load Script" FROM sys.heatwave_autopilot_report WHERE type = "sql" ORDE
|
| Applying changes will take approximately 7.00 s
|
| Caution: Executing the generated load script may alter column definitions and secondary engine fla
|
| Total errors encountered: 0
| Total warnings encountered: 0
|
+-----------------------------------------------------------------------------------------------------
```

Review the following sections in the load script:

- `LAKEHOUSE AUTO SCHEMA INFERENCE`: Displays the database name, table table, file size, number of columns, and the estimated number of rows. Any issues display here.
- `OFFLOAD ANALYSIS`: Confirms the number of tables and columns that Lakehouse can load to MySQL HeatWave.
- `CAPACITY ESTIMATION`: Displays the memory requirements and estimated load time.

  If there is insufficient memory, you can update the number of nodes for the DB System. See Editing a MySQL HeatWave Cluster in the *MySQL HeatWave on OCI Service Guide* or Editing a MySQL HeatWave Cluster in the *MySQL HeatWave on AWS Service Guide*.

3. If there are no issues or errors in the summary, you can run the `HEATWAVE_LOAD` command again in `normal` mode to load the data. If you set the options to `NULL`, the data loads in normal mode by default.

```
mysql> CALL sys.HEATWAVE_LOAD(CAST(@input_list AS JSON), NULL);
+-----------------------------------------+
| INITIALIZING HEATWAVE AUTO PARALLEL LOAD |
+-----------------------------------------+
| Version: 4.31                           |
|                                         |
| Load Mode: normal                       |
| Load Policy: disable_unsupported_columns |
| Output Mode: normal                     |
|                                         |
+-----------------------------------------+

+-----------------------------------------------------------------------------------------------------
| LAKEHOUSE AUTO SCHEMA INFERENCE
+-----------------------------------------------------------------------------------------------------
| Verifying external lakehouse tables: 1
|
| SCHEMA                   TABLE                    TABLE IS           RAW     NUM. OF      ESTIMATE
| NAME                     NAME                     CREATED      FILE SIZE     COLUMNS      ROW COUN
| ------                   -----                    --------     ---------     -------      --------
| `bank_marketing`         `bank_train`             NO          450.66 KiB          17         4.52
|
| New schemas to be created: 1
| External lakehouse tables to be created: 1
|
+-----------------------------------------------------------------------------------------------------
```

```
+----------------------------------------------------------------------+
| OFFLOAD ANALYSIS                                                     |
+----------------------------------------------------------------------+
| Verifying input schemas: 1                                           |
| User excluded items: 0                                               |
|                                                                      |
| SCHEMA                       OFFLOADABLE    OFFLOADABLE     SUMMARY OF |
| NAME                              TABLES        COLUMNS     ISSUES     |
| ------                       -----------    -----------     ---------- |
| `bank_marketing`                        1             17              |
|                                                                      |
| Total offloadable schemas: 1                                         |
|                                                                      |
+----------------------------------------------------------------------+

+-------------------------------------------------------------------------------------------------------------
| CAPACITY ESTIMATION
+-------------------------------------------------------------------------------------------------------------
| Default encoding for string columns: VARLEN (unless specified in the schema)
| Estimating memory footprint for 1 schema(s)
|
|                                TOTAL       ESTIMATED       ESTIMATED       TOTAL     DICTIONARY      V
| SCHEMA                     OFFLOADABLE   HEATWAVE NODE      MYSQL NODE      STRING        ENCODED     EN
| NAME                            TABLES       FOOTPRINT       FOOTPRINT     COLUMNS        COLUMNS     CO
| ------                     -----------       ---------       ---------     -------     ----------     --
| `bank_marketing`                     1        4.65 MiB        1.12 MiB          17              0
|
| Sufficient MySQL host memory available to load all tables.
| Sufficient HeatWave cluster memory available to load all tables.
|
+-------------------------------------------------------------------------------------------------------------

+-------------------------------------------------------------------------------------------------------------
| EXECUTING LOAD SCRIPT
+-------------------------------------------------------------------------------------------------------------
| HeatWave Load script generated
|   Retrieve load script containing 3 generated DDL command(s) using the query below:
| Deprecation Notice: "heatwave_load_report" will be deprecated, please switch to "heatwave_autopilot_r
|   SELECT log->>"$.sql" AS "Load Script" FROM sys.heatwave_autopilot_report WHERE type = "sql" ORDER BY
|
| Adjusting load parallelism dynamically per internal/external table.
| Using current parallelism of 32 thread(s) as maximum for internal tables.
|
| Warning: Executing the generated script may alter column definitions and secondary engine flags in the
|
| Using SQL_MODE: ONLY_FULL_GROUP_BY,STRICT_TRANS_TABLES,NO_ZERO_IN_DATE,NO_ZERO_DATE,ERROR_FOR_DIVISION
|
| Proceeding to load 1 table(s) into HeatWave.
|
| Applying changes will take approximately 7.03 s
|
+-------------------------------------------------------------------------------------------------------------

+--------------------------------------------+
| SCHEMA CREATION                            |
+--------------------------------------------+
| Schema `bank_marketing` creation succeeded! |
| Warnings/errors encountered: 0             |
| Elapsed time: 4.91 ms                      |
|                                            |
+--------------------------------------------+

+--------------------------------------------+
| TABLE LOAD                                 |
+--------------------------------------------+
| TABLE (1 of 1): `bank_marketing`.`bank_train` |
| Commands executed successfully: 2 of 2     |
| Warnings encountered: 0                    |
| Table load succeeded!                      |
|   Total columns loaded: 17                 |
|   Elapsed time: 4.36 s                     |
|                                            |
```

```
+---------------------------------------------+

+-----------------------------------------------------------------------------+
| LOAD SUMMARY                                                                |
+-----------------------------------------------------------------------------+
|                                                                             |
| SCHEMA                          TABLES       TABLES      COLUMNS        LOAD |
| NAME                            LOADED       FAILED       LOADED    DURATION |
| ------                          ------       ------      -------    -------- |
| `bank_marketing`                     1            0           17      4.36 s |
|                                                                             |
| Total errors encountered: 0                                                 |
| Total warnings encountered: 0                                               |
|                                                                             |
+-----------------------------------------------------------------------------+
```

Review the `SCHEMA CREATION`, `TABLE LOAD`, and `LOAD SUMMARY` sections for any errors or warnings during the load, and to confirm the external table loaded successfully.

4. Optionally, generate a load script to review a summary of the database and table created for the load. The output is similar to the following example.

```
mysql> SELECT log->>"$.sql" AS "Load Script" FROM sys.heatwave_autopilot_report WHERE type = "sql" O
+-------------------------------------------------------------------------------------------------------------
| Load Script
+-------------------------------------------------------------------------------------------------------------
| CREATE DATABASE `tpch`;
| CREATE TABLE `bank_marketing`.`bank_train`(
|   `age` tinyint unsigned NOT NULL,
|   `job` varchar(13) NOT NULL COMMENT 'RAPID_COLUMN=ENCODING=VARLEN',
|   `marital` varchar(8) NOT NULL COMMENT 'RAPID_COLUMN=ENCODING=VARLEN',
|   `education` varchar(9) NOT NULL COMMENT 'RAPID_COLUMN=ENCODING=VARLEN',
|   `default` varchar(3) NOT NULL COMMENT 'RAPID_COLUMN=ENCODING=VARLEN',
|   `balance` mediumint NOT NULL,
|   `housing` varchar(3) NOT NULL COMMENT 'RAPID_COLUMN=ENCODING=VARLEN',
|   `loan` varchar(3) NOT NULL COMMENT 'RAPID_COLUMN=ENCODING=VARLEN',
|   `contact` varchar(9) NOT NULL COMMENT 'RAPID_COLUMN=ENCODING=VARLEN',
|   `day` tinyint unsigned NOT NULL,
|   `month` varchar(3) NOT NULL COMMENT 'RAPID_COLUMN=ENCODING=VARLEN',
|   `duration` smallint unsigned NOT NULL,
|   `campaign` tinyint unsigned NOT NULL,
|   `pdays` smallint NOT NULL,
|   `previous` tinyint unsigned NOT NULL,
|   `poutcome` varchar(7) NOT NULL COMMENT 'RAPID_COLUMN=ENCODING=VARLEN',
|   `y` varchar(3) NOT NULL COMMENT 'RAPID_COLUMN=ENCODING=VARLEN')
|  ENGINE=lakehouse SECONDARY_ENGINE=RAPID
|  ENGINE_ATTRIBUTE='{"file": [{"par": "https://objectstorage.../bank.csv"}],
|                     "dialect": {"format": "csv", "has_header": true, "field_delimiter": ";", "rec
|  ALTER TABLE /*+ AUTOPILOT_DISABLE_CHECK */ `bank_marketing`.`bank_train` SECONDARY_LOAD;
+-------------------------------------------------------------------------------------------------------------
```

5. Optionally, query five rows of the table to confirm the data is loaded and accessible.

```
mysql> SELECT * FROM bank_marketing.bank_train LIMIT 5;
+-----+-------------+----------+-----------+---------+---------+---------+------+----------+-----+-
| age | job         | marital  | education | default | balance | housing | loan | contact  | day |
+-----+-------------+----------+-----------+---------+---------+---------+------+----------+-----+-
|  37 | technician  | single   | secondary | no      |     267 | no      | no   | cellular |  13 |
|  34 | technician  | single   | tertiary  | no      |     586 | yes     | no   | unknown  |  16 |
|  30 | blue-collar | single   | secondary | no      |     648 | yes     | yes  | unknown  |   8 |
|  57 | entrepreneur| divorced | secondary | yes     |      25 | yes     | no   | cellular |  11 |
|  26 | admin.      | single   | secondary | no      |      96 | yes     | no   | cellular |  13 |
+-----+-------------+----------+-----------+---------+---------+---------+------+----------+-----+-
```

Once you confirm the table successfully loaded into Lakehouse, you can use the data for the following:

- Refresh Data Using Event-Based Incremental Load
- Accelerate Query Processing

- Train and Use Machine Learning Models
- Perform AI-Powered Search and Content Generation

Load External Data Using Lakehouse Auto Parallel Load with an Existing Table

If you previously created a table and you want to load it with the specified file, you can use a single command to load it with Auto Parallel Load.

For example, you created a table with the following command.

```
mysql> CREATE EXTERNAL TABLE table_1(col_1 int, col_2 int, col_3 int)
        FILE_FORMAT = (FORMAT csv)
        FILES = (URI = 'oci://mybucket@mynamespace/data_files/data_file_1.csv');
```

You can load the table to the MySQL HeatWave Cluster with data from the specified file later on with the following command:

```
mysql> CALL sys.HEATWAVE_LOAD('[ {"db_name": "db_1", "tables": ["table_1"]} ]', NULL);
```

The value `db_1` refers to the name of the database storing the table, and `table_1` refers to the name of the table to load.

What's Next

After successfully creating the external table and loading it with data, learn how to do the following:

- Accelerate Query Processing
- Train and Use Machine Learning Models
- Perform AI-Powered Search and Content Generation
- Refresh Data Using Event-Based Incremental Load

4.3.12 Load Structured Data Manually

After creating an external table manually and specifying the files to load into the table, you have the option (as of MySQL 9.6.0) to validate a load command before loading data into the external table.

This topic contains the following sections:

- Before You Begin
- Load and Validate Tables Statement
- Guided Load
- Validate Load
- Load External Data Manually Example
- What's Next

Before You Begin

Complete the steps to Create an External Table Manually.

Load and Validate Tables Statement

To load an external table into MySQL HeatWave Cluster with data from files, specify the `SECONDARY_LOAD` clause in an `ALTER TABLE` statement.

As of MySQL 9.6.0, you have the option to validate a load operation before loading data. To do this, specify the SECONDARY_LOAD and VALIDATE ONLY clauses in an ALTER TABLE statement.

Use the following syntax:

```
ALTER TABLE table_name SECONDARY_LOAD VALIDATE [{n | ALL} ROWS] ONLY [GUIDED {ON | OFF}];
```

Specify the following clauses in the ALTER TABLE statement:

- table_name: Specify the name of the table to load to MySQL HeatWave.
- VALIDATE ONLY: Available as of MySQL 9.6.0. Allows you to validate the load. Lakehouse validates the rows in the external table, but does not load the data. See Validate Load.
- GUIDED: Available as of MySQL 9.4.1. Allows you to disable and enable Guided Load as needed. Set to OFF to disable Guided Load. By default, Guided Load is enabled. See Guided Load.

Guided Load

Available as of MySQL 8.2.0, the Guided Load feature performs a set of checks and validations before loading data.

These checks include the following:

- Automatically detect tables and columns that cannot be loaded. If there are tables and columns that are not compatible, stop the load.
- Automatically set SECONDARY_ENGINE to rapid.
- Detect any errors with ENGINE_ATTRIBUTE and report them.
- Infer the table definition and make any necessary adjustments before loading data. These adjustments are similar to those performed by Autopilot during Lakehouse Auto Parallel Load. See: About Lakehouse Auto Parallel Load Schema Inference. If the inferred table definition is not compatible, stop the load.
- Predict the amount of memory required for loading data. If the required memory is not available, stop the load.
- Infer the record and field delimiters for CSV files and make the necessary adjustments.

As of MySQL 9.4.1, you have the option to disable Guided Load if you want to skip these checks.

The following example manually creates an external table, and then loads the table into MySQL HeatWave with Guided Load disabled:

```
mysql> CREATE EXTERNAL TABLE table_1(col_1 int, col_2 int, col_3 int)
        FILE_FORMAT = (FORMAT csv)
        FILES = (URI = 'oci://mybucket@mynamespace/data_files/data_file_1.csv');
mysql> ALTER TABLE table_1 SECONDARY_LOAD GUIDED OFF;
```

Validate Load

Available as of MySQL 9.6.0, the VALIDATE ONLY clause allows you to validate a load without loading any data.

- You have the option to specify the number of rows to validate, or use ALL ROWS to validate all rows.
- If the number of rows is not specified, or ALL ROWS is not included, then the default is to validate all rows.
- Review any errors or warning messages before running the load command.
- If GUIDED ON is specified, or the statement does not contain the GUIDED clause, then Guided Load is enabled for the statement. With Guided Load enabled, the table definition may be modified.

The following example manually creates an external table, and then validates 1000 rows from the external table. No data is loaded. Guided Load is enabled by default.

```
mysql> CREATE EXTERNAL TABLE table_1(col_1 int, col_2 int, col_3 int)
        FILE_FORMAT = (FORMAT csv)
        FILES = (URI = 'oci://mybucket@mynamespace/data_files/data_file_1.csv');
mysql> ALTER TABLE table_1 SECONDARY_LOAD VALIDATE 1000 ROWS ONLY;
```

Load External Data Manually Example

To demonstrate how to load data manually, the following example uses the same file in the Create External Table Manually Example.

1. Create the external table. For this example, use the following command:

```
mysql> CREATE EXTERNAL TABLE bank_marketing(
        age int,
        job varchar(255),
        marital varchar(255),
        education varchar(255),
        default1 varchar(255),
        balance float,
        housing varchar(255),
        loan varchar(255),
        contact varchar(255),
        day int,
        month varchar(255),
        duration float,
        campaign int,
        pdays float,
        previous float,
        poutcome varchar(255),
        y varchar(255)
    )
        FILE_FORMAT = (FORMAT csv HEADER ON)
        FILES = (URI = 'oci://mybucket@mynamespace/bucket_1/bank.csv');
```

2. Optionally, validate the load before loading the external table.

```
mysql> ALTER TABLE bank_marketing SECONDARY_LOAD VALIDATE ONLY;
Warning (code 3877): Command executed during preprocessing: 'ALTER TABLE `tests`.`bank_marketing`
ENGINE_ATTRIBUTE='{"file": [{"uri": "oci://Carlos_bucket@mysql2/bank.csv"}], "dialect": {"format": "csv
```

   The command runs with Guided Load enabled, so changes to the table definition are included in a warning message. Review errors and warnings during validation before proceeding to load the external table.

3. Use the `ALTER TABLE` clause in the `SECONDARY_LOAD` command to load the data into the table.

```
mysql> ALTER TABLE bank_marketing SECONDARY_LOAD;
Warning (code 3877): Command executed during preprocessing: 'ALTER TABLE `bank_data`.`bank_marketing`
ENGINE_ATTRIBUTE='{"file": [{"par": "https://objectstorage.us-ashburn-1.oraclecloud.com/p/.../n/tenant_
                   "dialect": {"format": "csv", "has_header": true, "field_delimiter": ";", "record_del
```

   Review the message to confirm the external table is successfully created. For MySQL 9.1.2 and later, you can also review the `field_delimiter` and `record_delimiter` values that MySQL HeatWave automatically detected for the table.

4. Optionally, query five rows of the table to confirm the data is loaded and accessible.

```
mysql> SELECT * FROM bank_marketing.bank_train LIMIT 5;
+-----+-------------+----------+-----------+----------+---------+---------+------+-----------+-----+---
| age | job         | marital  | education | default1 | balance | housing | loan | contact   | day | mo
+-----+-------------+----------+-----------+----------+---------+---------+------+-----------+-----+---
|  37 | services    | married  | secondary | no       |    4760 | yes     | no   | cellular  |   8 | ma
|  32 | technician  | single   | secondary | no       |    2979 | no      | no   | cellular  |  25 | ma
|  43 | management  | married  | tertiary  | no       |     690 | yes     | no   | cellular  |   6 | au
|  50 | blue-collar | divorced | secondary | no       |     203 | yes     | no   | telephone |  19 | no
|  34 | blue-collar | married  | secondary | no       |     322 | yes     | no   | cellular  |  20 | ap
```

```
+-----+------------+----------+----------+----------+---------+---------+------+----------+-----
```

What's Next

After successfully loading the table to MySQL HeatWave Cluster with data from the specified files, learn how to do the following:

- Accelerate Query Processing
- Train and Use Machine Learning Models
- Perform AI-Powered Search and Content Generation
- Refresh Data Using Event-Based Incremental Load

4.3.13 Load Unstructured Data

MySQL HeatWave Lakehouse lets you load unstructured data from Object Storage using a MySQL HeatWave Vector Store. Learn how to Ingest Files into a Vector Store.

4.3.14 Update Tables

You can update loaded external tables by using one of the following methods:

- You can load data from Object Storage by using event-based incremental load. This feature is available as of MySQL 9.4.1.
- To prevent the data in a Lakehouse table from getting outdated, refresh the tables using Incremental Load. This feature is available as of MySQL 9.0.0.
- To edit the list of files comprising an external table without altering the directories in the Object Storage bucket, you can add or remove files using Selective Load. This feature is available as of MySQL 9.1.2.

4.3.14.1 Refresh Data Using Event-Based Incremental Load

As of MySQL 9.4.1, you can load data from Object Storage by using event-based incremental load. Once an event for an external file location is emitted due to a change to an object, Lakehouse automatically refreshes the data into the external table. This feature is currently only available for MySQL HeatWave on OCI.

Event-based incremental load automates the process of updating external tables based on changes to objects in Object Storage buckets. To do this, you set up streams in OCI and configure external tables to read from that stream to listen to Object Storage object-level events. To learn more about OCI streams, see Streaming. When an event is emitted, Lakehouse incrementally loads the object changes to the corresponding external tables. These changes include adding new files, removing deleted files, and updating modified files.

When setting up event-based incremental load, you specify the stream OCID in each external table by using the `AUTO_REFRESH_SOURCE` parameter when using SQL syntax, or the `ENGINE_ATTRIBUTE` parameter `auto_refresh_event_source` when using JSON syntax.

You can set up event-based incremental load before or after the external table is loaded. You can also update or remove event-based incremental load while an external table is loaded.

When a stream OCID for `auto_refresh_event_source` or `AUTO_REFRESH_SOURCE` is set or updated for a loaded external table, Lakehouse attempts to fetch stream information and call the get-messages API to verify that it can read from the stream. If either of these two operations fail, a `WARNING` is returned with one of the following messages:

- Could not fetch stream information. Please check the Stream OCID and your access policies.
- Could not read messages from the stream. Please check your access policies.

You can modify the `auto_refresh_event_source` or `AUTO_REFRESH_SOURCE` stream OCID for a loaded external table with one of the following results:

- OCID Added: Validations and potential warnings as above.
- OCID Removed: Lakehouse drops all remaining queued events for this table. Any currently ongoing auto refresh is finished.
- OCID Updated: Validations and potential warnings as above. Lakehouse performs a full incremental load of the table to guarantee consistency.

Keep in mind the following when setting up event-based incremental load:

- There is no guarantee that updates to external tables will be applied in the same order as updates to the files in Object Storage.
- For event-based incremental load, the strict mode (enabled by `STRICT_LOAD` and `is_strict_mode` parameters) does not verify whether the external path used for loading data becomes empty. This can lead to loaded external tables being empty.
- The setting for the `ALLOW_MISSING_FILES` parameter is ignored for event-based incremental load, which means it is possible to have an empty external table.

This topic contains the following sections:

- Before You Begin
- Create and Set Up Streams
- Set Up Policy Statements
- Set Up Object-Level Events
- Set Up Event Rules
- Set Up External Table with Event-Based Incremental Load
- What's Next

Before You Begin

- Prepare to load data by reviewing the following:
  - Additional MySQL HeatWave Lakehouse Requirements
  - MySQL HeatWave Lakehouse Privileges
  - Supported File Formats
  - Access Object Storage
- Review the following requirements and recommendations for setting up event-based incremental load:
  - Object Storage object-level events must be enabled for the bucket that stores the files for the external tables. See Set Up Object-Level Events.
  - External tables and stream OCIDs have the following restrictions:
    - Each external table can only have one stream OCID.
    - A single stream OCID can be configured for multiple external tables.
  - When setting up an external table with event-based incremental load, it is not possible to use the following types of PARs:

  - Bucket PARs (end in `/o/`)
  - Prefix PARs (end in `/o/`)
  - PAR URIs that do not have an object path (end in `/o/`).

  If an external table specifies one of these types of PARs and also specifies a stream for event-based incremental load, an error is returned.

- We recommend not sharing stream OCIDs across DB Systems, and limiting the usage of each stream OCID for only event-based incremental load.
- Renaming an object in an Object Storage bucket does not trigger an event. To trigger an event, you must delete and re-upload the object, or overwrite the object.
- It is typically sufficient to set up streams with only one partition. You should set up a stream with multiple partitions when you plan to do many object operations that should be incrementally loaded automatically through events. Lakehouse reads at most 1 MB of raw stream message payload per partition per second. See Limits on Streaming Resources to learn more. Consider this limit when calculating how many partitions are needed. Object Storage events are JSON objects. For relevant examples, see Contents of an Events Message and Services that Produce Events: Object Storage.

Create and Set Up Streams

The Events service requires an intermediary for events persistence and forwarding. To achieve this, the OCI Streams service is used, due to its benefit of persistence guarantees and asynchronous event delivery.

Refer to the following resources from *Oracle Cloud Infrastructure Documentation* to learn more about streaming:

- Getting Started with Streaming
- Security Best Practices for Streaming
- Partitioning a Stream
- Managing Streams

To create the stream, you have the option to use the OCI console, the command-line, or API. See Creating a Stream.

The default limit of partitions you can create is 200. To check your current limit, use the following command:

```
$ oci limits value list --compartment-id ocid1.tenancy.oc1... --service-name streaming
```

Replace *`ocid1.tenancy.oc1...`* with the appropriate compartment-id value.

Set Up Policy Statements

After creating the required streams, you need to set up policy statements that give appropriate dynamic groups access to the streams.

Refer to the following resources from *Oracle Cloud Infrastructure Documentation* to learn more about policies and dynamic groups:

- Overview of Working with Policies
- Managing Policies (IAM with Identity Domains)
- Managing Policies (IAM without Identity Domains)

- Managing Dynamic Groups (IAM with Identity Domains)
- Managing Dynamic Groups (IAM without Identity Domains)

You need to update the appropriate policy with policy statements that give dynamic groups access to the streams. The dynamic group you specify needs to be the one the DB System belongs to. You can include multiple stream IDs in one policy statement. To get the required stream OCID, select the stream from the *Streams* page in the OCI console.

To create the policy statement, use the following syntax:

```
Allow dynamic-group dynamic_group_name to use stream-pull in compartment compartment_name where any {target
```

Replace *dynamic_group_name*, *compartment_name*, and *stream_OCID_N* with your own values for the DB System group and all streams you use for Lakehouse, respectively.

Refer to the following resources from *Oracle Cloud Infrastructure Documentation* to learn more about stream details and policy statements:

- Getting Details for a Stream
- Updating a Policy's Statements (IAM with Identity Domains)
- Writing Policy Statements with the Policy Builder (IAM without Identity Domains)
- Policy Syntax (IAM without Identity Domains)

Set Up Object-Level Events

The Object Storage buckets that store the files need to enable the emitting of events. See Managing Emitting Events for Object State Changes in an Object Storage Bucket.

Set Up Event Rules

You need to set up the rules to have events routed from Object Storage to the streams. It is important to filter events appropriately before they are sent to the stream, so that only events relevant for the external file location are emitted. It is not possible to filter events after they are sent to the stream. See Matching Events with Filters to learn more.

The default limit of event rules you can create is 50. To check your current limit, use the following command:

```
$ oci limits value list --compartment-id ocid1.tenancy.oc1.. --service-name events
```

Replace *ocid1.tenancy.oc1...* with the appropriate compartment-id value.

Creating a rule involves the following steps:

1. Create the rule.
2. Extend the rule with attribute filters.

When creating the event rule in the OCI console, do the following:

- For **Condition**, select **Event Type**.
- For **Service Name**, select **Object Storage**
- For **Event Type**, select all of **Object - Create**, **Object - Update**, and **Object - Delete**.
- For **Action Type**, select **Streaming**. Then, also select the appropriate compartment and streams. You can add multiple destinations by clicking **+ Another Action**.
- Optionally, add tags to the rule in **Advanced options**.

- Click **Create Rule**.

After creating an event-type filter condition, you should add more filters on Object Storage-specific attributes by clicking **+ Another Condition**. Select the **Condition** of **Attribute**. You should only add more filters for the following attributes. You can use an asterisk (*) as a wildcard if needed.

- bucketId (an OCID)
- bucketName
- compartmentId (an OCID)
- compartmentName
- namespace
- resourceName

Please note that Lakehouse cannot cross-check event rules and file locations specified for the external table in `ENGINE_ATTRIBUTE` options.

It is also possible to change the event attribute filters later. To do that, go to the event's details page, and on the left-side under **Resources** select **Event Matching**. Then, under **Attributes**, select **Add Attribute**.

See Creating an Events Rule to learn more.

Set Up External Table with Event-Based Incremental Load

The final step is to set up the external table with the streams so that it can load data automatically with event-based incremental load. For the initial load, you must specify at least one external file location with the `file` or `FILES` option.

If you create the table manually, you can use the `CREATE EXTERNAL TABLE` command to set it up before loading data into the table. Then, use `ALTER TABLE` to load the table.

To set up the external table with a stream, use the `AUTO_REFRESH_SOURCE` parameter when using SQL syntax, or the `ENGINE_ATTRIBUTE` parameter `auto_refresh_event_source` when using JSON syntax.

The following example uses Auto Parallel Load to automatically create a new table and set up a stream. The `HEATWAVE_LOAD` command loads data into the table.

```
mysql> SET @input_list = '[{
  "db_name": "lakehouse_db",
  "tables": [{
    "table_name": "table_1",
    "engine_attribute": {
      "dialect": {"format": "csv"},
      "auto_refresh_event_source": "ocid1.stream.oc1.iad...",
      "file": [{"uri": "oci://mybucket@mynamespace/data_files/data_file_1.csv"}]
    }
  }]
}]';
mysql> CALL sys.HEATWAVE_LOAD(CAST(@input_list AS JSON), NULL);
```

The following example uses SQL syntax to create a new external table and set up a stream. The `HEATWAVE_LOAD` command loads the table with Auto Parallel Load.

```
mysql> CREATE EXTERNAL TABLE table_1(
        col_1 int,
        col_2 int,
        col_3 int)
        FILE_FORMAT = (FORMAT csv)
        FILES = (URI = 'oci://mybucket@mynamespace/data_files/data_file_1.csv')
        AUTO_REFRESH_SOURCE = 'ocid1.stream.oc1.iad...';
mysql> CALL sys.HEATWAVE_LOAD('[ {"db_name": "lakehouse_db", "tables": ["table_1"]} ]', NULL);
```

The following example uses SQL syntax to create a new external table and set up a stream. The SECONDARY_LOAD command loads the table with Guided Load.

```
mysql> CREATE EXTERNAL TABLE table_1(
        col_1 int,
        col_2 int,
        col_3 int)
        FILE_FORMAT = (FORMAT csv)
        FILES = (URI = 'oci://mybucket@mynamespace/data_files/data_file_1.csv')
        AUTO_REFRESH_SOURCE = 'ocid1.stream.oc1.iad...';
mysql> ALTER TABLE table_1 SECONDARY_LOAD;
```

The following example uses JSON syntax and updates a table with a stream. The SECONDARY_LOAD command loads the table with Guided Load.

```
mysql> CREATE EXTERNAL TABLE table_1(
        col_1 int,
        col_2 int,
        col_3 int)
        FILE_FORMAT = (FORMAT csv)
        FILES = (URI = 'oci://mybucket@mynamespace/data_files/data_file_1.csv');
mysql> ALTER TABLE table_1
        ENGINE_ATTRIBUTE = '{"auto_refresh_event_source": "ocid1.stream.oc1.iad..."}';
mysql> ALTER TABLE table_1 SECONDARY_LOAD;
```

What's Next

After successfully setting up a table with event-based data loading, learn how to do the following:

- Accelerate Query Processing
- Train and Use Machine Learning Models
- Perform AI-Powered Search and Content Generation
- Review the relevant syntax for loading data:
  - HEATWAVE_LOAD
  - Lakehouse External Table Syntax

4.3.14.2 Refresh Data Using Incremental Load

Available as of MySQL 9.0.0, Lakehouse Incremental Load allows you to refresh the data in an external table.

This topic contains the following sections:

- Before You Begin
- Refresh Data Overview
- Perform Initial Load
- Refresh Data
- What's Next

Before You Begin

- Load structured data using Lakehouse Auto Parallel Load or manually.

Refresh Data Overview

Lakehouse Incremental Load uses the existing `engine_attribute` settings from the initial load. Therefore, when you refresh data, you cannot include `engine_attribute` settings in subsequent loads of data.

You can add or remove individual files in Object Storage if they still match the defined file pattern or file prefix in the `engine_attribute` settings for the initial load. If you want to change the files in the `engine_attribute` settings for subsequent loads, see how to Add or Remove Files Using Selecting Load (supported as of MySQL 9.1.2).

To run Lakehouse Incremental Load, load the data using Lakehouse Auto Parallel Load and set the Auto Parallel Load `refresh_external_tables` option to `true`.

If a subsequent call to Auto Parallel Load includes tables that are not yet loaded, then Auto Parallel Load loads them for the first time.

A call to Auto Parallel Load might contain both loaded and unloaded tables. Those that are unloaded are loaded, and those that are already loaded are refreshed.

Perform Initial Load

The following example performs an initial automatic load of three files in an Object Storage folder, and then does a subsequent refresh of the data in the external table by loading an additional two files into the table. The example uses resource principals to load the data.

1. Upload the files to load into Object Storage. This example creates the Object Storage folder `data_files` and uploads the files `data_file_1.csv`, `data_file_2.csv`, and `data_file_3.csv`.

2. Connect to your MySQL HeatWave Database System.

3. Create a session variable with the characteristics of the `input_list` to create the external table and specify the files to load into the table. To review all syntax options, see HEATWAVE_LOAD.

```
mysql> SET @input_list = '[{
  "db_name": "data_tables",
  "tables": [{
    "table_name": "data_table_1",
    "engine_attribute": {
      "dialect": {"format": "csv", "has_header": true},
      "file": [{"uri": "oci://mybucket@mynamespace/data_files/"}]
    }
  }]
}]';
```

   Where:

   - `@input_list` is the name of the session variable.
   - `db_name` identifies the database name to store the table: `data_tables`. The database is automatically created if it does not exist.
   - `table_name` sets the table name to store the data: `data_table_1`. The table is automatically created if it does not exist.
   - `engine_attribute` defines the parameters of the specified files.
   - `format` defines the format of the external file: `csv`.
   - `has_header` identifies a header in the external file. Auto Parallel Load then infers the column names from the first row in the file.
   - `file` defines the uniform resource identifiers (URI) to access the files. Replace these values with your own. The `data_files/` prefix defines the Object Storage folder storing the files to load.

   If you are on MySQL 9.1.2 and earlier, you need to update `dialect` with the `field delimiter` and `record delimiter` parameters. As of MySQL 9.2.0, MySQL HeatWave Lakehouse can automatically detect these values. See Lakehouse External Table Syntax to learn more.

4. Run the HEATWAVE_LOAD command to create the external table and load the data into the table. If you set the options to NULL, the data loads in normal mode by default.

```
mysql> CALL sys.HEATWAVE_LOAD(CAST(@input_list AS JSON), NULL);
```

   Review the SCHEMA_CREATION, TABLE_LOAD, and LOAD_SUMMARY sections for any errors or warnings during the load, and to confirm the external table loaded successfully.

5. Optionally, use the new database and query the number of rows in the table to confirm data from all files loaded successfully.

```
mysql> USE data_tables;
mysql> SELECT COUNT(*) FROM data_table_1;
+----------+
| COUNT(*) |
+----------+
|     1384 |
+----------+
```

Refresh Data

After doing the initial load of data, this example refreshes the data in the table data_table_1 with the loading of additional files into the table.

1. Upload the new files to load into Object Storage. This example uploads two additional files to the same data_files Object Storage folder: data_file_4.csv and data_file_5.csv.

2. Create a new session variable with the input_list to set up the incremental load. The configuration must not have any engine_attribute parameters.

```
mysql> SET @input_list = '[{"db_name": "data_tables", "tables": [{"table_name": "data_table_1"}]}]';
```

3. Create an options variable that includes the options to load the data in normal mode and enable refresh_external_tables.

```
mysql> SET @options = JSON_OBJECT('mode', 'normal','refresh_external_tables', TRUE);
```

4. Run the HEATWAVE_LOAD command with the input_list and options variables to refresh the data. Tables that are identified to be refreshed are marked with a + symbol. If the files for the table have not changed, then no changes are made to the table. The estimates in the output do not cover the tables that are refreshed.

```
mysql> CALL sys.HEATWAVE_LOAD(CAST(@input_list AS JSON), @options);
+-----------------------------------------+
| INITIALIZING HEATWAVE AUTO PARALLEL LOAD |
+-----------------------------------------+
| Version: 4.34                           |
|                                         |
| Load Mode: normal                       |
| Load Policy: disable_unsupported_columns |
| Output Mode: normal                     |
|                                         |
+-----------------------------------------+

+-----------------------------------------------------------------------------------------------------------------+
| LAKEHOUSE AUTO SCHEMA INFERENCE                                                                                 |
+-----------------------------------------------------------------------------------------------------------------+
| Verifying external lakehouse tables: 1                                                                          |
|                                                                                                                 |
| SCHEMA                   TABLE                    TABLE IS           RAW     NUM. OF      ESTIMATED           |
| NAME                     NAME                     CREATED      FILE SIZE     COLUMNS      ROW COUNT           |
| ------                   -----                    --------     ---------     -------      ---------           |
| `data_tables`            `data_table_1`           YES +        40.06 KiB           8         1.38 K           |
|                                                                                                                 |
| New schemas to be created: 0                                                                                    |
| External lakehouse tables to be created: 0                                                                      |
| External lakehouse tables to be refreshed (marked with +): 1                                                    |
|                                                                                                                 |
```

```
+-----------------------------------------------------------------------------------------------------------

+-----------------------------------------------------------------------+
| OFFLOAD ANALYSIS                                                      |
+-----------------------------------------------------------------------+
| Verifying input schemas: 1                                            |
| User excluded items: 0                                                |
|                                                                       |
| SCHEMA                       OFFLOADABLE    OFFLOADABLE     SUMMARY OF |
| NAME                              TABLES        COLUMNS     ISSUES     |
| ------                       -----------    -----------     ---------- |
| `data_tables`                          1              8                |
|                                                                       |
| Total offloadable schemas: 1                                          |
|                                                                       |
+-----------------------------------------------------------------------+

+-----------------------------------------------------------------------------------------------------------
| CAPACITY ESTIMATION
+-----------------------------------------------------------------------------------------------------------
| Default encoding for string columns: VARLEN (unless specified in the schema)
| Estimating memory footprint for 1 schema(s)
| Estimates do not include 1 external lakehouse table(s) being refreshed.
|
|                                TOTAL       ESTIMATED       ESTIMATED       TOTAL     DICTIONARY
| SCHEMA                   OFFLOADABLE   HEATWAVE NODE      MYSQL NODE      STRING        ENCODED
| NAME                          TABLES       FOOTPRINT       FOOTPRINT     COLUMNS        COLUMNS
| ------                   -----------       ---------       ---------     -------     ----------
| `data_tables`                      1         0 bytes         0 bytes           3              0
|
| Sufficient MySQL host memory available to load all tables.
| Sufficient HeatWave cluster memory available to load all tables.
|
+-----------------------------------------------------------------------------------------------------------

+-----------------------------------------------------------------------------------------------------------
| EXECUTING LOAD SCRIPT
+-----------------------------------------------------------------------------------------------------------
| HeatWave Load script generated
|   Retrieve load script containing 3 generated DDL command(s) using the query below:
| Deprecation Notice: "heatwave_load_report" will be deprecated, please switch to "heatwave_autopilo
|   SELECT log->>"$.sql" AS "Load Script" FROM sys.heatwave_autopilot_report WHERE type = "sql" ORDE
|
| Adjusting load parallelism dynamically per internal/external table.
| Using current parallelism of 32 thread(s) as maximum for internal tables.
|
| Warning: Executing the generated script may alter column definitions and secondary engine flags in
|
| Using SQL_MODE: ONLY_FULL_GROUP_BY,STRICT_TRANS_TABLES,NO_ZERO_IN_DATE,NO_ZERO_DATE,ERROR_FOR_DIVI
|
| Proceeding to load 1 table(s) into HeatWave.
|
| Applying changes will take approximately 26.00 ms
|
+-----------------------------------------------------------------------------------------------------------

+----------------------------------------------+
| TABLE LOAD                                   |
+----------------------------------------------+
| TABLE (1 of 1): `data_tables`.`data_table_1` |
| Commands executed successfully: 3 of 3       |
| Warnings encountered: 1                      |
| Table load succeeded!                        |
|   Total columns loaded: 8                    |
|   Elapsed time: 3.66 s                       |
|                                              |
+----------------------------------------------+

+-------------------------------------------------------------------------------+
| LOAD SUMMARY                                                                  |
+-------------------------------------------------------------------------------+
|                                                                               |
```

```
| SCHEMA                          TABLES       TABLES      COLUMNS         LOAD    |
| NAME                            LOADED       FAILED       LOADED     DURATION    |
| ------                          ------       ------      -------     --------    |
| `data_tables`                        1            0            8       3.66 s    |
|                                                                                    |
| Total errors encountered: 0                                                        |
| Total warnings encountered: 2                                                      |
| Retrieve the associated logs from the report table using the query below:          |
|   SELECT log FROM sys.heatwave_autopilot_report WHERE type IN ('error', 'warn');   |
|                                                                                    |
+------------------------------------------------------------------------------------+
```

5. Optionally, query the number of rows in the table again to confirm the data from the two new files successfully loaded.

```
mysql> SELECT COUNT(*) FROM data_table_1;
+----------+
| COUNT(*) |
+----------+
|     4130 |
+----------+
```

What's Next

- Learn more about the following:
  - Accelerate Query Processing
  - Train and Use Machine Learning Models
  - Perform AI-Powered Search and Content Generation

4.3.14.3 Add or Remove Files Using Selective Load

Available as of MySQL 9.1.2, you can update external tables using Selective Load, which lets you edit the list of files that comprise an external table without altering the directories in the Object Storage bucket.

This topic contains the following sections:

- Before You Begin
- Selective Load Overview
- Use the `ALTER TABLE` Statement
- Use Auto Parallel Load
- What's Next

Before You Begin

- Load structured data using Lakehouse Auto Parallel Load or manually.

Selective Load Overview

To edit the list of files comprising an external table, you can add or remove the file paths specified in the `ENGINE_ATTRIBUTE` option for the table. If a new file path is specified, then the data from the file is loaded into the table. If an existing file path is removed, then the data pertaining to the file is removed from the table.

Note the following points about the `ENGINE_ATTRIBUTE` option:

- Ensure that the `ENGINE_ATTRIBUTE` option specifies all the files to be loaded.

- You can modify only the file parameters in the `ENGINE_ATTRIBUTE` option.
- If Lakehouse automatically detected dialect parameters, such as delimiters, you must include those parameters in subsequent loads using Selective Load.
- If the data refresh fails, the value of the `ENGINE_ATTRIBUTE` option is not updated, and an error message displays.

You can edit the `ENGINE_ATTRIBUTE` option using one of the following methods:

- Use the `ALTER TABLE` Statement
- Use Auto Parallel Load

For more information about the `ENGINE_ATTRIBUTE` option, see Lakehouse External Table Syntax.

Use the `ALTER TABLE` Statement

You can use the `ENGINE_ATTRIBUTE` option with the `ALTER TABLE` statement to add or remove data in an external table without reloading the entire table. Lakehouse applies the changes incrementally.

The following example performs an initial manual load of five files in an Object Storage folder, and then does a subsequent load of data in the external table that removes some of the files from the initial load. The example uses resource principals to load the data.

Complete the following steps to perform the initial load of data:

1. Upload the files to load into Object Storage. See Upload the files to load into Object Storage Object to a Bucket. This example creates the Object Storage folder `data_files` and uploads the files `data_file_1.csv`, `data_file_2.csv`, `data_file_3.csv`, `data_file_4.csv`, and `data_file_5.csv`.
2. Connect to your MySQL HeatWave Database System.
3. In the terminal window, create and use the database to store the table.

   ```
   mysql> CREATE DATABASE data_tables;
   mysql> USE DATABASE data_tables;
   ```

4. Set up the `CREATE TABLE` statement and the `ENGINE_ATTRIBUTE` options to specify the parameters needed to process the external files. As of MySQL 9.4.0, you can use the `CREATE EXTERNAL TABLE` statement. In versions earlier than MySQL 9.4.0, you must use the `CREATE TABLE` statement and set `ENGINE` to `lakehouse`, and `SECONDARY_ENGINE` to `rapid`. See CREATE TABLE Statement.

   Ensure that the table has the correct data type for each column. For this example, columns are defined according to the data in the `bank.csv` file. See the following to learn more:

   - CREATE TABLE
   - Data Types
   - Lakehouse External Table Syntax

   As of MySQL 9.4.0, you can use SQL syntax to set options for external tables. For earlier versions, you must set options using JSON syntax. To learn more, see Lakehouse External Table JSON Syntax and Lakehouse External Table SQL Syntax.

   The following example uses JSON syntax:

   ```
   mysql> CREATE EXTERNAL TABLE data_table_1(
     col_1 varchar(2) DEFAULT NULL,
     col_2 smallint unsigned DEFAULT NULL,
     col_3 varchar(4) DEFAULT NULL,
   ```

```
  col_4 varchar(4) DEFAULT NULL,
  col_5 tinyint unsigned DEFAULT NULL,
  col_6 smallint unsigned DEFAULT NULL,
  col_7 smallint unsigned DEFAULT NULL,
  col_8 tinyint unsigned DEFAULT NULL
)
ENGINE_ATTRIBUTE='{"dialect": {"format": "csv", "has_header": true},
                   "file": [{"uri": "oci://bucket_1@tenant1/data_files/"}]}';
```

Where:

- The `CREATE EXTERNAL TABLE` statement creates the table `data_table_1`, and automatically sets `ENGINE` to `lakehouse`, and `SECONDARY_ENGINE` to `rapid`.
- Each column for the table is defined according to the files in the `data_files` folder.
- `format` defines the format of the external file: `csv`.
- `has_header` identifies a header in the external file.
- `file` parameters define the resource principals to access the files to create the external table.

The following example uses SQL syntax:

```
mysql> CREATE EXTERNAL TABLE data_table_1(
       col_1 varchar(2) DEFAULT NULL,
       col_2 smallint unsigned DEFAULT NULL,
       col_3 varchar(4) DEFAULT NULL,
       col_4 varchar(4) DEFAULT NULL,
       col_5 tinyint unsigned DEFAULT NULL,
       col_6 smallint unsigned DEFAULT NULL,
       col_7 smallint unsigned DEFAULT NULL,
       col_8 tinyint unsigned DEFAULT NULL
)
       FILE_FORMAT = (FORMAT csv HEADER ON)
       FILES = (URI = 'oci://bucket_1@tenant1/data_files/');
```

If you are on MySQL 9.1.2 and earlier, you need to update `dialect` with the `field delimiter` and `record delimiter` parameters. As of MySQL 9.2.0, MySQL HeatWave Lakehouse can automatically detect these values. See Lakehouse External Table Syntax to learn more.

5. Specify the `SECONDARY_LOAD` clause in an `ALTER TABLE` statement to load the data into the external table.

```
mysql> ALTER TABLE data_table_1 SECONDARY_LOAD;
Warning (code 3877): Command executed during preprocessing: 'ALTER TABLE `data_tables`.`data_table_1`
ENGINE_ATTRIBUTE='{"file": [{"uri": "oci://bucket_1@tenant1/data_files/"}],
"dialect": {"format": "csv", "has_header": true, "field_delimiter": ",", "record_delimiter": "\\n"}}''.
```

   Refer to the `dialect` parameters that need to be used in subsequent loads. Lakehouse automatically detected the `field_delimiter` and `record delimiter` values. If you need to review `dialect` parameters later on, you can use the following command: `SHOW CREATE TABLE table_name;`.

6. Optionally, query the number of rows in the table to confirm all files loaded successfully.

```
mysql> SELECT COUNT(*) FROM data_table_1;
+----------+
| count(*) |
+----------+
|     4130 |
+----------+
```

For this example, you now want to remove `data_file_1.csv`, `data_file_2.csv`, and `data_file_3.csv`. You want to keep `data_file_4.csv` and `data_file_5.csv`.

Complete the following steps to perform the Selective Load and remove two files from the table:

1. Update the `ENGINE_ATTRIBUTE` parameters for the table, which updates the files for the table to only include `data_file_4.csv` and `data_file_5.csv`. Make sure all `dialect` parameters are included.

```
mysql> ALTER TABLE data_table_1
ENGINE_ATTRIBUTE='{"dialect": {"format": "csv", "has_header": true, "field_delimiter": ",", "record_
                  "file": [{"uri": "oci://bucket_1@tenant1/data_files/data_file_4.csv"},
                           {"uri": "oci://bucket_1@tenant1/data_files/data_file_5.csv"}
]}';
```

2. Optionally, query the number of rows in the table to confirm the appropriate files loaded successfully.

```
mysql> SELECT COUNT(*) FROM data_table_1;
+----------+
| count(*) |
+----------+
|     2746 |
+----------+
```

Use Auto Parallel Load

You can use Auto Parallel Load to edit the value of the `engine_attribute` option in the following scenarios:

- If you create a table without the `engine_attribute` option set, you can set the `engine_attribute` option using Auto Parallel Load. Auto Parallel Load then runs the `ALTER TABLE` statement and loads the data from the files you specified into the table.

  This method also works on empty DB System tables, which are later converted to external tables.

- If you create a table with the `engine_attribute` set, but the table is unloaded, you can set the `engine_attribute` option using Auto Parallel Load. Auto Parallel Load uses the new value of the `engine_attribute` option to run the `ALTER TABLE` statement and loads data only from the files you specified into the table.

- If a table with the `engine_attribute` option set is already loaded, then Auto Parallel Load lets you edit the `engine_attribute` option value only if Lakehouse Incremental Load is enabled. To enable Lakehouse Incremental Load, you must set the `refresh_external_tables` option to `TRUE`. Auto Parallel Load then runs the `ALTER TABLE` statement and adds or removes data as defined in the new file paths. Lakehouse applies the changes incrementally.

The following example performs an initial automatic load of five files in an Object Storage folder, and then does a subsequent load of data in the external table that removes some of the files from the initial load. The example uses resource principals to load the data.

Complete the following steps to perform the initial load of data:

1. Upload the files to load into Object Storage. This example creates the Object Storage folder `data_files` and uploads the files `data_file_1.csv`, `data_file_2.csv`, `data_file_3.csv`, `data_file_4.csv`, and `data_file_5.csv`.

2. Connect to your MySQL HeatWave Database System.

3. Create a session variable with the characteristics of the `input_list` to create the external table and specify the files to load into the table. To review all syntax options, see HEATWAVE_LOAD.

```
mysql> SET @input_list = '[{
  "db_name": "data_tables",
  "tables": [{
    "table_name": "data_table_1",
    "engine_attribute": {
      "dialect": {"format": "csv", "has_header": true},
      "file": [{"uri": "oci://bucket_1@tenant1/data_files/"}]
    }
  }]
```

```
}]';
```

Where:

- `@input_list` is the name of the session variable.
- `db_name` identifies the database name to store the table: `data_tables`. The database is automatically created if it does not exist.
- `table_name` sets the table name to store the data: `data_table_1`. The table is automatically created if it does not exist.
- `engine_attribute` defines the parameters of the specified files.
- `format` defines the format of the external file: `csv`.
- `has_header` identifies a header in the external file. Auto Parallel Load then infers the column names from the first row in the file.
- `file` defines the resource principals to access the files. Replace these values with your own. The `prefix` parameter defines the Object Storage folder storing the files to load.

If you are on MySQL 9.1.2 and earlier, you need to update `dialect` with the `field delimiter` and `record delimiter` parameters. As of MySQL 9.2.0, MySQL HeatWave Lakehouse can automatically detect these values. See Lakehouse External Table Syntax to learn more.

4. Run the `HEATWAVE_LOAD` command create the external table and load data into the table. If you set the options to `NULL`, the data loads in normal mode by default.

```
mysql> CALL sys.HEATWAVE_LOAD(CAST(@input_list AS JSON), NULL);
+-----------------------------------------+
| INITIALIZING HEATWAVE AUTO PARALLEL LOAD |
+-----------------------------------------+
| Version: 4.34                           |
|                                         |
| Load Mode: normal                       |
| Load Policy: disable_unsupported_columns |
| Output Mode: normal                     |
|                                         |
+-----------------------------------------+

+-----------------------------------------------------------------------------------------------------------
| LAKEHOUSE AUTO SCHEMA INFERENCE
+-----------------------------------------------------------------------------------------------------------
| Verifying external lakehouse tables: 1
|
| SCHEMA                   TABLE                    TABLE IS           RAW     NUM. OF      ESTIMATED
| NAME                     NAME                     CREATED      FILE SIZE     COLUMNS      ROW COUNT
| ------                   -----                    --------     ---------     -------      ---------
| `data_tables`            `data_table_1`           NO           40.06 KiB           8         1.38 K
|
| New schemas to be created: 1
| External lakehouse tables to be created: 1
|
+-----------------------------------------------------------------------------------------------------------

+-------------------------------------------------------------------+
| OFFLOAD ANALYSIS                                                  |
+-------------------------------------------------------------------+
| Verifying input schemas: 1                                        |
| User excluded items: 0                                            |
|                                                                   |
| SCHEMA                        OFFLOADABLE    OFFLOADABLE     SUMMARY OF |
| NAME                               TABLES        COLUMNS     ISSUES     |
| ------                        -----------    -----------     ---------- |
| `data_tables`                           1              8                |
|                                                                   |
| Total offloadable schemas: 1                                      |
```

```
|                                                                    |
+--------------------------------------------------------------------+

+-----------------------------------------------------------------------------------------------------
| CAPACITY ESTIMATION
+-----------------------------------------------------------------------------------------------------
| Default encoding for string columns: VARLEN (unless specified in the schema)
| Estimating memory footprint for 1 schema(s)
|
|                               TOTAL       ESTIMATED       ESTIMATED       TOTAL     DICTIONARY
| SCHEMA                  OFFLOADABLE   HEATWAVE NODE      MYSQL NODE      STRING        ENCODED
| NAME                         TABLES       FOOTPRINT       FOOTPRINT     COLUMNS        COLUMNS
| ------                  -----------       ---------       ---------     -------     ----------
| `data_tables`                     1        4.83 MiB      576.00 KiB           3              0
|
| Sufficient MySQL host memory available to load all tables.
| Sufficient HeatWave cluster memory available to load all tables.
|
+-----------------------------------------------------------------------------------------------------

+-----------------------------------------------------------------------------------------------------
| EXECUTING LOAD SCRIPT
+-----------------------------------------------------------------------------------------------------
| HeatWave Load script generated
|   Retrieve load script containing 3 generated DDL command(s) using the query below:
| Deprecation Notice: "heatwave_load_report" will be deprecated, please switch to "heatwave_autopilo
|   SELECT log->>"$.sql" AS "Load Script" FROM sys.heatwave_autopilot_report WHERE type = "sql" ORDE
|
| Adjusting load parallelism dynamically per internal/external table.
| Using current parallelism of 32 thread(s) as maximum for internal tables.
|
| Warning: Executing the generated script may alter column definitions and secondary engine flags in
|
| Using SQL_MODE: ONLY_FULL_GROUP_BY,STRICT_TRANS_TABLES,NO_ZERO_IN_DATE,NO_ZERO_DATE,ERROR_FOR_DIVI
|
| Proceeding to load 1 table(s) into HeatWave.
|
| Applying changes will take approximately 7.03 s
|
+-----------------------------------------------------------------------------------------------------

+-----------------------------------------+
| SCHEMA CREATION                         |
+-----------------------------------------+
| Schema `data_tables` creation succeeded! |
| Warnings/errors encountered: 0          |
| Elapsed time: 5.47 ms                   |
|                                         |
+-----------------------------------------+

+-----------------------------------------------+
| TABLE LOAD                                    |
+-----------------------------------------------+
| TABLE (1 of 1): `data_tables`.`data_table_1` |
| Commands executed successfully: 2 of 2        |
| Warnings encountered: 1                       |
| Table load succeeded!                         |
|   Total columns loaded: 8                     |
|   Elapsed time: 3.93 s                        |
|                                               |
+-----------------------------------------------+

+-------------------------------------------------------------------------------+
| LOAD SUMMARY                                                                  |
+-------------------------------------------------------------------------------+
|                                                                               |
| SCHEMA                          TABLES       TABLES      COLUMNS         LOAD    |
| NAME                            LOADED       FAILED       LOADED     DURATION    |
| ------                          ------       ------      -------     --------    |
| `data_tables`                        1            0            8       3.93 s    |
|                                                                               |
| Total errors encountered: 0                                                   |
```

```
| Total warnings encountered: 4                                                    |
| Retrieve the associated logs from the report table using the query below:        |
|   SELECT log FROM sys.heatwave_autopilot_report WHERE type IN ('error', 'warn'); |
|                                                                                  |
+----------------------------------------------------------------------------------+
```

Review the `SCHEMA CREATION`, `TABLE LOAD`, and `LOAD SUMMARY` sections for any errors or warnings during the load, and to confirm the external table loaded successfully.

5. Optionally, use the new database and query the number of rows in the table to confirm data from all files loaded successfully.

```
mysql> USE data_tables;
mysql> SELECT COUNT(*) FROM data_table_1;
+----------+
| COUNT(*) |
+----------+
|     4130 |
+----------+
```

For this example, you now want to remove `data_file_1.csv`, `data_file_2.csv`, and `data_file_3.csv`. You want to keep `data_file_4.csv` and `data_file_5.csv`.

Complete the following steps to perform the Selective Load and remove two files from the table:

1. Run the `SHOW CREATE TABLE` command to confirm all the `dialect` parameters for the loaded table.

```
mysql> SHOW CREATE TABLE data_tables.data_table_1;
+--------------+-----------------------------------------------------------------------------------------------------------
| Table        | Create Table
+--------------+-----------------------------------------------------------------------------------------------------------
| data_table_1 | CREATE TABLE `data_table_1` (
  `Airline` varchar(2) DEFAULT NULL COMMENT 'RAPID_COLUMN=ENCODING=VARLEN',
  `Flight` smallint unsigned DEFAULT NULL,
  `AirportFrom` varchar(4) DEFAULT NULL COMMENT 'RAPID_COLUMN=ENCODING=VARLEN',
  `AirportTo` varchar(4) DEFAULT NULL COMMENT 'RAPID_COLUMN=ENCODING=VARLEN',
  `DayOfWeek` tinyint unsigned DEFAULT NULL,
  `Time` smallint unsigned DEFAULT NULL,
  `Length` smallint unsigned DEFAULT NULL,
  `Delay` tinyint unsigned DEFAULT NULL
) ENGINE=Lakehouse DEFAULT CHARSET=utf8mb4 COLLATE=utf8mb4_0900_ai_ci SECONDARY_ENGINE=RAPID /*!80021
ENGINE_ATTRIBUTE='{"file": [{"bucket": "bucket_1", "prefix": "data_files/", "region": "us-ashburn-1", "
"dialect": {"format": "csv", "has_header": true, "field_delimiter": ",", "record_delimiter": "\\n"}}' *
+--------------+-----------------------------------------------------------------------------------------------------------
```

Refer to all the `dialect` parameters and ensure to include them in the subsequent load command. For this example, the dialect parameters are: `"format": "csv", "has_header": true, "field_delimiter": ",", "record_delimiter": "\\n"`.

2. Create a new `input_list` variable with the updated files for the table. Update the `engine_attribute` parameters, which updates the files for the table to only include `data_file_4.csv` and `data_file_5.csv`. Make sure all `dialect` parameters are included.

```
mysql> SET @input_list = '[{
  "db_name": "data_tables",
  "tables": [{
    "table_name": "data_table_1",
    "engine_attribute": {
      "dialect": {
        "format": "csv",
        "has_header": true,
        "field_delimiter": ",",
        "record_delimiter": "\\n"},
      "file": [{"uri": "oci://bucket_1@tenant1/data_files/data_file_4.csv"},
               {"uri": "oci://bucket_1@tenant1/data_files/data_file_5.csv"}]
    }
  }]
}]';
```

3. Create an `options` variable that includes the options to load the data in `normal` mode and enable `refresh_external_tables`.

```
mysql> SET @options = JSON_OBJECT('mode', 'normal','refresh_external_tables', TRUE);
```

4. Run the `HEATWAVE_LOAD` command with the `input_list` and `options` variables.

```
mysql> CALL sys.HEATWAVE_LOAD(CAST(@input_list AS JSON), @options);
+-----------------------------------------+
| INITIALIZING HEATWAVE AUTO PARALLEL LOAD |
+-----------------------------------------+
| Version: 4.34                           |
|                                         |
| Load Mode: normal                       |
| Load Policy: disable_unsupported_columns |
| Output Mode: normal                     |
|                                         |
+-----------------------------------------+

+-----------------------------------------------------------------------------------------------------------
| LAKEHOUSE AUTO SCHEMA INFERENCE
+-----------------------------------------------------------------------------------------------------------
| Verifying external lakehouse tables: 1
|
| SCHEMA                   TABLE                    TABLE IS           RAW     NUM. OF      ESTIMATE
| NAME                     NAME                     CREATED      FILE SIZE     COLUMNS      ROW COUN
| ------                   -----                    --------     ---------     -------      --------
| `data_tables`            `data_table_1`           YES +        40.06 KiB           8         1.38
|
| New schemas to be created: 0
| External lakehouse tables to be created: 0
| External lakehouse tables to be refreshed (marked with +): 1
|
+-----------------------------------------------------------------------------------------------------------

+-------------------------------------------------------------------------+
| OFFLOAD ANALYSIS                                                        |
+-------------------------------------------------------------------------+
| Verifying input schemas: 1                                              |
| User excluded items: 0                                                  |
|                                                                         |
| SCHEMA                        OFFLOADABLE    OFFLOADABLE     SUMMARY OF |
| NAME                               TABLES        COLUMNS     ISSUES     |
| ------                        -----------    -----------     ---------- |
| `data_tables`                           1              8                |
|                                                                         |
| Total offloadable schemas: 1                                            |
|                                                                         |
+-------------------------------------------------------------------------+

+-----------------------------------------------------------------------------------------------------------
| CAPACITY ESTIMATION
+-----------------------------------------------------------------------------------------------------------
| Default encoding for string columns: VARLEN (unless specified in the schema)
| Estimating memory footprint for 1 schema(s)
| Estimates do not include 1 external lakehouse table(s) being refreshed.
|
|                                TOTAL       ESTIMATED       ESTIMATED       TOTAL     DICTIONARY
| SCHEMA                    OFFLOADABLE   HEATWAVE NODE      MYSQL NODE      STRING        ENCODED
| NAME                           TABLES       FOOTPRINT       FOOTPRINT     COLUMNS        COLUMNS
| ------                    -----------       ---------       ---------     -------     ----------
| `data_tables`                       1         0 bytes         0 bytes           3             0
|
| Sufficient MySQL host memory available to load all tables.
| Sufficient HeatWave cluster memory available to load all tables.
|
+-----------------------------------------------------------------------------------------------------------

+-----------------------------------------------------------------------------------------------------------
| EXECUTING LOAD SCRIPT
+-----------------------------------------------------------------------------------------------------------
| HeatWave Load script generated
```

```
|   Retrieve load script containing 1 generated DDL command(s) using the query below:
| Deprecation Notice: "heatwave_load_report" will be deprecated, please switch to "heatwave_autopilot_r
|   SELECT log->>"$.sql" AS "Load Script" FROM sys.heatwave_autopilot_report WHERE type = "sql" ORDER B
|
| Adjusting load parallelism dynamically per internal/external table.
| Using current parallelism of 32 thread(s) as maximum for internal tables.
|
| Warning: Executing the generated script may alter column definitions and secondary engine flags in th
|
| Using SQL_MODE: ONLY_FULL_GROUP_BY,STRICT_TRANS_TABLES,NO_ZERO_IN_DATE,NO_ZERO_DATE,ERROR_FOR_DIVISIO
|
| Proceeding to load 1 table(s) into HeatWave.
|
| Applying changes will take approximately   0 ps
|
+-----------------------------------------------------------------------------------------------------

+---------------------------------------------+
| TABLE LOAD                                  |
+---------------------------------------------+
| TABLE (1 of 1): `data_tables`.`data_table_1` |
| Commands executed successfully: 1 of 1      |
| Warnings encountered: 0                     |
| Table load succeeded!                       |
|   Total columns loaded: 8                   |
|   Elapsed time: 3.09 s                      |
|                                             |
+---------------------------------------------+

+-------------------------------------------------------------------------------+
| LOAD SUMMARY                                                                  |
+-------------------------------------------------------------------------------+
|                                                                               |
| SCHEMA                          TABLES       TABLES      COLUMNS         LOAD    |
| NAME                            LOADED       FAILED       LOADED     DURATION    |
| ------                          ------       ------      -------     --------    |
| `data_tables`                        1            0            8       3.09 s    |
|                                                                               |
| Total errors encountered: 0                                                   |
| Total warnings encountered: 1                                                 |
| Retrieve the associated logs from the report table using the query below:     |
|   SELECT log FROM sys.heatwave_autopilot_report WHERE type IN ('error', 'warn'); |
|                                                                               |
+-------------------------------------------------------------------------------+
```

5. Optionally, query the number of rows in the table again to confirm the data from the appropriate files successfully loaded.

```
mysql> SELECT COUNT(*) FROM data_table_1;
+----------+
| count(*) |
+----------+
|     2746 |
+----------+
```

What's Next

- Learn more about the following:
  - Accelerate Query Processing
  - Train and Use Machine Learning Models
  - Perform AI-Powered Search and Content Generation

4.4 Unload Data from MySQL HeatWave Cluster

The following topics describe how to unload data.

4.4.1 Unload Data Automatically

You can use MySQL HeatWave Auto Unload to unload data automatically.

This topic contains the following sections:

- Before You Begin
- Overview for Unloading Data Automatically
- Unload Specific Tables
- Unload All Data
- Auto Unload Examples
- What's Next

Before You Begin

- Load structured data using Lakehouse Auto Parallel Load or manually.
- Review the syntax for Auto Unload.

Overview for Unloading Data Automatically

Unloading a table from MySQL HeatWave may be necessary to replace an existing table, to reload a table, to free up memory, or simply to remove a table that is no longer used.

Auto Unload facilitates the process of unloading data from MySQL HeatWave by automating many of the steps involved, including:

- Excluding schemas and tables that cannot be unloaded.
- Removes the secondary engine flag for tables that are to be unloaded.
- Unloading data from MySQL HeatWave.
- Unloading of automatically loaded tables if there is not enough memory to manually load a table. See Automatic Loading and Unloading of DB System Tables and Partitions.

Auto Unload unloads tables in sequence, ordered by schema and table name. It reports unload-time errors as they are encountered. If there is an error while unloading a table, Auto Unload does not terminate the operation. Auto Unload continues running, and moves on to the next table.

When Auto Unload finishes running, it checks if tables are unloaded and shows a summary with the number of tables that were unloaded and the number of tables that failed to unload.

Data is removed from MySQL HeatWave only. The table contents on the DB System are not affected.

To view command-line help in the terminal, run the following command:

```
mysql> CALL sys.heatwave_unload(JSON_ARRAY(),JSON_OBJECT("output","help"));
```

Unload Specific Tables

You can use the `heatwave_unload` command to unload specific loaded tables in the DB System.

The following examples use the Auto Unload Syntax for MySQL 8.4.0 and Later. If you are on an earlier version than MySQL 8.4.0, use the Auto Unload Syntax for MySQL 8.3.0-u2 and Earlier.

To unload specific tables:

1. Run `heatwave_unload` with the appropriate parameters to unload the desired tables.

   ```
   mysql> CALL sys.heatwave_unload (input_list,[options]);
   ```

Replace `input_list` and `[options]` with your own values. Review the syntax for Auto Unload.

The following example unloads a single table in `dryrun` mode. You can review the output before running the command in `normal` mode.

```
mysql> CALL sys.heatwave_unload(JSON_ARRAY(JSON_OBJECT("db_name", "data_tables", "tables", JSON_ARRAY("
+-----------------------------------+
| INITIALIZING HEATWAVE AUTO UNLOAD |
+-----------------------------------+
|                                   |
| Unload Mode: dryrun               |
| Output Mode: normal               |
|                                   |
+-----------------------------------+

+-------------------------------------------------------------------------------------------------------
| HEATWAVE UNLOAD
+-------------------------------------------------------------------------------------------------------
|
|
| SCHEMA                     TABLES         TABLES                      MEMORY GAIN              MEMO
| NAME                    TO UNLOAD      TO REMAIN                  (HEATWAVE NODES)
| -------                 -----------    -----------    -------------------------------   --------------
| `data_tables`                   1              0                         68.24 KiB                 0
|
| TOTAL                           1              0                         68.24 KiB                 0
|
|  Memory consumed by base relations after unload: 7.94 MiB
|
+-------------------------------------------------------------------------------------------------------

+-------------------------------------------------------------------------------------------------------
| UNLOAD SCRIPT GENERATION
+-------------------------------------------------------------------------------------------------------
| Dryrun mode only generates the unload script
| Set mode to "normal" in options to unload tables
| Retrieve unload script containing 2 generated DDL commands using the query below:
|   SELECT log->>"$.sql" AS "Unload Script" FROM sys.heatwave_autopilot_report WHERE type = "sql" ORDER
|
|
| Caution: Executing the generated unload script will affect the secondary engine flags in the schema
| Caution: Executing the generated unload script will unload tables. Queries on these tables will
|          no longer be offloaded to HeatWave.
|
+-------------------------------------------------------------------------------------------------------
```

If Auto Unload fails with an error, run the following command to review the report.

```
mysql> SELECT log FROM sys.heatwave_autopilot_report WHERE type="error";
```

Run the following command to review warnings.

```
mysql> SELECT log FROM sys.heatwave_autopilot_report WHERE type="warn";
```

2. Optionally, run the following command to review the unload script:

```
mysql> SELECT log->>"$.sql" AS "Unload Script" FROM sys.heatwave_autopilot_report WHERE type = "sql" OR
+-------------------------------------------------------------+
| Unload Script                                               |
+-------------------------------------------------------------+
| ALTER TABLE `data_tables`.`data_table_1` SECONDARY_UNLOAD;  |
| ALTER TABLE `data_tables`.`data_table_1` SECONDARY_ENGINE=NULL; |
+-------------------------------------------------------------+
```

3. If you are satisifed with the output and unload script, reissue the command in `normal` mode.

   The following example runs the command previously run in `dryrun` mode again in `normal` mode. Alternatively, including no options with `NULL` also runs the command in `normal` mode.

```
mysql> CALL sys.heatwave_unload(JSON_ARRAY(JSON_OBJECT("db_name", "data_tables", "tables", JSON_ARRA
+-----------------------------------+
| INITIALIZING HEATWAVE AUTO UNLOAD |
+-----------------------------------+
|                                   |
| Unload Mode: normal               |
| Output Mode: normal               |
|                                   |
+-----------------------------------+

+-----------------------------------------------------------------------------------------------------
| HEATWAVE UNLOAD
+-----------------------------------------------------------------------------------------------------
|
|
| SCHEMA                      TABLES         TABLES                       MEMORY GAIN             M
| NAME                     TO UNLOAD      TO REMAIN                   (HEATWAVE NODES)
| -------                  -----------    -----------    -------------------------------   -----------
| `data_tables`                    1              0                          68.24 KiB
|
| TOTAL                            1              0                          68.24 KiB
|
|  Memory consumed by base relations after unload: 7.94 MiB
|
+-----------------------------------------------------------------------------------------------------

+-----------------------------------------------------------------------------------------------------
| EXECUTING UNLOAD
+-----------------------------------------------------------------------------------------------------
| HeatWave Unload script generated
|   Retrieve unload script containing 2 generated DDL commands using the query below:
|   SELECT log->>"$.sql" AS "Unload Script" FROM sys.heatwave_autopilot_report WHERE type = "sql" OR
|
|
| Proceeding to unload 1 tables from HeatWave
|
+-----------------------------------------------------------------------------------------------------

+----------------------------------------+
| UNLOAD EXECUTE                         |
+----------------------------------------+
| Commands executed successfully: 2 of 2 |
| Warnings encountered: 0                |
+----------------------------------------+
```

4. Optionally, run the following commands to retrieve DDL statements.

```
mysql> SET SESSION group_concat_max_len = 1000000;
mysql> SELECT GROUP_CONCAT(log->>"$.sql" SEPARATOR ' ')
        FROM sys.heatwave_autopilot_report
        WHERE type = "sql" ORDER BY id;
```

Unload All Data

You can use the `heatwave_unload_all` command to unload all tables in the DB System. You have the option to unload all user-loaded and system tables, or only user-loaded tables.

Run `heatwave_unload_all` with the appropriate options to unload all tables.

```
mysql> CALL sys.heatwave_unload_all ([options]);
```

Replace `[options]` with your own values. Review the Auto Unload Syntax to Unload All Tables.

The following example unloads all tables with default options. Only user-loaded tables are unloaded since the default value for `only_user_loaded_tables` is `true`.

```
mysql> CALL sys.heatwave_unload_all (NULL);
```

Auto Unload Examples

- Unload three specific tables from a single database with default options.

```
mysql> CALL sys.heatwave_unload(JSON_ARRAY(JSON_OBJECT("db_name", "db_1", "tables", JSON_ARRAY("data_tabl
```

  The tables `data_table_1`, `data_table_2`, and `data_table_3` are unloaded from the `db_1` database.

- Unload one table from two specific databases in `dryrun` mode.

```
mysql> CALL sys.heatwave_unload(JSON_ARRAY(
                                JSON_OBJECT("db_name", "db_1", "tables", JSON_ARRAY("data_table_1")),
                                JSON_OBJECT("db_name", "db_2", "tables", JSON_ARRAY("data_table_2"))),
                                JSON_OBJECT("mode", "dryrun"));
```

  The table `data_table_1` is unloaded from the `db_1` database, and the table `data_table_2` is unloaded from the `db_2` database.

- Unload all tables from the databases that have any tables with an `hw` prefix.

```
mysql> SELECT JSON_ARRAYAGG(schema_name)
        FROM performance_schema.rpd_table_id
        WHERE table_name LIKE 'hw%'
        INTO @input_list;
mysql> CALL sys.heatwave_unload(@input_list, NULL);
```

- Unload tables that begin with an “hw” prefix from a schema named `db_1`.

```
mysql> SELECT JSON_ARRAY(JSON_OBJECT('db_name', 'db_1', 'tables', JSON_ARRAYAGG(table_name)))
        FROM performance_schema.rpd_table_id
        WHERE schema_name = 'db_1'
        AND table_name LIKE 'hw%'
        INTO @input_list;
mysql> CALL sys.heatwave_unload(@input_list, NULL);
```

- Unload all user and system tables.

```
mysql> CALL sys.heatwave_unload_all (JSON_OBJECT("only_user_loaded_tables",false));
```

- Unload user tables with the `silent` option.

```
mysql> CALL sys.heatwave_unload_all (JSON_OBJECT("output","silent"));
```

- Unload all user and system tables with the `silent` option.

```
mysql> CALL sys.heatwave_unload_all (JSON_OBJECT("only_user_loaded_tables",false,"output","silent"));
```

- Call Auto Unload from a stored procedure:

```
DROP PROCEDURE IF EXISTS auto_unload_wrapper;
DELIMITER //
CREATE PROCEDURE auto_unload_wrapper()
BEGIN
-- AUTOMATED INPUT
SET @db_list = (SELECT JSON_ARRAYAGG(unique_schemas) FROM (SELECT DISTINCT(schema_name) as unique_schemas
SET @exc_list = (SELECT JSON_ARRAYAGG(CONCAT(schema_name,'.',table_name))
              FROM performance_schema.rpd_table_id
              WHERE table_name NOT LIKE 'hw%');
CALL sys.heatwave_unload(@db_list, JSON_OBJECT("output","silent","exclude_list",
CAST(@exc_list AS JSON)));

-- CUSTOM OUTPUT
SELECT log as 'Warnings' FROM sys.heatwave_autopilot_report WHERE type="warn";
SELECT Count(*) AS "Total Unload commands Generated"
FROM sys.heatwave_autopilot_report WHERE type = "sql" ORDER BY id;
```

```
END //
DELIMITER ;

CALL auto_unload_wrapper();
```

What's Next

- Learn more about the following:
  - Accelerate Query Processing
  - Train and Use Machine Learning Models
  - Perform AI-Powered Search and Content Generation

4.4.2 Unload Data Manually

This topic describes how to unload tables manually.

Before You Begin

- Load structured data using Lakehouse Auto Parallel Load or manually.

Unload Tables

Unloading a table from MySQL HeatWave may be necessary to replace an existing table, to reload a table, to free up memory, or simply to remove a table that is no longer used.

To unload a table manually, specify the `SECONDARY_UNLOAD` clause in an `ALTER TABLE` statement:

```
mysql> ALTER TABLE tbl_name SECONDARY_UNLOAD;
```

The following example unloads the `data_table_1` table.

```
mysql> ALTER TABLE data_table_1 SECONDARY_UNLOAD;
```

Data is removed from MySQL HeatWave only. The table contents on the DB System are not affected.

Unload Partitions

As of MySQL 9.1.0, MySQL HeatWave supports partitions for DB System tables. `INFORMATION_SCHEMA.PARTITIONS` includes a `SECONDARY_LOAD` column that indicates whether a partition is loaded.

See the following to learn more:

- Load Partitions
- Partition Selection Limitations

When an unload operation successfully unloads the last partition, it unloads the whole table.

To unload partitions, specify the `SECONDARY_UNLOAD` clause in an `ALTER TABLE` statement.

```
mysql> ALTER TABLE tbl_name SECONDARY_UNLOAD PARTITION (p0, p1, ..., pn);
```

The following example unloads the `p0` partition from `data_table_1`.

```
mysql> ALTER TABLE data_table_1 SECONDARY_UNLOAD PARTITION (p0);
```

If a partition unload fails, the table becomes stale.

What's Next

- Learn more about the following:
  - Accelerate Query Processing
  - Train and Use Machine Learning Models
  - Perform AI-Powered Search and Content Generation

4.5 Retrieve MySQL HeatWave Metadata

This section provides Information Schema and Performance Schema queries that you can use to retrieve MySQL HeatWave metadata.

Note

For queries that monitor MySQL HeatWave node status, memory usage, data loading, change propagation, and queries, see Section 5.10, “MySQL HeatWave Monitoring”.

4.5.1 View Secondary Engine Definitions

To identify tables in the DB System that are defined with a secondary engine, query the `CREATE_OPTIONS` column in the `INFORMATION_SCHEMA.TABLES` table. If the `CREATE_OPTIONS` column contains both `SECONDARY_ENGINE=RAPID` and `SECONDARY_LOAD=1`, the table is offloaded and loaded into the MySQL HeatWave Cluster

Before You Begin

- Ensure that the `enable_secondary_engine_statistics` system variables is set to `ON`.
- Check if you have the `SELECT` and `INSERT` privileges for the table that you want to analyze. If not, ask your admin user to grant you the required privileges.

Viewing Table Offload Status

Use the following `SELECT TABLE` to verify the offload status of tables in a specific schema:

```
mysql> SELECT TABLE_SCHEMA, TABLE_NAME, CREATE_OPTIONS
FROM INFORMATION_SCHEMA.TABLES
WHERE CREATE_OPTIONS LIKE '%SECONDARY_ENGINE%'
AND TABLE_SCHEMA LIKE 'your_schema_name';
```

If the table is not loaded to the DB System, you can load tables with `SECONDARY_ENGINE="RAPID"` into MySQL HeatWave either explicitly using `ALTER TABLE` with `SECONDARY_LOAD` or load using Lakehouse Auto Parallel Load using the MySQL HeatWave Autopilot subsystem.

```
mysql> CREATE TABLE airline_info (
    airline_id INT,
    flight_id INT,
    PRIMARY KEY (airline_id)
) SECONDARY_ENGINE = RAPID;
Query OK, 0 rows affected (0.025 sec)
mysql> SELECT TABLE_SCHEMA, TABLE_NAME, CREATE_OPTIONS FROM INFORMATION_SCHEMA.TABLES WHERE CREATE_OPTIONS

+--------------+------------------+--------------------------------------------+
| TABLE_SCHEMA | TABLE_NAME       | CREATE_OPTIONS                             |
+--------------+------------------+--------------------------------------------+
| airportdb    | airline_info     | SECONDARY_ENGINE="RAPID" SECONDARY_LOAD="0" |
| ...          |                  |                                            |
+--------------+------------------+--------------------------------------------+
mysql> ALTER TABLE airline_info SECONDARY_LOAD;
```

```
Query OK, 0 rows affected (3.9098 sec)
mysql> SELECT TABLE_SCHEMA, TABLE_NAME, CREATE_OPTIONS FROM INFORMATION_SCHEMA.TABLES WHERE CREATE_OPTI

+--------------+------------------+--------------------------------------------+
| TABLE_SCHEMA | TABLE_NAME       | CREATE_OPTIONS                             |
+--------------+------------------+--------------------------------------------+
| airportdb    | airline_info     | SECONDARY_ENGINE="RAPID" SECONDARY_LOAD="1" |
| ...          |                  |                                            |
+--------------+------------------+--------------------------------------------+
```

You can also view create options for an individual table using `SHOW CREATE TABLE`.

The `INFORMATION_SCHEMA.PARTITIONS` view includes a `SECONDARY_LOAD` column that indicates whether a partition has been loaded. The possible values are:

- `1`: The partition is loaded.
- `0`: The partition is not loaded.
- `NULL`: The table does not have partitions.

A successfully completed load or unload operation updates `INFORMATION_SCHEMA.PARTITIONS`. If an operation loads or unloads the whole table, and does not specify individual partitions, then it updates all the `SECONDARY_LOAD` columns when successfully completed.

MySQL HeatWave persists the partition information. During a restart, only the previously loaded partitions are loaded.

What's Next

- Learn to view the excluded columns.

4.5.2 List Excluded Columns

In MySQL HeatWave, you can explicitly exclude certain table columns from being loaded into the MySQL HeatWave Cluster, by defining them with the `NOT SECONDARY` attribute. These columns are typically not needed for analytic queries and are therefore omitted to optimize memory usage and performance.

Before You Begin

- Ensure that the `enable_secondary_engine_statistics` system variables is set to `ON`.
- Check if you have the `SELECT` and `INSERT` privileges for the table that you want to analyze. If not, ask your admin user to grant you the required privileges.

Viewing Excluded Columns

To identify table columns defined as `NOT SECONDARY` on the DB System, query the `EXTRA` column of the `INFORMATION_SCHEMA.COLUMNS` table. Columns defined with the `NOT SECONDARY` column attribute are not loaded into MySQL HeatWave when executing a table load operation.

```
mysql> SELECT COLUMN_NAME, EXTRA
          FROM INFORMATION_SCHEMA.COLUMNS
          WHERE TABLE_NAME LIKE 't1' AND EXTRA LIKE '%NOT SECONDARY%';
+-------------+---------------+
| COLUMN_NAME | EXTRA         |
+-------------+---------------+
| O_COMMENT   | NOT SECONDARY |
+-------------+---------------+
```

You can also view columns defined as `NOT SECONDARY` for an individual table using `SHOW CREATE TABLE`.

What's Next

- Workload Optimization for OLAP
- Workload Optimization for OLTP

4.6 Export Query Results to Object Storage

As of MySQL 9.1.1-u1, you can export your query results on DB System tables and external Lakehouse tables to Object Storage. This feature is available for MySQL HeatWave on OCI and MySQL HeatWave on AWS.

The capability to convert the output into different file formats and write it directly to object storage helps to seamlessly integrate MySQL HeatWave with various data pipelines, analytics workflows, and downstream applications.

After exporting query results successfully, the output displays the number of rows written and the folder created to save the export query result.

4.6.1 About Exporting Query Results

This section provides an overview of exporting query results to Object Storage.

Methods to Export Query Results

You can export query results in the following ways.

- Uniform Resource Identifier (URI): Supported as of MySQL 9.3.1. Specify the details to access OCI resources with a URI. This option is only available for MySQL HeatWave on OCI. See OCIFS Documentation to learn more about OCIFS file format and its specifications.
- Pre-Authenticated Request (PAR): Create a temporary URL that is the export destination for the query results. This option is only available for MySQL HeatWave on OCI. Learn more about Object Storage Pre-Authenticated Requests.
- Resource Principal: Define the details to access OCI resources and export query results. This option is only available for MySQL HeatWave on OCI. Learn more about Resource Principals.
- AWS S3 URL: For MySQL HeatWave on AWS, you must use AWS S3 URLs. See Manage AWS Access from the *MySQL HeatWave on AWS Service Guide* to learn more.

Supported File Formats

You can export the query results in the following file formats:

- CSV (As of MySQL 9.1.1-u1)

  See Exporting Query Results to CSV Files.

- Parquet (As of MySQL 9.1.1-u1)

  See Exporting Query Results to Parquet Files.

- Newline-Delimited JSON (As of MySQL 9.3.1)

  See Exporting Query Results to JSON Files.

What's Next

- Review the Requirements to Export Query Results.

4.6.2 Requirements to Export Query Results

Review the following requirements to export query results.

Required Privileges to Export Query Results

- Ask the admin user to grant you the EXPORT_QUERY_RESULTS privilege. Replace *user_name* and *database_name* in the following command with the appropriate user name and database name.

```
mysql>GRANT EXPORT_QUERY_RESULTS ON database_name.* TO 'user_name'@'%';
```

- The resource principal policy for Exporting Query Results to Object Storage. See Resource Principal Privileges.
- The required pre-authenticated request privileges. See PAR Privileges.
- The required uniform resource identifier (URI) privileges. See URI Privileges.

Pre-Authenticated Request Requirements

When you create pre-authenticated requests (PAR) to export query results, do the following:

- For **Pre-Authenticated Request Target**, select **Bucket** or **Objects with prefix**.
- For **Access Type**, select **Permit object reads and writes**.
- Select **Enable Object Listing**.

See Object Storage Pre-Authenticated Requests from *Oracle Cloud Infrastructure Documentation* to learn more.

MySQL HeatWave on AWS Requirements

For MySQL HeatWave on AWS, you need the Lakehouse role to access Object Storage. This role is required even when Lakehouse is disabled. For more information, see Configure Policies and Roles to Access Resources.

What's Next

Depending on the type of file you want to create, review the appropriate section:

- Exporting Query Results to CSV Files
- Exporting Query Results to Parquet Files
- Exporting Query Results to JSON Files

4.6.3 Export Query Results to CSV Files

You can export the query results to a RFC-4180 compatible CSV file.

4.6.3.1 Export Query Results to CSV Files with SQL Syntax

This topic shows the different ways to export query results as a CSV file with SQL syntax.

This topic contains the following sections:

- Before You Begin
- Syntax
- Parameter Descriptions
- Examples

- Example to Export the Query Results and Load the Data Using Auto Parallel Load
- What's Next

Before You Begin

- Review the following:
  - Overview of Exporting Query Results
  - Requirements to Export Query Results

Syntax

Use the following SQL `SELECT INTO` syntax to export query results to CSV files.

```
SELECT ... INTO OUTFILE
{URL | URI} 'uri'
[FORMAT csv]
[CHARACTER SET 'charset_name']
[HEADER {ON | OFF}]
[{FIELDS | COLUMNS}
  [NULL AS 'null_char']
  [TERMINATED BY 'string']
  [[OPTIONALLY] ENCLOSED BY 'char']
  [ESCAPED BY 'char'] ...
]
[LINES [TERMINATED BY 'string'] [STARTING BY 'string'] ...]
```

Parameter Descriptions

The following parameters are included in the SQL syntax to export the query results.

- `URI`: The Object Storage URL to store the export query results. You must specify a URL. As of MySQL 9.4.0, you can use the `URI` keyword. In earlier versions, you must use the `URL` keyword.

  The URL includes the Object Storage location with the region name, the tenancy, bucket, and the directory to store the export query results. You can use one of the following URL types:

  - URI: Supported as of MySQL 9.3.1. The Object Storage location is represented as a URI link. The URI must be a prefix URI and end in an unencoded `/` character.

    You can create the following types of URIs:

    - OCIFS Syntax:

      ```
      oci://bucket_name@namespace_name/object_path
      ```

      Example:

      ```
      oci://bucket_1@tenant_1/data_files/
      ```

    - PAR URI Syntax. You can use one of the following formats:

      ```
      https://namespace_name.objectstorage.region_name.oci.customer-oci.com/p/PAR_token/n/namespace_name/b
      ```

      ```
      https://objectstorage.region_name.oraclecloud.com/p/PAR_token/n/namespace_name/b/bucket_name/o/objec
      ```

      Example:

      ```
      https://tenant_1.objectstorage.us-ashburn-1.oci.customer-oci.com/p/.../n/tenant_1/b/bucket_1/o/data_
      ```

    - Native URI Syntax. You can use one of the following formats:

      ```
      https://namespace_name.objectstorage.region_name.oci.customer-oci.com/n/namespace_name/b/bucket_name
      ```

```
https://objectstorage.region_name.oraclecloud.com/n/namespace_name/b/bucket_name/o/object_path
```

Example:

```
https://tenant_1.objectstorage.us-ashburn-1.oci.customer-oci.com/n/tenant_1/b/bucket_1/o/data_fil
```

- PAR URL: The Object Storage location is represented as a pre-authenticated link. Learn more about Pre-Authenticated Request Requirements to export query results. To learn more about PARs, see Object Storage Pre-Authenticated Requests from *Oracle Cloud Infrastructure Documentation*.

  For security reasons, the PAR URL is hidden in the log file. For more information and samples, see Redacted PAR URLs in Exported Log Files.

  Syntax:

  ```
  https://objectstorage.region_name.oraclecloud.com/p/PAR_URI/n/namespace_name/b/bucket_name/o/", "pr
  ```

  Example:

  ```
  https://objectstorage.region1.oraclecloud.com/p/PTn9XsVegDBbSz1v-FgDma4G_e_78fJjv1GK04zlSqD9975MPZy
  ```

- Non-PAR URL with resource principal: The Object Storage location is represented as a bucket, namespace, and tenancy.

  Syntax:

  ```
  https://objectstorage.region_name.oraclecloud.com/n/namespace_name/b/bucket_name/o/prefixes
  ```

  Example:

  ```
  https://objectstorage.region1.oraclecloud.com/n/tenancy1/b/usr-bucket/o/summary/2024/Mar/
  ```

- AWS S3 URL: Available for MySQL HeatWave on AWS. The Object Storage location is represented as an AWS S3 URL. Learn more about MySQL HeatWave on AWS Requirements.

  Syntax:

  ```
  s3://bucket_name/prefixes/
  ```

  Example:

  ```
  s3://my-bucket/export-results/
  ```

- `FORMAT`: The file format to export the query results. Export the query results as a `CSV`, `Parquet`, or `JSON` (As of MySQL 9.3.1) file. By default, the results are exported to a `CSV` file.
- `HEADER`: Specifies if the `CSV` file has a header row. The default is `OFF`.
- `NULL AS`: Represents a string as a null value. The supported null values are `"NULL"` and `"\N"`.
- `{COLUMNS | FIELDS} TERMINATED BY`: Defines the single character used to enclose fields. The delimiter for `CSV` format is `","`. To export query results to Object Storage, COLUMNS or FIELDS TERMINATED BY only supports single character values.
- `LINES TERMINATED BY`: Defines the line terminator characters that represent the end of a row. The default is `"\n"`.
- `LINES STARTING BY`: Defines the character(s) that represent the prefix of a line.
- `ESCAPED_BY`: The escape character to ignore with a warning for export. The default is `"\"`.
- `FIELDS ENCLOSED BY`: The quotation marks to ignore with a warning for export. The default is `"\""`.

Examples

The following example uses default delimiters, and exports the query results to a `CSV` file. The file location is represented as an OCIFS URI. The `URI` keyword is supported as of MySQL 9.4.0.

```
SELECT sum(l_quantity) from lineitem
INTO OUTFILE URI
'oci://bucket1@tenancy1/data_files_1/';
```

The following example uses default delimiters, and exports the query results to a `CSV` file. The file location is represented as Resource principals.

```
SELECT sum(l_quantity) from lineitem
INTO OUTFILE URL
'https://objectstorage.region1.oraclecloud.com/n/tenancy1/b/usr-bucket/o/summary/2024/Mar/';
```

The following example uses default delimiters, and exports the query results to a `CSV` file with a header row. The file location is represented as a PAR URL.

```
SELECT sum(l_quantity) from lineitem
INTO OUTFILE URL
'https://objectstorage.region1.oraclecloud.com/p/PTn9XsVegDBbSz1v-FgDma4G_e_78fJjv1GK04zlSqD9975MPZydieUJAC
FORMAT csv
HEADER ON;
```

Example to Export the Query Results and Load the Data Using Auto Parallel Load

In the following example:

1. Create an aggregate table from the TPC-H benchmark—LINEITEM, NATION, ORDERS, and CUSTOMER tables by running a query that sums up sales grouped by year and month using the tables.
2. Load this data into MySQL HeatWave as an aggregate table by using Auto Parallel Load.
3. Update this table with new data loaded to the same bucket for new aggregates by using the incremental load feature. See Refresh Data Using Incremental Load.

Therefore, instead of having queries compute results by scanning all the rows for a given range of dates, you can run queries against this aggregate table, saving both time and cost.

The following query computes the sum of quantity and price and groups it by year, month, and day for the year 1992. The tenancy, bucket and the folder to save the query results are provided in the `URL` parameter mentioned in the `INTO OUTFILE` clause.

```
SELECT SUM(l.l_quantity)
  sum_quantity,
  SUM(o.o_totalprice) total_price,
  n.n_name nation_name,
  YEAR(o.o_orderdate) year,
  MONTH(o.o_orderdate) month,
  MONTHNAME(o.o_orderdate) month_name,
  DAY(o.o_orderdate) day_number
FROM
  lineitem l,
  orders o,
  nation n,
  customer c
WHERE
  l.l_orderkey = o.o_orderkey
  and n.n_nationkey = c.c_nationkey
  and c.c_custkey = o.o_custkey
  and YEAR(o.o_orderdate) = '1992'
GROUP BY
```

```
  YEAR(o.o_orderdate),
  nation_name,
  MONTH(o.o_orderdate),
  MONTHNAME(o.o_orderdate),
  DAY(o.o_orderdate)
INTO OUTFILE
  URL 'https://objectstorage.us-ashburn-1.oraclecloud.com/n/mytenancy/b/usr-bucket/o/summary-layer/'
  FORMAT csv HEADER ON;
```

After the query runs successfully, a destination folder with a unique system-generated prefix as the folder name is created, and the query results are stored in that folder. Refer to the sample success message.

```
 Note Code (3877) : Query results will be written to object storage with prefix : 'summary-layer/29597a
```

You can use Auto Parallel Load to load query results (CSV files) into a new aggregate table named `lineitem_agg` in MySQL HeatWave. Auto Parallel Load automatically gets the schema of the `CSV` files and creates the necessary table in the specified database and loads all the data from the CSV files into the new aggregate table.

```
SET @input_list = '[{
  "db_name": "lakehouse",
  "tables": [
    "lineitem_agg",{
      "table_name": "lineitem_agg",
      "engine_attribute": {
        "dialect": {
          "format": "csv",
          "field_delimiter": ",",
          "has_header": true,
          "record_delimiter": "\\n",
          "is_strict_mode": false
        },
        "file":[{
          "region": "us-ashburn-1",
          "namespace": "mytenancy",
          "bucket": "usr-bucket",
          "prefix": "summary-layer/"
        }]
      }
    }
  ]
}]';

SET @options = JSON_OBJECT('mode', 'normal', 'refresh_external_tables', 'true');
CALL sys.heatwave_load(CAST(@input_list AS JSON), @options);
```

After the aggregate table loads successfully, you can use it to do the following.

- Chapter 5, *Accelerate Query Processing*
- Chapter 6, *Train and Use Machine Learning Models*
- Chapter 7, *Perform AI-Powered Search and Content Generation*

What's Next

- Review the following:
  - Redacted PAR URLs in Exported Log Files
  - File Name Format for Exported Query Result Files

4.6.3.2 Export Query Results to CSV Files with JSON Syntax

This topic shows the different ways to export query results as a CSV file with JSON syntax.

This topic contains the following sections:

- Before You Begin
- JSON Syntax
- Parameter Descriptions for URI
- Parameter Descriptions for PAR URL
- Parameter Descriptions for Resource Principal
- Dialect Parameters for CSV Files
- Example for URI
- Example for PAR URL
- Examples for Resource Principal
- What's Next

Before You Begin

- Review the following:
  - Overview of Exporting Query Results
  - Requirements to Export Query Results

JSON Syntax

Use the following `SELECT INTO` JSON syntax to export the query results into a CSV file.

```
SELECT ... INTO OUTFILE WITH PARAMETERS

file_JSON_array: {
  begin-array file_def [, file_def] … end-array
}

file_def: file_RP_JSON | file_PAR_JSON | file_URI_JSON

file_def_RP_JSON: {
  "bucket": "bucket_name",
  "namespace": "namespace_name",
  "region": "region_name",
  {"prefix": "prefix_value" | "name": "filename" | "pattern" : "pattern_value"},
}

file_PAR_JSON: {
  "par": "PAR URL",
  ["prefix": "prefix_value", | "name": "filename", | "pattern" : "pattern_value",]
}

file_URI_JSON: {
  ("uri": "OCIFS_URI") | ("uri": "PAR_URI") | ("uri": "NATIVE_URI"),
}

dialect_JSON_object: begin-object dialect_option_key_value [, dialect_option_key_value]... end-object

dialect_option_key_value: {
    "format": "csv"
  | "field_delimiter": ","
  | "record_delimiter": "\\n"
  | "encoding": "utf8mb4"
  | "escape_character": "\\"
  | "quotation_marks": "\""
```

```
  | "null_value": "\\N"
  | "has_header": {true | false}
}
```

Parameter Descriptions for URI

Supported as of MySQL 9.3.1. The following parameters are included in the JSON syntax to export the query results to the Object Storage location represented as a URI.

- `URI`: The Object Storage location is represented as a URI link. The URI must be a prefix URI and end in an unencoded `/` character.

  You can specify the following types of URIs.

  - OCIFS Syntax:

    ```
    oci://bucket_name@namespace_name/object_path
    ```

    Example:

    ```
    oci://bucket_1@tenant_1/data_files/
    ```

  - PAR URI Syntax. You can use one of the following formats:

    ```
    https://namespace_name.objectstorage.region_name.oci.customer-oci.com/p/PAR_token/n/namespace_name/
    ```

    ```
    https://objectstorage.region_name.oraclecloud.com/p/PAR_token/n/namespace_name/b/bucket_name/o/obje
    ```

    Example:

    ```
    https://tenant_1.objectstorage.us-ashburn-1.oci.customer-oci.com/p/.../n/tenant_1/b/bucket_1/o/data
    ```

  - Native URI Syntax. You can use one of the following formats:

    ```
    https://namespace_name.objectstorage.region_name.oci.customer-oci.com/n/namespace_name/b/bucket_nam
    ```

    ```
    https://objectstorage.region_name.oraclecloud.com/n/namespace_name/b/bucket_name/o/object_path
    ```

    Example:

    ```
    https://tenant_1.objectstorage.us-ashburn-1.oci.customer-oci.com/n/tenant_1/b/bucket_1/o/data_files
    ```

- `dialect`: The dialect parameters assist you to organize the data in the exported `CSV` file.

Parameter Descriptions for PAR URL

The following parameters are included in the JSON syntax to export the query results to the Object Storage location represented as a PAR URL.

- `URI`: You can use a PAR URL to access Object Storage. Learn more about Pre-Authenticated Request Requirements to export query results. To learn more about PARs, see Object Storage Pre-Authenticated Requests from *Oracle Cloud Infrastructure Documentation*.

  For security reasons, the PAR URL is hidden in the log file. For more information and samples, see Redacted PAR URLs in Exported Log Files.

- `dialect`: The dialect parameters assist you to organize the data in the exported `CSV` file.

Parameter Descriptions for Resource Principal

The following parameters are included in the JSON syntax to export the query results to the Object Storage location based on the assigned resource principal.

- `bucket`: The bucket name.

- `prefix`: The directory located within the bucket to store the export query results. This is an optional parameter.
- `namespace`: The tenancy namespace.
- `region`: The region that the tenancy is in.
- `dialect`: The dialect parameters assist you to organize the data in the exported `CSV` file.

Dialect Parameters for CSV Files

The `dialect` parameters help you to structure the export query results data in the exported `CSV` file. The `dialect` parameters are applicable when you export the query results using `JSON` syntax. These parameters are based on the dialect parameters required to create the external table.

You can use the following `dialect` parameters to export the query results to `CSV` files.

- `format`: The file format to export the query results. Export the query results as a `csv`, `parquet`, or `json` (As of MySQL 9.3.1) file. By default, the results are exported to a `csv` file.
- `has_header`: Specify if the CSV file has a header row. The default is `false`.
- `null_value`: Represents a string as a null value. The supported null values are `"NULL"` and `"\N"`.
- `field_delimiter`: Defines a single character to enclose fields. The delimiter for CSV format is `","`. This parameter only supports single character values.
- `record_delimiter`: Specifies one or more characters to delimit records. The maximum field delimiter length is 64 characters. The default is `"\\n"`.
- `encoding`: Encoding is ignored with a warning for export. The default is `utf8mb4`.
- `escape_character`: The escape character to ignore with a warning for export. The default is `"\\"`.
- `quotation_marks`: The quotation marks to ignore with a warning for export. The default is `"\""`.

Example for URI

The following example uses an OCIFS URI to export query results. It uses default delimiters and specifies a header row in the file.

```
SELECT * FROM example
INTO OUTFILE WITH PARAMETERS
'{"file":[{
     "uri":"oci://bucket_1@tenancy1/data_files_1/"}],
     "dialect":
               {"has_header": true,
                "format": "csv"}
}';
```

Example for PAR URL

The following example uses a PAR URL to export query results. It uses default delimiters and specifies a header row in the file.

```
SELECT * FROM example
INTO OUTFILE WITH PARAMETERS
'{"file":[{
     "par":"https://objectstorage.region1.oraclecloud.com/p/1P4yKjnwXLEPh9Ol0Fbg2r81tvgJfh0XRcCcsr45ggh7uu
     "dialect":
               {"has_header": true,
                "format": "csv"}
}';
```

Examples for Resource Principal

The following example uses resource principals to export query results as a CSV file. No dialect parameters are defined, so default values are used.

```
SELECT sum(l_quantity) from lineitem limit 1000
INTO OUTFILE WITH PARAMETERS
'{"file":[{"region": "region1",
           "namespace": "org_namespace",
           "bucket": "usr-bucket",
           "prefix": "export/summary/"}]
}';
```

The following example uses resource principals to export query results as a CSV file. A header row is defined in the file.

```
SELECT sum(l_quantity) from lineitem limit 1000
INTO OUTFILE WITH PARAMETERS
'{"file":[{"region": "region1",
           "namespace": "org_namespace",
           "bucket": "usr-bucket",
           "prefix": "export/summary/"}]
           "dialect": {"has_header": true, "format": "csv"}
}';
```

What's Next

- Review the following:
  - Redacted PAR URLs in Exported Log Files
  - File Name Format for Exported Query Result Files

4.6.4 Export Query Results to Parquet Files

You can export query results to a Parquet file.

4.6.4.1 Export Query Results to Parquet Files with SQL Syntax

This topic shows the different ways to export query results as a Parquet file with SQL syntax.

Before You Begin

- Review the following:
  - Overview of Exporting Query Results
  - Requirements to Export Query Results

Syntax

Use the following SQL `SELECT INTO` syntax to export query results to Parquet files.

```
SELECT ... INTO OUTFILE
{URL | URI} 'uri'
[FORMAT parquet]
[COMPRESSION {snappy | gzip | brotli | zstd | lz4 | uncompressed}]
```

Note

As of MySQL 9.5.0, you can enclose options for `FORMAT` and `COMPRESSION` with or without quotation characters. In earlier versions, you must not enclose `FORMAT` options with quotation characters, and you must enclose `COMPRESSION` options in quotation characters.

Parameter Descriptions

The following parameters are included in the SQL syntax to export the query results.

- `URI`: The Object Storage URL to store the export query results. You must specify a URL. As of MySQL 9.4.0, you can use the `URI` keyword. For earlier versions, you must use the `URL` keyword.

  The URL includes the Object Storage location with the region name, the tenancy, bucket and the directory to store the export query results. You can use one of the following URL types:

  - URI: Supported as of MySQL 9.3.1. The Object Storage location is represented as a URI link. The URI must be a prefix URI and end in an unencoded `/` character.

    You can create the following types of URIs:

    - OCIFS Syntax:

      ```
      oci://bucket_name@namespace_name/object_path
      ```

      Example:

      ```
      oci://bucket_1@tenant_1/data_files/
      ```

    - PAR URI Syntax. You can use one of the following formats:

      ```
      https://namespace_name.objectstorage.region_name.oci.customer-oci.com/p/PAR_token/n/namespace_name/b
      ```

      ```
      https://objectstorage.region_name.oraclecloud.com/p/PAR_token/n/namespace_name/b/bucket_name/o/objec
      ```

      Example:

      ```
      https://tenant_1.objectstorage.us-ashburn-1.oci.customer-oci.com/p/.../n/tenant_1/b/bucket_1/o/data_
      ```

    - Native URI Syntax. You can use one of the following formats:

      ```
      https://namespace_name.objectstorage.region_name.oci.customer-oci.com/n/namespace_name/b/bucket_name
      ```

      ```
      https://objectstorage.region_name.oraclecloud.com/n/namespace_name/b/bucket_name/o/object_path
      ```

      Example:

      ```
      https://tenant_1.objectstorage.us-ashburn-1.oci.customer-oci.com/n/tenant_1/b/bucket_1/o/data_files/
      ```

  - PAR URL: The Object Storage location is represented as a pre-authenticated link. Learn more about Pre-Authenticated Request Requirements to export query results. To learn more about PARs, see Object Storage Pre-Authenticated Requests from *Oracle Cloud Infrastructure Documentation*.

    For security reasons, the PAR URL is hidden in the log file. For more information and samples, see Redacted PAR URLs in Exported Log Files.

    Syntax:

    ```
    https://objectstorage.region_name.oraclecloud.com/p/PAR_URI/n/namespace_name/b/bucket_name/o/", "prefix
    ```

    Example:

    ```
    https://objectstorage.region1.oraclecloud.com/p/PTn9XsVegDBbSz1v-FgDma4G_e_78fJjv1GK04zlSqD9975MPZydieU
    ```

  - Non-PAR URL with resource principal: The Object Storage location is represented as a bucket, namespace, and tenancy.

    Syntax:

    ```
    https://objectstorage.region_name.oraclecloud.com/n/namespace_name/b/bucket_name/o/prefixes
    ```

Example:

```
https://objectstorage.region1.oraclecloud.com/n/tenancy1/b/usr-bucket/o/summary/2024/Mar/
```

- AWS S3 URL: Available for MySQL HeatWave on AWS. The Object Storage location is represented as an AWS S3 URL. Learn more about MySQL HeatWave on AWS Requirements.

  Syntax:

  ```
  s3://bucket_name/prefixes/
  ```

  Example:

  ```
  s3://my-bucket/export-results/
  ```

- `FORMAT`: The file format to export the query results. Export the query results as a `CSV`, `Parquet`, or `JSON` (As of MySQL 9.3.1) file. By default, the results are exported to a `CSV` file.
- `COMPRESSION`: The file compression type for CSV format. The supported compression types are `uncompressed`, `snappy`, `gzip`, `brotli`, `zstd` and `lz4` (as supported by the Lakehouse load). By default, the `COMPRESSION` type is set to `snappy`.

Examples

The following example uses default delimiters, and exports the query results to a `Parquet` file with compression method SNAPPY. The file location is represented as an OCIFS URI. The `URI` keyword is supported as of MySQL 9.4.0.

```
mysql>SELECT sum(l_quantity) from lineitem
INTO OUTFILE URI
'oci://bucket_1@tenancy1/data_files_1/'
FORMAT parquet
COMPRESSION snappy;
```

The following example uses default delimiters, and exports the query results to a `Parquet` file with compression method SNAPPY. The file location is represented as resource principals.

```
mysql>SELECT sum(l_quantity) from lineitem
INTO OUTFILE URL
'https://objectstorage.region1.oraclecloud.com/n/tenancy1/b/usr-bucket/o/summary/2024/Mar/'
FORMAT parquet
COMPRESSION snappy;
```

What's Next

- Review the following:
  - Redacted PAR URLs in Exported Log Files
  - File Name Format for Exported Query Result Files

4.6.4.2 Export Query Results to Parquet Files with JSON Syntax

This topic shows the different ways to export query results to Parquet files.

This topic contains the following sections:

- Before You Begin
- JSON Syntax
- Parameter Descriptions for URI
- Parameter Descriptions for PAR URL

- Parameter Descriptions for Resource Principal
- Example for URI
- Example for PAR
- Examples for Resource Principal
- What's Next

Before You Begin

- Review the following:
  - Overview of Exporting Query Results
  - Requirements to Export Query Results

JSON Syntax

Use the following `SELECT INTO` JSON syntax to export the query results into a Parquet file.

```
SELECT ... INTO OUTFILE WITH PARAMETERS
file_JSON_array: {
  begin-array file_def [, file_def] … end-array
}

file_def: file_RP_JSON | file_PAR_JSON | file_URI_JSON

file_def_RP_JSON: {
  "bucket": "bucket_name",
  "namespace": "namespace_name",
  "region": "region_name",
  {"prefix": "prefix_value" | "name": "filename" | "pattern" : "pattern_value"},
}

file_PAR_JSON: {
  "par": "PAR URL",
  ["prefix": "prefix_value", | "name": "filename", | "pattern" : "pattern_value",]
}

file_URI_JSON: {
  ("uri": "OCIFS_URI") | ("uri": "PAR_URI") | ("uri": "NATIVE_URI"),
}

dialect_JSON_object: begin-object dialect_option_key_value [, dialect_option_key_value]... end-object

dialect_option_key_value: {
    "format": "parquet"
  | "compression": {"uncompressed"|"snappy"|"gzip"|"brotli"|"zstd"|"lz4"}
}
```

Parameter Descriptions for URI

Supported as of MySQL 9.3.1. The following parameters are included in the JSON syntax to export the query results to the Object Storage location represented as a URI.

- `URI`: The Object Storage location is represented as a URI link. The URI must be a prefix URI and end in an unencoded `/` character.

  You can specify the following types of URIs.

  - OCIFS Syntax:

    ```
    oci://bucket_name@namespace_name/object_path
    ```

    Example:

```
oci://bucket_1@tenant_1/data_files/
```

- PAR URI Syntax. You can use one of the following formats:

  ```
  https://namespace_name.objectstorage.region_name.oci.customer-oci.com/p/PAR_token/n/namespace_name/
  ```

  ```
  https://objectstorage.region_name.oraclecloud.com/p/PAR_token/n/namespace_name/b/bucket_name/o/obje
  ```

  Example:

  ```
  https://tenant_1.objectstorage.us-ashburn-1.oci.customer-oci.com/p/.../n/tenant_1/b/bucket_1/o/data
  ```

- Native URI Syntax. You can use one of the following formats:

  ```
  https://namespace_name.objectstorage.region_name.oci.customer-oci.com/n/namespace_name/b/bucket_nam
  ```

  ```
  https://objectstorage.region_name.oraclecloud.com/n/namespace_name/b/bucket_name/o/object_path
  ```

  Example:

  ```
  https://tenant_1.objectstorage.us-ashburn-1.oci.customer-oci.com/n/tenant_1/b/bucket_1/o/data_files
  ```

- `dialect`: The dialect parameters assist you to organize the data in the exported `CSV` file.

Parameter Descriptions for PAR URL

The following parameters are included in the JSON syntax to export the query results to the Object Storage location represented as a PAR URL.

- `URL`: You can use a PAR URL to access Object Storage. Learn more about Pre-Authenticated Request Requirements to export query results. To learn more about PARs, see Object Storage Pre-Authenticated Requests from *Oracle Cloud Infrastructure Documentation*.

  For security reasons, the PAR URL is hidden in the log file. For more information and samples, see Redacted PAR URLs in Exported Log Files.

- `format`: The file format to export the query results. Export the query results as a `CSV`, `Parquet`, or `JSON` (As of MySQL 9.3.1) file. By default, the results are exported to a `CSV` file.

- `compression`: The file compression type for CSV format. The supported compression types are `uncompressed`, `snappy`, `gzip`, `brotli`, `zstd` and `lz4` (as supported by the Lakehouse load).

Parameter Descriptions for Resource Principal

The following parameters are included in the JSON syntax to export the query results to the Object Storage location based on the assigned resource principal.

- `format`: The file format to export the query results. Export the query results as a `CSV`, `Parquet`, or `JSON` (As of MySQL 9.3.1) file. By default, the results are exported to a `CSV` file.

- `bucket`: The bucket name.

- `prefix`: The directory located within the bucket to store the export query results. This is an optional parameter.

- `namespace`: The tenancy namespace.

- `region`: The region that the tenancy is in.

- `compression`: The file compression type for CSV format. The supported compression types are `uncompressed`, `snappy`, `gzip`, `brotli`, `zstd` and `lz4` (as supported by the Lakehouse load).

Example for URI

The following example uses a specified OCIFS URI to export the query results as a `Parquet` file.

```
SELECT sum(l_quantity) from lineitem limit 1000
INTO OUTFILE WITH PARAMETERS
'{"file":[{
     "uri":"oci://bucket_1@tenancy1/data_files_1/"}],
     "dialect":
    {"format": "parquet"
    }
}';
```

Example for PAR

The following example uses a specified PAR URL to export the query results as a `Parquet` file.

```
SELECT sum(l_quantity) from lineitem limit 1000
INTO OUTFILE WITH PARAMETERS
'{"file":[{
     "par":"https://objectstorage.region1.oraclecloud.com/p/PTn9XsVegDBbSz1v-FgDma4G_e_78fJjv1GK04zlSqD9975
     "dialect":
    {"format": "parquet"
    }
}';
```

Examples for Resource Principal

The following example uses resource principals to export the query results as a `parquet` file. This sample uses the default values, dialect parameters are not required.

```
mysql>SELECT sum(l_quantity) from lineitem limit 1000
INTO OUTFILE WITH PARAMETERS
'{"file":[{"region": region1,
           "namespace": org_namespace,
           "bucket": usr-bucket,
           "prefix": export/summary/
           "dialect": {"format": "parquet"}

         }]
}';
```

The following example sets the compression to `snappy`.

```
mysql> SELECT sum(l_quantity) from lineitem limit 1000
INTO OUTFILE WITH PARAMETERS
'{"file":[{"region": "region1",
           "namespace": "org_namespace",
           "bucket": "usr-bucket",
           "prefix": "prefix1/prefix2/"
         }]
           "dialect": {"compression": "snappy", "format": "parquet"}
}';
```

What's Next

- Review the following:
  - Redacted PAR URLs in Exported Log Files
  - File Name Format for Exported Query Result Files

4.6.5 Export Query Results to ND-JSON Files

As of MySQL 9.3.1, you can export the query results to a Newline-Delimited JSON (ND-JSON) file.

4.6.5.1 Export Query Results to ND-JSON Files with Non-JSON Syntax

The option to export query results as an ND-JSON file is available as of MySQL 9.3.1.

This topic contains the following sections:

- Before You Begin
- Syntax
- Parameter Descriptions
- Examples
- What's Next

Before You Begin

- Review the following:
  - Overview of Exporting Query Results
  - Requirements to Export Query Results

Syntax

Use the following SQL `SELECT INTO` syntax to export query results to ND-JSON files.

```
SELECT ... INTO OUTFILE
{URL | URI} 'uri'
[FORMAT json]
[LINES TERMINATED BY 'string']
```

Parameter Descriptions

The following parameters are included in the SQL syntax to export the query results.

- `URI`: The Object Storage URL to store the export query results. You must specify a URL.

  The URL includes the Object Storage location with the region name, the tenancy, bucket and the directory to store the export query results. You can use one of the following URL types:

  - URI: The Object Storage location is represented as a URI link. The URI must be a prefix URI and end in an unencoded `/` character.

    You can create the following types of URIs:

    - OCIFS Syntax:

      ```
      oci://bucket_name@namespace_name/object_path
      ```

      Example:

      ```
      oci://bucket_1@tenant_1/data_files/
      ```

    - PAR URI Syntax. You can use one of the following formats:

      ```
      https://namespace_name.objectstorage.region_name.oci.customer-oci.com/p/PAR_token/n/namespace_nam
      ```

      ```
      https://objectstorage.region_name.oraclecloud.com/p/PAR_token/n/namespace_name/b/bucket_name/o/ob
      ```

      Example:

      ```
      https://tenant_1.objectstorage.us-ashburn-1.oci.customer-oci.com/p/.../n/tenant_1/b/bucket_1/o/da
      ```

    - Native URI Syntax. You can use one of the following formats:

      ```
      https://namespace_name.objectstorage.region_name.oci.customer-oci.com/n/namespace_name/b/bucket_n
      ```

      ```
      https://objectstorage.region_name.oraclecloud.com/n/namespace_name/b/bucket_name/o/object_path
      ```

Example:

```
https://tenant_1.objectstorage.us-ashburn-1.oci.customer-oci.com/n/tenant_1/b/bucket_1/o/data_files/
```

- PAR URL: The Object Storage location is represented as a pre-authenticated link. Learn more about Pre-Authenticated Request Requirements to export query results. To learn more about PARs, see Object Storage Pre-Authenticated Requests from *Oracle Cloud Infrastructure Documentation*.

  For security reasons, the PAR URL is hidden in the log file. For more information and samples, see Redacted PAR URLs in Exported Log Files.

  Syntax:

  ```
  https://objectstorage.region_name.oraclecloud.com/p/PAR_URI/n/namespace_name/b/bucket_name/o/", "prefix
  ```

  Example:

  ```
  https://objectstorage.region1.oraclecloud.com/p/PTn9XsVegDBbSz1v-FgDma4G_e_78fJjv1GK04zlSqD9975MPZydieU
  ```

- Non-PAR URL with resource principal: The Object Storage location is represented as bucket, namespace and tenancy.

  Syntax:

  ```
  https://objectstorage.region_name.oraclecloud.com/n/namespace_name/b/bucket_name/o/prefixes
  ```

  Example:

  ```
  https://objectstorage.region1.oraclecloud.com/n/tenancy1/b/usr-bucket/o/summary/2024/Mar/
  ```

- AWS S3 URL: Available for MySQL HeatWave on AWS. The Object Storage location is represented as an AWS S3 URL. Learn more about MySQL HeatWave on AWS Requirements.

  Syntax:

  ```
  s3://bucket_name/prefixes/
  ```

  Example:

  ```
  s3://my-bucket/export-results/
  ```

- `LINES TERMINATED BY`: Defines the line terminator characters that represent the end of a row. The default is `"\n"`.

Examples

The following example exports the query results to an ND-JSON file. The file location is represented as an OCIFS URI. The `URI` keyword is supported as of MySQL 9.4.0.

```
SELECT c_varchar as a FROM datatypes_dw
       UNION ALL
       SELECT c_json as a FROM datatypes_dw
INTO OUTFILE URI
'oci://bucket_1@tenancy1/data_files_1/'
FORMAT json;
```

The following example exports the query results to an ND-JSON file. The file location is represented as a PAR URL.

```
SELECT c_varchar as a FROM datatypes_dw
       UNION ALL
       SELECT c_json as a FROM datatypes_dw
INTO OUTFILE URL
'https://objectstorage.region1.oraclecloud.com/p/PTn9XsVegDBbSz1v-FgDma4G_e_78fJjv1GK04zlSqD9975MPZydieUJfs
```

```
FORMAT json;
```

What's Next

- Review the following:
  - Redacted PAR URLs in Exported Log Files
  - File Name Format for Exported Query Result Files

4.6.5.2 Export Query Results to ND-JSON Files with JSON Syntax

The option to export query results as an ND-JSON file is available as of MySQL 9.3.1.

This topic shows the different ways to export query results as an ND-JSON file.

This topic contains the following sections:

- Before You Begin
- JSON Syntax
- Parameter Descriptions for URI
- Parameter Descriptions for PAR URL
- Parameter Descriptions for Resource Principal
- Example for URI
- Example for PAR URL
- Example for Resource Principal
- What's Next

Before You Begin

- Review the following:
  - Overview of Exporting Query Results
  - Requirements to Export Query Results

JSON Syntax

Use the following `SELECT INTO` JSON syntax to export the query results into an ND-JSON file.

```
SELECT ... INTO OUTFILE WITH PARAMETERS
file_JSON_array: {
  begin-array file_def [, file_def] … end-array
}

file_def: file_RP_JSON | file_PAR_JSON | file_URI_JSON

file_def_RP_JSON: {
  "bucket": "bucket_name",
  "namespace": "namespace_name",
  "region": "region_name",
  {"prefix": "prefix_value" | "name": "filename" | "pattern" : "pattern_value"},
}

file_PAR_JSON: {
  "par": "PAR URL",
  ["prefix": "prefix_value", | "name": "filename", | "pattern" : "pattern_value",]
}

file_URI_JSON: {
```

```
  ("uri": "OCIFS_URI") | ("uri": "PAR_URI") | ("uri": "NATIVE_URI"),
}

dialect_JSON_object: begin-object dialect_option_key_value [, dialect_option_key_value]... end-object

dialect_option_key_value: {
{
    "format": "json"
  | "record_delimiter": "\\n"
}
```

Parameter Descriptions for URI

The following parameters are included in the JSON syntax to export the query results to the Object Storage location represented as a URI.

- `URI`: The Object Storage location is represented as a URI link. The URI must be a prefix URI and end in an unencoded `/` character.

  You can specify the following types of URIs.

  - OCIFS Syntax:

    ```
    oci://bucket_name@namespace_name/object_path
    ```

    Example:

    ```
    oci://bucket_1@tenant_1/data_files/
    ```

  - PAR URI Syntax. You can use one of the following formats:

    ```
    https://namespace_name.objectstorage.region_name.oci.customer-oci.com/p/PAR_token/n/namespace_name/b/bu
    ```

    ```
    https://objectstorage.region_name.oraclecloud.com/p/PAR_token/n/namespace_name/b/bucket_name/o/object_p
    ```

    Example:

    ```
    https://tenant_1.objectstorage.us-ashburn-1.oci.customer-oci.com/p/.../n/tenant_1/b/bucket_1/o/data_fi
    ```

  - Native URI Syntax. You can use one of the following formats:

    ```
    https://namespace_name.objectstorage.region_name.oci.customer-oci.com/n/namespace_name/b/bucket_name/o/
    ```

    ```
    https://objectstorage.region_name.oraclecloud.com/n/namespace_name/b/bucket_name/o/object_path
    ```

    Example:

    ```
    https://tenant_1.objectstorage.us-ashburn-1.oci.customer-oci.com/n/tenant_1/b/bucket_1/o/data_files/
    ```

- `dialect`: The dialect parameters assist you to organize the data in the exported `CSV` file.

Parameter Descriptions for PAR URL

The following parameters are included in the JSON syntax to export the query results to the Object Storage location represented as a PAR URL.

- `URI`: You can use a PAR URL to access Object Storage. Learn more about Pre-Authenticated Request Requirements to export query results. To learn more about PARs, see Object Storage Pre-Authenticated Requests from *Oracle Cloud Infrastructure Documentation*.

  For security reasons, the PAR URL is hidden in the logs file. For more information and samples, see Redacted PAR URLs in Exported Log Files.

- `dialect`: Set the exported file to `JSON` format. You also have the option to set the `record_delimiter` parameter, which specifies one or more characters to delimit records. The maximum field delimiter length is 64 characters. The default is `"\\n"`.

Parameter Descriptions for Resource Principal

The following parameters are included in the JSON syntax to export the query results to the Object Storage location based on the assigned resource principal.

- `bucket`: The bucket name.
- `prefix`: The directory located within the bucket to store the export query results. This is an optional parameter.
- `namespace`: The tenancy namespace.
- `region`: The region that the tenancy is in.
- `dialect`: Set the exported file to `JSON` format. You also have the option to set the `record_delimiter` parameter, which specifies one or more characters to delimit records. The maximum field delimiter length is 64 characters. The default is `"\\n"`.

Example for URI

The following example uses a specific OCIFS URI to export the query results as an ND-JSON file.

```
SELECT sum(l_quantity) from lineitem limit 1000
INTO OUTFILE WITH PARAMETERS
'{"file":[{
     "uri":"oci://bucket_1@tenancy1/data_files_1/"}],
     "dialect":
    {"format": "json"
    }
}';
```

Example for PAR URL

The following example uses a specific PAR URL to export the query results as an ND-JSON file.

```
SELECT sum(l_quantity) from lineitem limit 1000
INTO OUTFILE WITH PARAMETERS
'{"file":[{
     "par":"https://objectstorage.region1.oraclecloud.com/p/PTn9XsVegDBbSz1v-FgDma4G_e_78fJjv1GK04zlSqD
     "dialect":
    {"format": "json"
    }
}';
```

Example for Resource Principal

The following example uses the specified resource principals to export the query results as an ND-JSON file.

```
SELECT sum(l_quantity) from lineitem limit 1000
INTO OUTFILE WITH PARAMETERS
'{"file":[{"region": "region1",
           "namespace": "org_namespace",
           "bucket": "usr-bucket",
           "prefix": "prefix1/prefix2/
         }],
           "dialect": {"format": "json"}
}';
```

What's Next

- Review the following:
  - Redacted PAR URLs in Exported Log Files
  - File Name Format for Exported Query Result Files

4.6.6 Redacted PAR URLs in Exported Log Files

The pre-authenticated request (PAR) URL is hidden in the log file for security reasons.

The following example exports the query results as a CSV file to the object storage location represented as a PAR URL. It also displays the latest query results from the query text from the `performance_schema.rpd_query_stats table` and retrieves all the MySQL statements currently being executed in all MySQL active connections on the network.

The output displays with the redacted PAR URL.

```
mysql> SELECT * INTO OUTFILE URL
"https://mybucket.objectstorage.myregion.oci.customer-oci.com/p/CAN_BE_DISPLAYED_PAR_ID/n/myNamespace/b/MyP
FORMAT CSV
FROM  LINEITEM;

# Performance schema logs
mysql> SELECT QUERY_TEXT FROM performance_schema.rpd_query_stats ORDER BY QUERY_ID DESC LIMIT 1;
+-----------------------------------------------------------------------------------------------------------------
| QUERY_TEXT
+-----------------------------------------------------------------------------------------------------------------
| SELECT * INTO OUTFILE URL
| "https://mybucket.objectstorage.myregion.oci.customer-oci.com/p/<redacted>/n/       |
|  myNamespace/b/MyParBucket/o/"
| FORMAT CSV
| FROM  LINEITEM ;
+-----------------------------------------------------------------------------------------------------------------
```

```
# In the process list
mysql> SELECT info FROM INFORMATION_SCHEMA.PROCESSLIST;
+-----------------------------------------------------------------------------------------------------------------
| info
+-----------------------------------------------------------------------------------------------------------------
| NULL
| SELECT * INTO OUTFILE URL
| "https://mybucket.objectstorage.myregion.oci.customer-oci.com/p/<redacted>/n/       |
|  myNamespace/b/MyParBucket/o/"
| FORMAT CSV
| FROM  LINEITEM;
| SELECT info FROM INFORMATION_SCHEMA.PROCESSLIST
+-----------------------------------------------------------------------------------------------------------------
```

4.6.7 File Name Format for Exported Query Result Files

Exported query results are in CSV, `Parquet`, or `JSON` (As of MySQL 9.3.1) format. The number of output files and size of individual files depends on the amount of data generated by the query.

The names for the output files are auto-generated and based on the following format:

```
file_prefix/universal unique number/nnode_ID-tthread_ID-cChunk_ID.compression.file_format
```

File Name Parameters

- File_prefix: The user-provided prefix for the file name.
- Universal unique number: The unique ID generated for each query.
- Node_ID: The node ID of the file.
- Thread_ID: The thread ID of the file.
- Chunk_ID: The chunk ID of the file.
- Compression: The file compression type selected by the user.
- File_format: The output file format selected by the user.

Export File Prefix

After the query results are exported to Object Storage, the export file prefix is added to the `rpd_query_stats` table. See The rpd_query_stats Table. The export file prefix includes the file prefix and the unique ID.

To view the export file prefix for all export queries, run the following command:

```
mysql>SELECT QUERY_ID,EXPORT_FILE_PREFIX from performance_schema.rpd_query_stats WHERE EXPORT_FILE_PREF
```

To view the export file prefix for the most recent export, run the following command:

```
mysql>SELECT /*+set_var(use_secondary_engine=off) */  EXPORT_FILE_PREFIX FROM performance_schema.rpd_qu
      WHERE EXPORT_FILE_PREFIX is NOT NULL ORDER BY QUERY_ID DESC LIMIT 1;
```

Sample Object Names

In the following example, these object names are generated for the user-defined prefix:

- **`outfile-basic`**
- universal unique number: `976b11a5-4499-11ef-acb7-0200171701b`

The export file prefix `outfile-basic/976b11a5-4499-11ef-acb7-0200171701b` is appended to the filenames.

Refer to the following file names:

```
universal unique number: 976b11a5-4499-11ef-acb7- 0200171701b (unique number generated for each query),
compression: snappy,
format : parquet
  - outfile-basic/976b11a5-4499-11ef-acb7-0200171701b/n0-t1-c1.snappy.parquet
  - outfile-basic/976b11a5-4499-11ef-acb7-0200171701b/n0-t0-c1.snappy.parquet
  - outfile-basic/976b11a5-4499-11ef-acb7-0200171701b/n0-t0-c2.snappy.parquet
  - outfile-basic/976b11a5-4499-11ef-acb7-0200171701b/n0-t1-c2.snappy.parquet
```

4.7 Recover DB System Data

In the event of a problem where query processing stops or the MySQL HeatWave Cluster stops, the external tables are recovered when the Cluster resumes and begins processing queries again using the MySQL HeatWave recovery process. See the following to learn more:

- MySQL HeatWave Cluster Failure and Recovery in the *MySQL HeatWave on OCI Service Guide*.
- MySQL HeatWave Cluster Data Recovery in the *MySQL HeatWave on AWS Service Guide*.
- Restore a Database in the *MySQL HeatWave for Azure Service Guide*.

4.8 Monitor Data Loading

This section provides queries that you can use to monitor data loading.

4.8.1 MySQL HeatWave Node Status Monitoring

To view the status of each MySQL HeatWave node:

```
mysql> SELECT ID, STATUS
          FROM performance_schema.rpd_nodes;
+----+---------------+
| ID | STATUS        |
+----+---------------+
|  0 | AVAIL_RNSTATE |
|  1 | AVAIL_RNSTATE |
```

```
+----+--------------+
```

For column descriptions, see Section 8.1.6, “The rpd_nodes Table”.

4.8.2 MySQL HeatWave Memory Usage Monitoring

To view the memory usage for each MySQL HeatWave node:

```
mysql> SELECT ID, MEMORY_USAGE, MEMORY_TOTAL, BASEREL_MEMORY_USAGE
         FROM performance_schema.rpd_nodes;
+----+--------------+--------------+----------------------+
| ID | MEMORY_USAGE | MEMORY_TOTAL | BASEREL_MEMORY_USAGE |
+----+--------------+--------------+----------------------+
|  0 |    115760258 | 515396075520 |            115760152 |
|  1 |    115845086 | 515396075520 |            115844980 |
+----+--------------+--------------+----------------------+
```

For column descriptions, see Section 8.1.6, “The rpd_nodes Table”.

4.8.3 Data Load Progress and Status Monitoring

- The time required to load a table into MySQL HeatWave depends on data size. You can monitor load progress by issuing the following query, which returns a percentage value indicating load progress.

```
mysql> SELECT VARIABLE_VALUE
          FROM performance_schema.global_status
          WHERE VARIABLE_NAME = 'rapid_load_progress';
+----------------+
| VARIABLE_VALUE |
+----------------+
| 100.000000     |
+----------------+
```

Note

If necessary, you can halt a load operation using `Ctrl-C`.

- To check the load status of tables in a particular schema:

```
mysql> USE performance_schema;
mysql> SELECT NAME, LOAD_STATUS
          FROM rpd_tables,rpd_table_id
          WHERE rpd_tables.ID = rpd_table_id.ID AND SCHEMA_NAME LIKE 'tpch';
+-----------------------------+---------------------+
| NAME                        | LOAD_STATUS         |
+-----------------------------+---------------------+
| tpch.supplier               | AVAIL_RPDGSTABSTATE |
| tpch.partsupp               | AVAIL_RPDGSTABSTATE |
| tpch.orders                 | AVAIL_RPDGSTABSTATE |
| tpch.lineitem               | AVAIL_RPDGSTABSTATE |
| tpch.customer               | AVAIL_RPDGSTABSTATE |
| tpch.nation                 | AVAIL_RPDGSTABSTATE |
| tpch.region                 | AVAIL_RPDGSTABSTATE |
| tpch.part                   | AVAIL_RPDGSTABSTATE |
+-----------------------------+---------------------+
```

For information about load statuses, see Section 8.1.10, “The rpd_tables Table”.

- To view the amount of data loaded in MySQL HeatWave for a table, in bytes:

```
mysql> USE performance_schema;
mysql> SELECT rpd_table_id.TABLE_NAME, rpd_tables.NROWS,
          rpd_tables.LOAD_STATUS, rpd_tables.SIZE_BYTES
          FROM rpd_table_id, rpd_tables
          WHERE rpd_table_id.ID = rpd_tables.ID
          ORDER BY SIZE_BYTES;
------------+---------+---------------------+------------+
```

```
| TABLE_NAME | NROWS   | LOAD_STATUS         | SIZE_BYTES |
+------------+---------+---------------------+------------+
| region     |       5 | AVAIL_RPDGSTABSTATE |    4194304 |
| nation     |      25 | AVAIL_RPDGSTABSTATE |    8388608 |
| part       |  200000 | AVAIL_RPDGSTABSTATE |   33554432 |
| customer   |  150000 | AVAIL_RPDGSTABSTATE |   41943040 |
| orders     | 1500000 | AVAIL_RPDGSTABSTATE |  226492416 |
+------------+---------+---------------------+------------+
```

- To view the time that a table load operation completed:

```
mysql> USE performance_schema;
mysql> SELECT rpd_table_id.TABLE_NAME, rpd_tables.NROWS,
          rpd_tables.LOAD_STATUS, rpd_tables.LOAD_END_TIMESTAMP
          FROM rpd_table_id, rpd_tables
          WHERE rpd_table_id.ID = rpd_tables.ID;
+------------+---------+---------------------+----------------------------+
| TABLE_NAME | NROWS   | LOAD_STATUS         | LOAD_END_TIMESTAMP         |
+------------+---------+---------------------+----------------------------+
| region     |       5 | AVAIL_RPDGSTABSTATE | 2021-12-06 14:32:15.209825 |
| part       |  200000 | AVAIL_RPDGSTABSTATE | 2021-12-06 14:32:07.594575 |
| customer   |  150000 | AVAIL_RPDGSTABSTATE | 2021-12-06 14:31:57.210649 |
| nation     |      25 | AVAIL_RPDGSTABSTATE | 2021-12-06 14:17:53.472208 |
| orders     | 1500000 | AVAIL_RPDGSTABSTATE | 2021-12-06 14:24:45.809931 |
+------------+---------+---------------------+----------------------------+
```

- To view the time that the last successful recovery took across all tables.

```
mysql> SELECT variable_value
          FROM performance_schema.global_status
          WHERE variable_name="rapid_recovery_time";
+----------------+
| variable_value |
+----------------+
| N/A            |
+----------------+
```

4.8.4 Change Propagation Monitoring

- To determine if change propagation is enabled globally, query the `rapid_change_propagation_status` variable:

```
mysql> SELECT VARIABLE_VALUE FROM performance_schema.global_status
          WHERE VARIABLE_NAME = 'rapid_change_propagation_status';
+----------------+
| VARIABLE_VALUE |
+----------------+
| ON             |
+----------------+
```

- To determine if change propagation is enabled for a particular table, query the `POOL_TYPE` data from the MySQL HeatWave Performance Schema tables.

  - As of MySQL 9.2.1, `TRANSACTIONAL` indicates that change propagation is enabled for the table. `SNAPSHOT` indicates that change propagation is disabled.

  - In previous versions of MySQL, `RAPID_LOAD_POOL_TRANSACTIONAL` indicates that change propagation is enabled for the table. `RAPID_LOAD_POOL_SNAPSHOT` indicates that change propagation is disabled.

```
mysql> USE performance_schema;
mysql> SELECT NAME, POOL_TYPE FROM rpd_tables,rpd_table_id
          WHERE rpd_tables.ID = rpd_table_id.ID AND SCHEMA_NAME LIKE 'tpch';
+---------------+---------------+
| NAME          | POOL_TYPE     |
+---------------+---------------+
| tpch.orders   | TRANSACTIONAL |
| tpch.region   | TRANSACTIONAL |
| tpch.lineitem | TRANSACTIONAL |
```

```
| tpch.supplier | TRANSACTIONAL |
| tpch.partsupp | TRANSACTIONAL |
| tpch.part     | TRANSACTIONAL |
| tpch.customer | TRANSACTIONAL |
+---------------+---------------+
```

Chapter 5 Accelerate Query Processing

Table of Contents

5.1 About MySQL HeatWave 212
5.2 Supported SQL Modes 212
5.3 Supported Functions and Operators 213
5.3.1 Aggregate Functions 213
5.3.2 Arithmetic Operators 218
5.3.3 Cast Functions and Operators 218
5.3.4 Comparison Functions and Operators 219
5.3.5 Control Flow Functions and Operators 220
5.3.6 Data Masking and De-Identification Functions 220
5.3.7 Encryption and Compression Functions 221
5.3.8 JSON Functions 222
5.3.9 Logical Operators 224
5.3.10 Mathematical Functions 224
5.3.11 String Functions and Operators 225
5.3.12 Temporal Functions 228
5.3.13 Vector Functions 230
5.3.14 Window Functions 231
5.3.15 Miscellaneous Functions 231
5.4 Run Queries 232
5.4.1 Requirements for Running Queries 232
5.4.2 SELECT Statement Clauses for MySQL HeatWave 233
5.4.3 Explain Query and Check Execution Time 234
5.4.4 CREATE TABLE ... SELECT Statements 236
5.4.5 Create MySQL HeatWave Temporary Tables 237
5.4.6 INSERT ... SELECT Statements 239
5.4.7 Query Views 240
5.4.8 Query Materalized Views 241
5.4.9 View Query Runtimes and Estimates 244
5.4.10 Diagnose Query Offload 247
5.4.11 Auto Scheduling 254
5.4.12 Auto Query Plan Improvement 255
5.4.13 Dynamic Query Offload 255
5.5 Modify Tables 256
5.6 Analyze Tables 256
5.7 Run Tasks Asynchronously 257
5.8 Optimize Workloads for OLAP 260
5.8.1 String Column Encoding 261
5.8.2 Data Placement Keys 266
5.8.3 About MySQL HeatWave Autopilot Advisor 268
5.8.4 Auto Encoding 269
5.8.5 Auto Data Placement 272
5.8.6 Auto Query Time Estimation 275
5.8.7 Unload Advisor 278
5.8.8 Autopilot Report Table 279
5.8.9 Advisor Report Table 280
5.9 Optimize Workloads for OLTP 281
5.9.1 Autopilot Indexing 282
5.10 MySQL HeatWave Monitoring 285
5.10.1 Query Execution Monitoring 285
5.10.2 Query History and Statistics Monitoring 286
5.10.3 Scanned Data Monitoring 288

5.1 About MySQL HeatWave

When a MySQL HeatWave Cluster is enabled, queries that meet certain prerequisites are automatically offloaded from the DB System to the MySQL HeatWave Cluster for accelerated processing.

Queries are issued from a MySQL client or application that interacts with the MySQL HeatWave Cluster by connecting to the DB System. Results are returned to the DB System and to the MySQL client or application that issued the query.

Manually loading data into MySQL HeatWave involves preparing tables on the DB System and executing load statements. The Auto Parallel Load utility facilitates the process of loading data by automating required steps and optimizing the number of parallel load threads. See Section 4.2.5, “Load Data Using Auto Parallel Load”.

For MySQL HeatWave on AWS, load data into MySQL HeatWave using the MySQL HeatWave Console. See Manage Data in MySQL HeatWave with Workspaces in the *MySQL HeatWave on AWS Service Guide*.

For MySQL HeatWave for Azure, see Importing Data to MySQL HeatWave in the *MySQL HeatWave for Azure Service Guide*.

When MySQL HeatWave loads a table, the data is sharded and distributed among MySQL HeatWave nodes. After a table is loaded, DML operations on the tables are automatically propagated to the MySQL HeatWave nodes. No user action is required to synchronize data. For more information, see Section 4.2.7, “About Change Propagation”.

Data loaded into MySQL HeatWave, including propagated changes, automatically get persisted in the MySQL HeatWave Storage Layer to Object Storage for fast recovery in case of a MySQL HeatWave node or cluster failure.

As of MySQL 9.2.0, MySQL HeatWave Storage Layer persists additional metadata to allow fast recovery after a planned or unplanned restart of Standalone DB system (without High Availability). This enables MySQL HeatWave to reload data from the Storage Layer after the DB System restarts. In previous versions of MySQL, MySQL HeatWave reloads data from the original source, InnoDB or the Object Storage in case of Lakehouse tables, which takes longer for data to reload.

After running a number of queries, use the MySQL HeatWave Autopilot Advisor to optimize the workload. Advisor analyzes the data and query history to provide string column encoding and data placement recommendations. See Section 5.8, “Optimize Workloads for OLAP”.

What's Next

- Check the list of supported data types, supported SQL modes, and supported functions and operators.

5.2 Supported SQL Modes

Default DB System SQL modes are supported, which include `ONLY_FULL_GROUP_BY`, `STRICT_TRANS_TABLES`, `NO_ZERO_IN_DATE`, `NO_ZERO_DATE`, `ERROR_FOR_DIVISION_BY_ZERO`, and `NO_ENGINE_SUBSTITUTION`. See Server SQL Modes.

In addition, the following SQL modes are supported:

- `ANSI_QUOTES`
- `HIGH_NOT_PRECEDENCE`
- `IGNORE_SPACE`
- `NO_BACKSLASH_ESCAPES`

- `REAL_AS_FLOAT`
- `TIME_TRUNCATE_FRACTIONAL`

Temporal types are supported only with strict SQL mode. See Strict SQL Mode.

What's Next

- Learn how to perform the following tasks:
  - Run queries
  - Modify tables
  - Retrieve MySQL HeatWave Metadata
  - Optimize workloads for OLAP and OLTP
  - Monitor MySQL HeatWave processes

5.3 Supported Functions and Operators

This section describes functions and operators supported by MySQL HeatWave.

5.3.1 Aggregate Functions

The following table shows supported aggregate functions. The *VARLEN Support* column identifies functions that support variable-length encoded string columns. See Section 5.8.1.4, “Encoding String Columns”.

Table 5.1 Aggregate (GROUP BY) Functions

Name	VARLEN Support	Description
`AVG()`		Return the average value of the argument
`COUNT()`	Yes	Return a count of the number of rows returned
`COUNT(DISTINCT)`		Return the count of a number of different values
`GROUP_CONCAT()`	Yes	Return a concatenated string. See Section 11.2.1.4, “Functions and Operator Limitations”.
`HLL()`	Yes	Only available in MySQL HeatWave. Similar to `COUNT(DISTINCT)`, but faster and with user defined precision.
`JSON_ARRAYAGG()`		Return result set as a single JSON array. Supported as of MySQL 9.0.0.
`JSON_OBJECTAGG()`		Return result set as a single JSON object literal. Supported as of MySQL 9.0.0.
`MAX()`	Yes	Return the maximum value
`MIN()`	Yes	Return the minimum value

Name	VARLEN Support	Description
STD()		Return the population standard deviation
STDDEV()		Return the population standard deviation
STDDEV_POP()		Return the population standard deviation
STDDEV_SAMP()		Return the sample standard deviation
SUM()		Return the sum
VAR_POP()		Return the population standard variance
VAR_SAMP()		Return the sample variance
VARIANCE()		Return the population standard variance

- HLL(*expr*,[*expr*...], *precision*)

  Returns an approximate count of the number of rows with different non-NULL *expr* values. 4 ≤ *precision* ≤ 15. The default value is 10.

  If there are no matching rows, HLL() returns 0.

  As of MySQL 8.4.0, HyperLogLog, HLL(), is available in the MySQL HeatWave primary and secondary engines. Before MySQL 8.4.0, HyperLogLog, HLL(), is only available in the MySQL HeatWave secondary engine. It is similar to COUNT(DISTINCT), but with user defined precision.

  HLL() is faster than COUNT(DISTINCT). The greater the *precision*, the greater the accuracy, and the greater the amount of memory required.

  For each GROUP, HLL() requires $2^{precision}$ bytes of memory. The default value of 10 requires 1KiB of memory per GROUP.

  An example with precision set to 8 that uses 256 bytes of memory per GROUP:

```
mysql> SELECT HLL(results, 8) FROM student;
```

5.3.1.1 GROUP BY Modifiers

In the MySQL HeatWave secondary engine, the GROUP BY clause permits the following modifiers:

- A ROLLUP modifier. This is a preferred alternative to the WITH ROLLUP modifier. See ROLLUP and CUBE Modifiers.
- A CUBE modifier in addition to the WITH ROLLUP modifier. Available in MySQL HeatWave only. See ROLLUP and CUBE Modifiers.
- A GROUPING SETS modifier (as of MySQL 9.6.0). Available in MySQL HeatWave only. See GROUPING SETS Modifier.

See GROUP BY Modifiers.

ROLLUP and CUBE Modifiers

The ROLLUP modifier generates aggregated results that follow the hierarchy for the selected columns. The CUBE modifier generates aggregated results for all possible combinations of the selected columns. For a single column the results are the same.

A `ROLLUP` modifier example:

```
mysql> SELECT
         IF(GROUPING(year), 'All years', year) AS year,
         IF(GROUPING(country), 'All countries', country) AS country,
         IF(GROUPING(product), 'All products', product) AS product,
       SUM(profit) AS profit
       FROM sales
       GROUP BY ROLLUP (year, country, product);
+-----------+---------------+--------------+--------+
| year      | country       | product      | profit |
+-----------+---------------+--------------+--------+
| 2000      | Finland       | Computer     |   1500 |
| 2000      | Finland       | Phone        |    100 |
| 2000      | Finland       | All products |   1600 |
| 2000      | India         | Calculator   |    150 |
| 2000      | India         | Computer     |   1200 |
| 2000      | India         | All products |   1350 |
| 2000      | USA           | Calculator   |     75 |
| 2000      | USA           | Computer     |   1500 |
| 2000      | USA           | All products |   1575 |
| 2000      | All countries | All products |   4525 |
| 2001      | Finland       | Phone        |     10 |
| 2001      | Finland       | All products |     10 |
| 2001      | USA           | Calculator   |     50 |
| 2001      | USA           | Computer     |   2700 |
| 2001      | USA           | TV           |    250 |
| 2001      | USA           | All products |   3000 |
| 2001      | All countries | All products |   3010 |
| All years | All countries | All products |   7535 |
+-----------+---------------+--------------+--------+
```

A `CUBE` modifier example that uses the same data:

```
mysql> SELECT
         IF(GROUPING(year), 'All years', year) AS year,
         IF(GROUPING(country), 'All countries', country) AS country,
         IF(GROUPING(product), 'All products', product) AS product,
       SUM(profit) AS profit
       FROM sales
       GROUP BY CUBE (year, country, product);
+-----------+---------------+--------------+--------+
| year      | country       | product      | profit |
+-----------+---------------+--------------+--------+
| 2001      | USA           | Computer     |   2700 |
| 2000      | USA           | Computer     |   1500 |
| 2000      | India         | Calculator   |    150 |
| 2001      | USA           | TV           |    250 |
| 2000      | USA           | Calculator   |     75 |
| 2000      | Finland       | Phone        |    100 |
| 2001      | Finland       | Phone        |     10 |
| 2000      | Finland       | Computer     |   1500 |
| 2001      | USA           | Calculator   |     50 |
| 2000      | India         | Computer     |   1200 |
| All years | All countries | All products |   7535 |
| 2001      | All countries | All products |   3010 |
| 2000      | All countries | All products |   4525 |
| All years | India         | All products |   1350 |
| All years | Finland       | All products |   1610 |
| All years | USA           | All products |   4575 |
| 2001      | USA           | All products |   3000 |
| 2000      | India         | All products |   1350 |
| 2000      | Finland       | All products |   1600 |
| 2000      | USA           | All products |   1575 |
| 2001      | Finland       | All products |     10 |
| All years | All countries | TV           |    250 |
| All years | All countries | Computer     |   6900 |
| All years | All countries | Phone        |    110 |
| All years | All countries | Calculator   |    275 |
| 2001      | All countries | Computer     |   2700 |
| 2000      | All countries | Phone        |    100 |
| 2000      | All countries | Calculator   |    225 |
```

```
| 2001      | All countries | Phone        |     10 |
| 2001      | All countries | TV           |    250 |
| 2001      | All countries | Calculator   |     50 |
| 2000      | All countries | Computer     |   4200 |
| All years | Finland       | Computer     |   1500 |
| All years | USA           | Calculator   |    125 |
| All years | USA           | TV           |    250 |
| All years | USA           | Computer     |   4200 |
| All years | India         | Calculator   |    150 |
| All years | Finland       | Phone        |    110 |
| All years | India         | Computer     |   1200 |
+-----------+---------------+--------------+--------+
```

GROUPING SETS Modifier

Available as of MySQL 9.6.0 only on MySQL HeatWave, the `GROUPING SETS` modifier allows you to concisely specify multiple groupings of data in a single query. The modifier avoids the need to specify multiple `GROUP BY` queries combined with `UNION ALL`.

Use the following syntax:

```
SELECT
    ..
FROM
    table_name
GROUP BY
    GROUPING SETS <left paren>
        (grouping_set_1),
        (grouping_set_2),
        ..,
        ()
    <right paren>;
```

Each grouping set is a comma-separated list of `GROUP BY` columns or expressions. The elements inside each grouping set, and the grouping sets themselves, are comma-separated.

Grouping sets can have one or more grouping sets, up to a maximum of 128 branches.

You can also specify empty grouping sets. Consider the following example:

```
mysql> SELECT MAX(id)
FROM t1
GROUP BY GROUPING SETS (());
```

The `()` represents an empty grouping set, which means the query creates a full-table aggregate.

For example, suppose that a `sales` table has `year`, `country`, `product`, and `profit` columns for recording sales profitability:

```
CREATE TABLE sales(
sales_id INT PRIMARY KEY,
year INT,
country VARCHAR(20),
product VARCHAR(32),
profit INT
);
```

You can use grouping sets to group the data by country and product, by country only, and by product only. The empty grouping set represents an aggregate of the full table. For example:

```
mysql> SELECT country, product, SUM(profit) AS total_profit
FROM sales
GROUP BY GROUPING SETS (
  (country, product),
  (country),
  (product),
  ()
);
+---------+---------+--------------+
```

```
| country | product | total_profit |
+---------+---------+--------------+
| Canada  | Widget  |          200 |
| USA     | Widget  |          220 |
| Canada  | Gadget  |          180 |
| USA     | Gadget  |          150 |
| USA     | NULL    |          370 |
| Canada  | NULL    |          380 |
| NULL    | Gadget  |          330 |
| NULL    | Widget  |          420 |
| NULL    | NULL    |          750 |
+---------+---------+--------------+
```

The output creates additional rows with `NULL` values to provide aggregate values from the grouping sets. The output from the previous example summarizes the following:

- The row that has `country` of `USA` and `product` of `NULL` shows an aggregate total profit value for the USA of `370`.
- The row that has `country` of `Canada` and `product` of `NULL` shows an aggregate total profit value for Canada of `380`.
- The row that has `country` of `NULL` and `product` of `Gadget` shows an aggregate total profit value for gadgets of `330`.
- The row that has `country` of `NULL` and `product` of `Widget` shows an aggregate total profit value for widgets of `420`.
- The row that has `country` and `product` of `NULL` shows an overall aggregate total profit value of `750`.

When using `GROUPING SETS`, the set of expressions or columns within each grouping set must be unique. For example, the following command generates an error because `col_1` is duplicated in the first grouping set:

```
mysql> SELECT COUNT(*)
FROM table_1
GROUP BY GROUPING SETS ((col_1, col_1), (col_1,col_2));
```

Since the `GROUPING_SETS` modifier is only supported in MySQL HeatWave, the following query generates an error if the data is not loaded into MySQL HeatWave Cluster:

```
mysql> SELECT COUNT(*)
FROM table_1
GROUP BY GROUPING SETS ((col_1), (col_2));
```

However, if the `GROUPING_SETS` modifier is only an empty grouping, or is a single grouping set, it can run for data not loaded into MySQL HeatWave Cluster. Consider these examples:

```
mysql> SELECT COUNT(*)
FROM table_1
GROUP BY GROUPING SETS ((()));

mysql> SELECT COUNT(*)
FROM table_1
GROUP BY GROUPING SETS ((col_1));

mysql> SELECT COUNT(*)
FROM table_1
GROUP BY GROUPING SETS ((col_1, col_2));
```

What's Next

- Learn how to perform the following tasks:
  - Run queries

- Modify tables
- Retrieve MySQL HeatWave Metadata
- Optimize workloads for OLAP and OLTP
- Monitor MySQL HeatWave processes

5.3.2 Arithmetic Operators

The following table shows supported arithmetic operators. All arithmetic operators are supported with variable-length encoded string columns, see Section 5.8.1.4, “Encoding String Columns”.

Table 5.2 Arithmetic Operators

Name	Description
`DIV`	Integer division
`/`	Division operator
`-`	Minus operator
`%`, `MOD`	Modulo operator
`+`	Addition operator
`*`	Multiplication operator
`-`	Change the sign of the argument

What's Next

- Learn how to perform the following tasks:
  - Run queries
  - Modify tables
  - Retrieve MySQL HeatWave Metadata
  - Optimize workloads for OLAP and OLTP
  - Monitor MySQL HeatWave processes

5.3.3 Cast Functions and Operators

The following operations are supported with the `CAST()` function.

- `CAST()` from and to all the MySQL HeatWave supported numeric, temporal, string and text data types. See Section 4.2.1, “Supported Data Types for MySQL HeatWave”.
- `CAST()` of `ENUM` columns to `CHAR`, `DECIMAL`, `FLOAT`, and to `SIGNED` and `UNSIGNED` numeric values. `CAST()` operates on the `ENUM` index rather than the `ENUM` values.

See Section 11.2.1.4, “Functions and Operator Limitations”.

What's Next

- Learn how to perform the following tasks:
  - Run queries
  - Modify tables

- Retrieve MySQL HeatWave Metadata
- Optimize workloads for OLAP and OLTP
- Monitor MySQL HeatWave processes

5.3.4 Comparison Functions and Operators

The following table shows supported comparison functions and operators. The *VARLEN Support* column identifies functions and operators that support variable-length encoded string columns. See Section 5.8.1.4, “Encoding String Columns”.

Table 5.3 Comparison Functions and Operators

Name	VARLEN Support	Description
`BETWEEN ... AND ...`	Yes	Check whether a value is within a range of values
`COALESCE()`	Yes	Return the first non-NULL argument. Not supported as a `JOIN` predicate.
`=`	Yes	Equal operator
`<=>`		NULL-safe equal to operator
`>`	Yes	Greater than operator
`>=`	Yes	Greater than or equal operator
`GREATEST()`	Yes	Return the largest argument
`IN()`	Yes	Check whether a value is within a set of values. *`expr IN (value,...)`* comparisons where the expression is a single value and compared values are constants of the same data type and encoding are optimized for performance. For example, the following `IN()` comparison is optimized: `SELECT * FROM Customers WHERE Country IN ('Germany', 'France', 'Spain');`
`IS`		Test a value against a boolean
`IS NOT`		Test a value against a boolean
`IS NOT NULL`	Yes	NOT NULL value test
`IS NULL`	Yes	NULL value test
`ISNULL()`		Test whether the argument is NULL
`LEAST()`	Yes	Return the smallest argument
`<`	Yes	Less than operator
`<=`	Yes	Less than or equal operator
`LIKE`	Yes	Simple pattern matching

Name	VARLEN Support	Description
`NOT BETWEEN ... AND ...`	Yes	Check whether a value is not within a range of values
`!=`, `<>`	Yes	Not equal operator
`NOT IN()`	Yes	Check whether a value is not within a set of values
`NOT LIKE`	Yes	Negation of simple pattern matching
`STRCMP()`	Yes	Compare two strings.

What's Next

- Learn how to perform the following tasks:
  - Run queries
  - Modify tables
  - Retrieve MySQL HeatWave Metadata
  - Optimize workloads for OLAP and OLTP
  - Monitor MySQL HeatWave processes

5.3.5 Control Flow Functions and Operators

The following table shows supported control flow operators. The *VARLEN Support* column identifies functions and operators that support variable-length encoded string columns. See Section 5.8.1.4, “Encoding String Columns”.

Table 5.4 Control Flow Functions and Operators

Name	VARLEN Support	Description
`CASE`	Yes	Case operator
`IF()`	Yes	If/else construct
`IFNULL()`	Yes	Null if/else construct
`NULLIF()`	Yes	Return NULL if expr1 = expr2

What's Next

- Learn how to perform the following tasks:
  - Run queries
  - Modify tables
  - Retrieve MySQL HeatWave Metadata
  - Optimize workloads for OLAP and OLTP
  - Monitor MySQL HeatWave processes

5.3.6 Data Masking and De-Identification Functions

The following table shows data masking and de-identification functions supported by MySQL HeatWave.

Table 5.5 Data Masking and De-Identification Functions

Name	Description
`gen_range()`	Generate random number within range
`gen_rnd_email()`	Generate random email address
`gen_rnd_pan()`	Generate random payment card Primary Account Number
`gen_rnd_ssn()`	Generate random US Social Security number
`gen_rnd_us_phone()`	Generate random US phone number
`mask_inner()`	Mask interior part of string
`mask_outer()`	Mask left and right parts of string
`mask_pan()`	Mask payment card Primary Account Number part of string
`mask_pan_relaxed()`	Mask payment card Primary Account Number part of string
`mask_ssn()`	Mask US Social Security number

What's Next

- Learn how to perform the following tasks:
  - Run queries
  - Modify tables
  - Retrieve MySQL HeatWave Metadata
  - Optimize workloads for OLAP and OLTP
  - Monitor MySQL HeatWave processes

5.3.7 Encryption and Compression Functions

The following table shows supported encryption and compression functions. Encryption and compression functions are supported with variable-length columns. These functions are not supported with dictionary-encoded columns.

Table 5.6 Encryption and Compression Functions

Name	Description
`COMPRESS()`	Return the result as a binary string.
`MD5()`	Calculate the MD5 checksum. **Note** Deprecated as of MySQL 9.4.0, and removed as of MySQL 9.6.0. `SHA2()` is recommended instead.
`RANDOM_BYTES()`	Return a random byte vector.
`SHA()`, `SHA1()`	Calculate an SHA-1 160-bit checksum. **Note** Deprecated as of MySQL 9.4.0, and removed as of

Name	Description
	MySQL 9.6.0. SHA2() is recommended instead.
SHA2()	Calculate an SHA-2 checksum.
UNCOMPRESS()	Uncompress a compressed string.
UNCOMPRESSED_LENGTH()	Return the length of a string before compression.

What's Next

- Learn how to perform the following tasks:
  - Run queries
  - Modify tables
  - Retrieve MySQL HeatWave Metadata
  - Optimize workloads for OLAP and OLTP
  - Monitor MySQL HeatWave processes

5.3.8 JSON Functions

The following table shows supported JSON functions.

Table 5.7 JSON Functions

Name	Description
->	Return value from JSON column after evaluating path; equivalent to JSON_EXTRACT().
->>	Return value from JSON column after evaluating path and unquoting the result; equivalent to JSON_UNQUOTE(JSON_EXTRACT()).
JSON_ARRAY()	Create JSON array
JSON_ARRAY_APPEND()	Append data to JSON document
JSON_ARRAY_INSERT()	Insert into JSON array
JSON_CONTAINS()	Whether JSON document contains specific object at path. Supported as of MySQL 9.0.0.
JSON_CONTAINS_PATH()	Whether JSON document contains any data at path. Supported as of MySQL 9.0.0.
JSON_DEPTH()	Maximum depth of JSON document
JSON_EXTRACT()	Return data from JSON document
JSON_INSERT()	Insert or update data into JSON document
JSON_KEYS()	Array of keys from JSON document. Supported as of MySQL 9.0.0.
JSON_LENGTH()	Number of elements in JSON document
JSON_MERGE()	Merge JSON documents, preserving all values of duplicate keys. Deprecated synonym for JSON_MERGE_PRESERVE()
JSON_MERGE_PATCH()	Merge JSON documents, replacing values of duplicate keys

Name	Description
JSON_MERGE_PRESERVE()	Merge JSON documents, preserving duplicate keys
JSON_OBJECT()	Create JSON object literal
JSON_OVERLAPS()	Compares two JSON documents, returns TRUE (1) if these have any key-value pairs or array elements in common, otherwise FALSE (0). Supported as of MySQL 9.0.0.
JSON_PRETTY()	Print a JSON document in human-readable format. Supported as of MySQL 9.0.0.
JSON_QUOTE()	Quote JSON document. Supported as of MySQL 9.0.0.
JSON_REMOVE()	Remove data from JSON document
JSON_REPLACE()	Replace values in JSON document
JSON_SCHEMA_VALID()	Validate JSON document against JSON schema; returns TRUE or 1 if document validates against schema, or FALSE or 0 if it does not
JSON_SCHEMA_VALIDATION_REPORT()	Validate JSON document against JSON schema; returns report in JSON format on outcome on validation including success or failure and reasons for failure
JSON_SEARCH()	Path to value within JSON document. Supported as of MySQL 9.0.0.
JSON_SET()	Insert data into JSON document
JSON_STORAGE_FREE()	Freed space within binary representation of JSON column value following partial update. Supported as of MySQL 9.0.0.
JSON_STORAGE_SIZE()	Space used for storage of binary representation of a JSON document. Supported as of MySQL 9.0.0.
JSON_TABLE()	Return data from a JSON expression as a relational table. Supported as of MySQL 9.2.0. See: Section 11.2.1.4, “Functions and Operator Limitations”.
JSON_TYPE()	Type of JSON value. Supported as of MySQL 9.0.0.
JSON_UNQUOTE()	Unquote JSON value
JSON_VALID()	Whether JSON value is valid. Supported as of MySQL 9.0.0.
JSON_VALUE()	Extract value from JSON document at location pointed to by path provided; return this value as VARCHAR(512) or specified type. Supported as of MySQL 9.0.0.
MEMBER OF()	Returns TRUE (1) if first operand matches any element of JSON array passed as second operand, otherwise returns FALSE (0). Supported as of MySQL 9.0.0.

What's Next

- Learn how to perform the following tasks:

- Run queries
- Modify tables
- Retrieve MySQL HeatWave Metadata
- Optimize workloads for OLAP and OLTP
- Monitor MySQL HeatWave processes

5.3.9 Logical Operators

The following table shows supported logical operators.

Table 5.8 Logical Operators

Name	Description
`AND`, `&&`	Logical AND
`NOT`, `!`	Negates value
`||`, `OR`	Logical OR
`XOR`	Logical XOR

What's Next

- Learn how to perform the following tasks:
  - Run queries
  - Modify tables
  - Retrieve MySQL HeatWave Metadata
  - Optimize workloads for OLAP and OLTP
  - Monitor MySQL HeatWave processes

5.3.10 Mathematical Functions

The following table shows supported mathematical functions. These are not supported with dictionary-encoded columns.

Table 5.9 Mathematical Functions

Name	Description
`ABS()`	Return the absolute value.
`ACOS()`	Return the arc cosine.
`ASIN()`	Return the arc sine.
`ATAN()`	Return the arc tangent.
`ATAN2()`, `ATAN()`	Return the arc tangent of the two arguments
`CEIL()`	Return the smallest integer value not less than the argument. The function is not applied to `BIGINT` values. The input value is returned. `CEIL()` is a synonym for `CEILING()`.

Name	Description
CEILING()	Return the smallest integer value not less than the argument. The function is not applied to BIGINT values. The input value is returned. CEILING() is a synonym for CEIL().
COS()	Return the cosine.
COT()	Return the cotangent.
CRC32()	Compute a cyclic redundancy check value.
DEGREES()	Convert radians to degrees.
EXP()	Raise to the power of.
FLOOR()	Return the largest integer value not greater than the argument. The function is not applied to BIGINT values. The input value is returned.
LN()	Return the natural logarithm of the argument.
LOG()	Return the natural logarithm of the first argument.
LOG10()	Return the base-10 logarithm of the argument.
LOG2()	Return the base-2 logarithm of the argument.
MOD()	Return the remainder.
PI()	Return the value of pi
POW()	Return the argument raised to the specified power
POWER()	Return the argument raised to the specified power.
RADIANS()	Return argument converted to radians.
RAND()	Return a random floating-point value.
ROUND()	Round the argument.
SIGN()	Return the sign of the argument.
SIN()	Return the sine of the argument.
SQRT()	Return the square root of the argument.
TAN()	Return the tangent of the argument.
TRUNCATE()	Truncate to specified number of decimal places.

What's Next

- Learn how to perform the following tasks:
  - Run queries
  - Modify tables
  - Retrieve MySQL HeatWave Metadata
  - Optimize workloads for OLAP and OLTP
  - Monitor MySQL HeatWave processes

5.3.11 String Functions and Operators

The following table shows supported string functions and operators. With the exception of the FORMAT() function, string functions and operators described in the following table are supported with variable-length encoded columns. Dictionary encoded columns are not supported.

Table 5.10 String Functions and Operators

Name	Description
ASCII()	Return numeric value of left-most character
BIN()	Return a string containing binary representation of a number
BIT_LENGTH()	Return length of argument in bits
CHAR_LENGTH()	Return number of characters in argument
CONCAT()	Return concatenated string
CONCAT_WS()	Return concatenated with separator
ELT()	Return string at index number
EXPORT_SET()	Return a string such that for every bit set in the value bits, it returns an on string (and for every unset bit, it returns an off string).
FIELD()	Index (position) of first argument in subsequent arguments
FIND_IN_SET()	Index (position) of first argument within second argument
FORMAT()	Return a number formatted to specified number of decimal places. *Does not support variable-length-encoded columns.*
FROM_BASE64()	Decode base64 encoded string and return result
GREATEST()	Return the largest argument
HEX()	Hexadecimal representation of decimal or string value
INSERT()	Return the index of the first occurrence of substring
INSTR()	Return the index of the first occurrence of substring
LEAST()	Return the smallest argument
LEFT()	Return the leftmost number of characters as specified
LENGTH()	Return the length of a string in bytes
LIKE	Simple pattern matching
LOCATE()	Return the position of the first occurrence of substring
LOWER()	Return the argument in lowercase
LPAD()	Return the string argument, left-padded with the specified string
LTRIM()	Remove leading spaces
MAKE_SET()	Return a set of comma-separated strings that have the corresponding bit in bits
MID()	Return a substring starting from the specified position
NOT LIKE	Negation of simple pattern matching
OCTET_LENGTH()	Synonym for LENGTH()

Name	Description
ORD()	Return character code for leftmost character of the argument
POSITION()	Synonym for LOCATE()
QUOTE()	Escape the argument for use in an SQL statement
REGEXP	Whether string matches regular expression
REGEXP_INSTR()	Starting index of substring matching regular expression
REGEXP_LIKE()	Whether string matches regular expression
REGEXP_REPLACE()	Replace substrings matching regular expression. Supports up to three arguments.
REGEXP_SUBSTR()	Return substring matching regular expression. Supports up to three arguments.
REPEAT()	Repeat a string the specified number of times
REPLACE()	Replace occurrences of a specified string
REVERSE()	Reverse the characters in a string
RIGHT()	Return the specified rightmost number of characters
RLIKE	Whether string matches regular expression
RPAD()	Append string the specified number of times
RTRIM()	Remove trailing spaces
SOUNDEX()	Return a soundex string
SPACE()	Return a string of the specified number of spaces
STRCMP()	Compare two strings
SUBSTR()	Return the substring as specified
SUBSTRING()	Return the substring as specified
SUBSTRING_INDEX()	Return a substring from a string before the specified number of occurrences of the delimiter
TO_BASE64()	Return the argument converted to a base-64 string
TRIM()	Remove leading and trailing spaces
UNHEX()	Return a string containing hex representation of a number
UPPER()	Convert to uppercase
WEIGHT_STRING()	Return the weight string for a string

What's Next

- Learn how to perform the following tasks:
  - Run queries
  - Modify tables
  - Retrieve MySQL HeatWave Metadata
  - Optimize workloads for OLAP and OLTP
  - Monitor MySQL HeatWave processes

5.3.12 Temporal Functions

The following table shows supported temporal functions.

As of MySQL 8.4.0, MySQL HeatWave supports named time zones such as `MET` or `Europe/Amsterdam` for `CONVERT_TZ()`. For a workaround before MySQL 8.4.0, see Section 11.2.1.4, "Functions and Operator Limitations".

All functions support variable-length encoded string columns. See Section 5.8.1.4, "Encoding String Columns".

Table 5.11 Temporal Functions

Name	Description
`ADDDATE()`	Add time values (intervals) to a date value
`ADDTIME()`	Add time
`CONVERT_TZ()`	Convert from one time zone to another.
`CURDATE()`	Return the current date
`CURRENT_DATE()`, `CURRENT_DATE`	Synonyms for `CURDATE()`
`CURRENT_TIME()`, `CURRENT_TIME`	Synonyms for `CURTIME()`
`CURRENT_TIMESTAMP()`, `CURRENT_TIMESTAMP`	Synonyms for `NOW()`
`CURTIME()`	Return the current time
`DATE()`	Extract the date part of a date or datetime expression
`DATE_ADD()`	Add time values (intervals) to a date value
`DATE_FORMAT()`	Format date as specified
`DATE_SUB()`	Subtract a time value (interval) from a date
`DATEDIFF()`	Subtract two dates
`DAY()`	Synonym for `DAYOFMONTH()`
`DAYNAME()`	Return the name of the weekday
`DAYOFMONTH()`	Return the day of the month (0-31)
`DAYOFWEEK()`	Return the weekday index of the argument
`DAYOFYEAR()`	Return the day of the year (1-366)
`EXTRACT()`	Extract part of a date
`FROM_DAYS()`	Convert a day number to a date
`FROM_UNIXTIME()`	Format Unix timestamp as a date
`GET_FORMAT()`	Return a date format string
`HOUR()`	Extract the hour
`LAST_DAY()`	Return the last day of the month for the argument
`LOCALTIME()`, `LOCALTIME`	Synonym for `NOW()`
`LOCALTIMESTAMP`, `LOCALTIMESTAMP()`	Synonym for `NOW()`
`MAKEDATE()`	Create a date from the year and day of year. Supports `FLOAT`, `DOUBLE`, `INTEGER`, and `YEAR` data types.
`MAKETIME()`	Create time from hour, minute, second
`MICROSECOND()`	Return the microseconds from argument
`MINUTE()`	Return the minute from the argument

Name	Description
MONTH()	Return the month from the date passed
MONTHNAME()	Return the name of the month
NOW()	Return the current date and time
PERIOD_ADD()	Add a period to a year-month
PERIOD_DIFF()	Return the number of months between periods
QUARTER()	Return the quarter from a date argument
SEC_TO_TIME()	Converts seconds to 'HH:MM:SS' format
SECOND()	Return the second (0-59)
STR_TO_DATE()	Convert a string to a date
SUBDATE()	Synonym for DATE_SUB() when invoked with three arguments
SUBTIME()	Subtract times
SYSDATE()	Return the time at which the function executes
TIME()	Extract the time portion of the expression passed
TIME_FORMAT()	Format as time.
TIME_TO_SEC()	Return the argument converted to seconds
TIMEDIFF()	Subtract time
TIMESTAMP()	With a single argument, this function returns the date or datetime expression; with two arguments, the sum of the arguments
TIMESTAMPADD()	Add an interval to a datetime expression
TIMESTAMPDIFF()	Subtract an interval from a datetime expression
TO_DAYS()	Return the date argument converted to days
TO_SECONDS()	Return the date or datetime argument converted to seconds since Year 0
UNIX_TIMESTAMP()	Return a Unix timestamp
UTC_DATE()	Return the current UTC date
UTC_TIME()	Return the current UTC time
UTC_TIMESTAMP()	Return the current UTC date and time
WEEK()	Return the week number. Restrictions apply. See Section 11.2.1.4, “Functions and Operator Limitations”
WEEKDAY()	Return the weekday index
WEEKOFYEAR()	Return the calendar week of the date (1-53)
YEAR()	Return the year
YEARWEEK()	Return the year and week

What's Next

- Learn how to perform the following tasks:
  - Run queries
  - Modify tables
  - Retrieve MySQL HeatWave Metadata

- Optimize workloads for OLAP and OLTP
- Monitor MySQL HeatWave processes

5.3.13 Vector Functions

The following table shows supported vector functions. These are not supported with dictionary-encoded columns. These were introduced in MySQL 9.0.0.

Table 5.12 Vector Functions

Name	Description
DISTANCE()	Calculates the distance between two vectors with the specified method
VECTOR_TO_STRING()	Get the string representation of a VECTOR column, given its value as a binary string
FROM_VECTOR()	A synonym for VECTOR_TO_STRING()
TO_VECTOR()	A synonym for STRING_TO_VECTOR()
VECTOR_DIM()	Get the length of a vector, that is, the number of entries it contains
VECTOR_DISTANCE	A synonym for DISTANCE()
VECTOR_TO_STRING()	Get the string representation of a VECTOR column, given its value as a binary string

- DISTANCE(*vector*, *vector*, *string*)

  Calculates the distance between two vectors per the specified calculation method. It accepts the following arguments:

  - A column of VECTOR data type.
  - An input query of VECTOR data type.
  - A string that specifies the distance metric. The supported values are COSINE, DOT, and EUCLIDEAN. The argument is a string, and requires quotation marks.

  Examples:

```
mysql> SELECT DISTANCE(STRING_TO_VECTOR("[1.01231, 2.0123123, 3.0123123, 4.01231231]"), STRING_TO_VECTOR(
+-----------------------------------------------------------------------------------------------------------
| DISTANCE(STRING_TO_VECTOR("[1.01231, 2.0123123, 3.0123123, 4.01231231]"), STRING_TO_VECTOR("[1, 2, 3, 4
+-----------------------------------------------------------------------------------------------------------
|                                                                                                 0.00000166
+-----------------------------------------------------------------------------------------------------------
```

What's Next

- Learn how to perform the following tasks:
  - Run queries
  - Modify tables
  - Retrieve MySQL HeatWave Metadata
  - Optimize workloads for OLAP and OLTP
  - Monitor MySQL HeatWave processes

5.3.14 Window Functions

This topic describes MySQL HeatWave window function support. For optimal performance, window functions in MySQL HeatWave utilize a massively parallel, partitioning-based algorithm. For general information about window functions, see Window Functions, in the *MySQL Reference Manual*.

MySQL HeatWave window function support includes support for:

- `WINDOW` and `OVER` clauses in conjunction with `PARTITION BY`, `ORDER BY`, and `WINDOW` frame specifications.
- Nonaggregate window functions supported by MySQL Server, as described in Window Function Descriptions.
- The following aggregate functions used as window functions:
  - `AVG()`
  - `COUNT()`
  - `MIN()`
  - `MAX()`
  - `STD()`.
  - `STDDEV()`.
  - `STDDEV_POP()`.
  - `STDDEV_SAMP()`.
  - `SUM()`
  - `VAR_POP()`.
  - `VAR_SAMP()`.
  - `VARIANCE()`.

What's Next

- Learn how to perform the following tasks:
  - Run queries
  - Modify tables
  - Retrieve MySQL HeatWave Metadata
  - Optimize workloads for OLAP and OLTP
  - Monitor MySQL HeatWave processes

5.3.15 Miscellaneous Functions

The following table shows supported miscellaneous functions.

Table 5.13 Miscellaneous Functions

Name	Description
ETAG()	Compute hash for each row, using one or more column or other values. See Section 11.1.1, “MySQL HeatWave Issues”.

What's Next

- Learn how to perform the following tasks:
  - Run queries
  - Modify tables
  - Retrieve MySQL HeatWave Metadata
  - Optimize workloads for OLAP and OLTP
  - Monitor MySQL HeatWave processes

5.4 Run Queries

When MySQL HeatWave is enabled and the data you want to query is loaded, queries that qualify are automatically offloaded from the DB System to MySQL HeatWave for accelerated processing. No special action is required. Simply run the query from a client, application, or interface that is connected to the DB System associated with the MySQL HeatWave Cluster. After MySQL HeatWave processes a query, results are sent back to the DB System and to the client, application, or interface that issued the query.

As of MySQL 9.1.0, MySQL HeatWave supports InnoDB partitions. Query processing in MySQL HeatWave can access partitions with standard syntax.

For information about connecting to a DB System on MySQL HeatWave Service:

- For MySQL HeatWave on OCI see Connecting to a DB System in the *MySQL HeatWave on OCI Service Guide*.
- For MySQL HeatWave on AWS, see Connecting from a Client in the *MySQL HeatWave on AWS Service Guide*.

5.4.1 Requirements for Running Queries

Ensure that the following requirements are met before offloading queries to MySQL HeatWave:

- Use a SELECT statement. INSERT ... SELECT and CREATE TABLE ... SELECT statements are supported, but only the SELECT portion is offloaded to MySQL HeatWave.

  See Section 5.4.4, “CREATE TABLE ... SELECT Statements”, and Section 5.4.6, “INSERT ... SELECT Statements”.

- Define all tables accessed by the query using RAPID as the secondary engine.
- Load all accessed tables in MySQL HeatWave.
- Enable autocommit. If autocommit is disabled, MySQL HeatWave does not offload the query and execution is performed on the DB System. To check the autocommit setting:

```
mysql> SHOW VARIABLES LIKE 'autocommit';
+--------------+-------+
| Variable_name | Value |
+--------------+-------+
| autocommit    | ON    |
```

```
+---------------+-------+
```

- To execute queries using the MySQL HeatWave secondary engine, set the `use_secondary_engine` system variable to either `ON` or `FORCED` state using one of the following syntax options. By default, this variable is set to `ON`.
  - `ON`: Queries are executed using the secondary engine only if supported by MySQL HeatWave (secondary_engine). If MySQL HeatWave execution is not possible for a query due to unsupported syntax or any other reason as mentioned in Diagnose Query Offload, the query is executed by the "DB System", and no errors are generated.
  - `FORCED`: Queries are executed exclusively using the secondary engine. If the query is not supported by MySQL HeatWave, it is not executed, and an error is be returned.

```
mysql> SET SESSION use_secondary_engine = 'ON';
Query OK, 0 rows affected (0.2831 sec)
```

```
mysql> SET SESSION use_secondary_engine = 'FORCED';
Query OK, 0 rows affected (0.3451 sec)
```

- Ensure that queries use only functions and operators supported by MySQL HeatWave. See Section 5.3, “Supported Functions and Operators”.
- Ensure that queries use only SQL modes supported by MySQL HeatWave.
- Queries should not include any known limitations.

If any prerequisite is not satisfied, the query is not offloaded and falls back to the DB System for processing by default.

What's Next

- Learn how to perform the following tasks:
  - Modify tables
  - Retrieve MySQL HeatWave Metadata
  - Optimize workloads for OLAP and OLTP
  - Monitor MySQL HeatWave processes

5.4.2 SELECT Statement Clauses for MySQL HeatWave

This topic provides information related to the `SELECT` statement clauses supported in MySQL HeatWave and describes how to use the `GROUP BY` and `QUALIFY` clauses to group or filter the processing results.

Before You Begin

- Review the requirements.
- Connect to your DB System.
- Verify if the query is offloaded to MySQL HeatWave for processing.

SELECT STATEMENT Clauses

The following `SELECT` statement clauses are supported in the MySQL HeatWave secondary engine.

- `GROUP BY` clause
  - A `CUBE` modifier in addition to the `WITH ROLLUP` modifier. `CUBE` modifier is available only in secondary engine and is not supported in the DB System.

  - A `ROLLUP` modifier as a preferred alternative to the `WITH ROLLUP` modifier.
- `QUALIFY` clause is added between the `WINDOW` clause and the `ORDER BY` clause. `QUALIFY` clause is available only in secondary engine and is not supported in the DB System.

  In addition to constraints similar to the `HAVING` clause, the `QUALIFY` clause can also include predicates related to a window function.

  Similar to the `HAVING` clause, the `QUALIFY` clause can refer to aliases mentioned in the `SELECT` list.

  The `QUALIFY` clause requires the inclusion of at least one window function in the query. The window function can be part of any one of the following:

  - The `SELECT` column list.
  - A filter predicate of the `QUALIFY` clause.
- `TABLESAMPLE` clause is available only in secondary engine and is not supported in the DB System.

SELECT STATEMENT Syntax

The following `SELECT` statement syntax shows the clauses that are supported in the MySQL HeatWave secondary engine.

```
SELECT
    [ALL | DISTINCT | DISTINCTROW ]
    [STRAIGHT_JOIN]
    [SQL_BUFFER_RESULT]
    select_expr [, select_expr] ...
    [FROM table_references
      [PARTITION partition_list]
   [TABLESAMPLE { SYSTEM | BERNOULLI } (sample_percentage)]]
    [WHERE where_condition]
    [GROUP BY {{col_name | expr | position}, ... [WITH ROLLUP]
         | {CUBE | ROLLUP} ({col_name | expr | position}, ...)}]
    [HAVING where_condition]
    [WINDOW window_name AS (window_spec)
        [, window_name AS (window_spec)] ...]
    [QUALIFY qualify_condition]
    [ORDER BY {col_name | expr | position} [ASC | DESC], ...]
    [LIMIT {[offset,] row_count | row_count OFFSET offset}]
```

For a full explanation of the `SELECT` syntax, see SELECT Statement.

MySQL HeatWave uses a query optimization technique where some conditions from `HAVING` clause are moved earlier in the execution pipeline to reduce the amount of data processed. This helps reduce the intermediate result set sizes and identify data-related issues that might go unnoticed in MySQL. However, this optimization technique changes the order of operations performed as compared to a standard SQL execution of the `SELECT` statement on MySQL.

What's Next

Learn how to explain a query and check query execution time.

5.4.3 Explain Query and Check Execution Time

This topic describes how to use the `EXPLAIN` statement to analyze query execution and verify whether a query is offloaded to MySQL HeatWave for accelerated processing.

Before You Begin

- Review the requirements.
- Connect to your DB System.

- The examples in this topic use the sample database `airportdb`. To learn how to download the sample database, see AirportDB Analytics Quickstart.

Execute EXPLAIN Statement

Before running a query, you can use `EXPLAIN` to determine if the query is offloaded to MySQL HeatWave for processing. If so, the `Extra` column of `EXPLAIN` output includes `Using secondary engine RAPID`.

If `Using secondary engine RAPID` does not appear in the `Extra` column, the query is not offloaded to MySQL HeatWave. To determine why a query is not offload, refer to Diagnose Query Offload Errors or MySQL HeatWave Issues to diagnose and resolve the offloading error.

```
mysql> EXPLAIN SELECT airline_id,
          COUNT(*) AS flight_id FROM flight
          WHERE flight_id <= 66810
          GROUP BY airline_id
          ORDER BY airline_id;

*************************** 1. row ***************************
           id: 1
  select_type: NONE
        table: NULL
   partitions: NULL
         type: NULL
possible_keys: NULL
          key: NULL
      key_len: NULL
          ref: NULL
         rows: NULL
     filtered: NULL
        Extra: Using secondary engine RAPID. Use EXPLAIN FORMAT=TREE to show the plan.
1 row in set, 1 warning (0.2521 sec)
```

After using `EXPLAIN` to verify that the query can be offloaded, run the query and note the execution time.

```
mysql> SELECT airline_id, COUNT(*) AS flight_id
          FROM flight
          WHERE flight_id <= 66810
          GROUP BY airline_id
          ORDER BY airline_id;
+------------+-----------+
| airline_id | flight_id |
+------------+-----------+
|          1 |       436 |
|          2 |       430 |
|...         |           |
+------------+-----------+
113 rows in set (0.2953 sec)
```

To compare MySQL HeatWave query execution time with DB System execution time, disable the `use_secondary_engine` variable and run the query again to see how long it takes to run on the DB System.

```
mysql> SET SESSION use_secondary_engine=OFF;

mysql> SELECT airline_id, COUNT(*) AS flight_id
          FROM flight
          WHERE flight_id <= 66810
          GROUP BY airline_id
          ORDER BY airline_id;
+------------+-----------+
| airline_id | flight_id |
+------------+-----------+
|          1 |       436 |
|          2 |       430 |
|...         |           |
+------------+-----------+
```

```
113 rows in set (0.4476 sec)
```

Note

Concurrently issued queries are prioritized for execution. For information about query prioritization, see Section 5.4.11, “Auto Scheduling”.

What's Next

Learn how to use Create Table ... SELECT Statements.

5.4.4 CREATE TABLE ... SELECT Statements

This topic describes how to create a new table and insert data using a `SELECT` query.

When the query is offloaded to MySQL HeatWave, it is processed faster, improving the overall performance of the operation.

Note

However, the `CREATE TABLE ... SELECT` statement is not permitted when the `sql_require_primary_key` system variable is enabled. This setting is enabled by default in all high availability database configurations.

Before You Begin

- Review the requirements.
- Connect to your DB System.
- Verify if the query is offloaded to MySQL HeatWave for processing.
- Ensure that the `SELECT` table is loaded in MySQL HeatWave.
- The examples in this topic use the sample database `airportdb`. To learn how to download the sample database, see AirportDB Analytics Quickstart.

Execute Create TABLE ... SELECT Statement

When a `CREATE TABLE ... SELECT` statement is executed, the `SELECT` query is offloaded to MySQL HeatWave for processing, while the table is created in the DB System. This offloading speeds up the execution of long-running and complex queries. However, if the `SELECT` query returns a large result set, queries that produce large result sets performance may be limited because inserting the data into the DB System takes time.

For example, the following statement sample uses the `SELECT` statement with a subquery to find all flights where the departure airport (`from`) is located in the United Kingdom. Then, using the `CREATE TABLE ... SELECT` it creates a new table called `flight_from_UK`.

```
mysql> EXPLAIN SELECT * FROM flight WHERE `from` IN (SELECT airport_id FROM airport_geo WHERE country='UNI

*************************** 1. row ***************************
           id: 1
  select_type: NONE
        table: NULL
   partitions: NULL
         type: NULL
possible_keys: NULL
          key: NULL
      key_len: NULL
          ref: NULL
         rows: NULL
     filtered: NULL
        Extra: Using secondary engine RAPID. Use EXPLAIN FORMAT=TREE to show the plan.
1 row in set, 1 warning (0.2239 sec)
```

```
mysql> CREATE TABLE flight_from_UK SELECT * FROM flight WHERE `from` IN (SELECT airport_id FROM airport
Query OK, 10777 rows affected (0.3180 sec)

Records: 10777  Duplicates: 0  Warnings: 0
```

The `SELECT` portion of the `CREATE TABLE ... SELECT` statement is subject to the same MySQL HeatWave requirements and limitations as regular `SELECT` queries.

What's Next

Learn how to create MySQL HeatWave temporary tables.

5.4.5 Create MySQL HeatWave Temporary Tables

As of MySQL 9.3.0, you can create temporary tables that are stored in the MySQL HeatWave Cluster. A temporary table is visible only within the current session, and is dropped automatically when you close the session.

Once created, you can use a MySQL HeatWave temporary table as an independent entity. After the MySQL HeatWave temporary table is created, you can unload the table that you used to create the temporary table. If you make updates in the original table after the temporary table is created, the updates are not reflected in the temporary table.

Unlike the MySQL temporary table that exists in DB System, MySQL HeatWave temporary tables are stored in the MySQL HeatWave Cluster by the MySQL HeatWave engine and are used for MySQL HeatWave processing.

Refer to the following sections, to create a MySQL HeatWave temporary table and execute queries.

- Before You Begin
- Create a MySQL HeatWave Temporary Table
- Comparing Query Execution Times
- What's Next

Before You Begin

- To create a MySQL HeatWave temporary table, you must have the `CREATE TEMPORARY TABLES` privilege. After a session has created a temporary table, the server performs no further privilege checks on the table.
- Load the original table based on which you want to create a temporary table in the MySQL HeatWave Cluster.

  The examples in this topic use the sample database `airportdb`. To learn how to download the sample database, see AirportDB Analytics Quickstart.

Create a MySQL HeatWave Temporary Table

- As of MySQL 9.4.0, to create a MySQL HeatWave temporary table, use the `CREATE TEMPORARY TABLE` statement with the `ENGINE=RAPID` option:

  ```
  mysql> CREATE TEMPORARY TABLE tbl_name
  ENGINE=RAPID
  SELECT …;
  ```

  For example:

  ```
  mysql> CREATE TEMPORARY TABLE seat_sold_htt
  ENGINE=RAPID
  AS
  SELECT flight_id, COUNT(*) AS sold_seat
  FROM booking
  ```

```
GROUP BY flight_id;
```

For versions earlier than MySQL 9.4.0, use the `CREATE TEMPORARY TABLE` statement with the `SECONDARY_ENGINE=RAPID` option instead of the `ENGINE=RAPID` option:

```
mysql> CREATE TEMPORARY TABLE tbl_name
SECONDARY_ENGINE=RAPID
SELECT …;
```

- Alternatively, as of MySQL 9.4.0, you can set the `default_tmp_storage_engine` variable to `RAPID` to specify the primary engine for MySQL HeatWave temporary tables for the current session:

```
mysql> SET SESSION default_tmp_storage_engine='RAPID';
```

In this case, use the `CREATE TEMPORARY TABLE` statement without the `ENGINE=RAPID` option to create a MySQL HeatWave temporary table:

```
mysql> CREATE TEMPORARY TABLE tbl_name
SELECT …;
```

For example:

```
mysql> CREATE TEMPORARY TABLE seat_sold_htt
AS
SELECT flight_id, COUNT(*) AS sold_seat
FROM booking
GROUP BY flight_id;
```

The temporary table information is added to the `rpd_tables` performance schema table, and a new column `MATERIALIZATION_QUERY` is added to the `rpd_table_id` performance schema table. The information related to the query used to create the temporary table, is provided in the `MATERIALIZATION_QUERY` column.

```
mysql> SELECT TABLE_NAME, MATERIALIZATION_QUERY FROM performance_schema.rpd_table_id\G
*************************** 1. row ***************************
           TABLE_NAME: seat_sold_htt
MATERIALIZATION_QUERY: CREATE TEMPORARY TABLE seat_sold_htt
ENGINE=RAPID
AS
SELECT flight_id, COUNT(*) AS sold_seat
FROM booking
GROUP BY flight_id
```

Comparing Query Execution Times

The following examples demonstrate how to calculate load factor of each airline for which data is available in the `airportdb` database, using a derived table and a MySQL HeatWave temporary table.

Run the following query to calculate the load factor using a derived table *`seat_sold`*.

```
mysql> SELECT airlinename 'Airline Name', SUM(sold_seat)/SUM(capacity) 'Load Factor'
FROM (SELECT flight_id, COUNT(*) sold_seat FROM booking GROUP BY flight_id) seat_sold
  JOIN flight USING (flight_id)
  JOIN airplane USING (airplane_id)
  JOIN airline ON airline.airline_id = flight.airline_id
GROUP BY airlinename
ORDER BY airlinename;
+----------------------+-------------+
| Airline Name         | Load Factor |
+----------------------+-------------+
| Afghanistan Airlines |      0.4925 |
| Albania Airlines     |      0.4927 |
| American Samoa Airli |      0.4936 |
| Angola Airlines      |      0.4926 |
| Argentina Airlines   |      0.4922 |
| Australia Airlines   |      0.4921 |
| Azerbaijan Airlines  |      0.4930 |
| Bahamas Airlines     |      0.4927 |
|...                   |             |
```

```
| Zimbabwe Airlines    |      0.4934 |
+----------------------+-------------+
113 rows in set (4.8334 sec)
```

Run the following query to calculate the load factor using MySQL HeatWave temporary table *`seat_sold_htt`* created in Creating a MySQL HeatWave Temporary Table section.

```
mysql> SELECT airlinename 'Airline Name', SUM(sold_seat)/SUM(capacity) 'Load Factor'
FROM seat_sold_htt
  JOIN flight USING (flight_id)
  JOIN airplane USING (airplane_id)
  JOIN airline ON airline.airline_id = flight.airline_id
GROUP BY airlinename
ORDER BY airlinename;
+----------------------+-------------+
| Airline Name         | Load Factor |
+----------------------+-------------+
| Afghanistan Airlines |      0.4925 |
| Albania Airlines     |      0.4927 |
| American Samoa Airli |      0.4936 |
| Angola Airlines      |      0.4926 |
| Argentina Airlines   |      0.4922 |
| Australia Airlines   |      0.4921 |
| Azerbaijan Airlines  |      0.4930 |
| Bahamas Airlines     |      0.4927 |
|...                   |             |
| Zimbabwe Airlines    |      0.4934 |
+----------------------+-------------+
113 rows in set (0.3585 sec)
```

Creating the MySQL HeatWave temporary table takes approximately 4 seconds. However, it significantly reduces query execution time—from 4.8 seconds to just 0.35 seconds. Reusing the temporary table across multiple reports balances the initial creation cost.

What's Next

Learn how to use the INSERT ... SELECT Statements.

5.4.6 INSERT ... SELECT Statements

This topic describes how to run a `SELECT` query and insert the results into a target table using the `INSERT ... SELECT` statement. The target table can be an InnoDB table and does not need to use a secondary engine.

When the query is offloaded to MySQL HeatWave, it is processed faster and improves the overall performance of the operation.

Before You Begin

- Review the requirements.
- Connect to your DB System.
- Verify if the query is offloaded to MySQL HeatWave for processing.
- The `SELECT` table is loaded in MySQL HeatWave, and the `INSERT` table is present on the DB System.
- The examples in this topic use the sample database `airportdb`. To learn how to download the sample database, see AirportDB Analytics Quickstart.

Execute INSERT ... SELECT Statement

When you execute an `INSERT ... SELECT` statement, the `SELECT` query is offloaded to MySQL HeatWave for execution, and result is copied into the target table in the DB System. This offloading reduces execution time, especially for long-running and complex queries. However, it may not be

beneficial for SELECT queries that produce large result sets, as the large volume of DML operations on the DB System can limit performance improvements.

For example, the following statement inserts data into the flight_from_US table on the DB System, by selecting rows from the flight table on MySQL HeatWave. It filters the flights that depart from airports located in the United States, based on matching airport IDs present in table airport_geo.

```
mysql>  CREATE TABLE flight_from_US LIKE flight;
Query OK, 0 rows affected (0.2368 sec)

INSERT INTO flight_from_US SELECT * FROM flight WHERE `from` IN (SELECT airport_id FROM airport_geo WHERE c
Query OK, 99431 rows affected (4.2551 sec)

Records: 99431  Duplicates: 0  Warnings: 0

EXPLAIN INSERT INTO flight_from_US SELECT * FROM flight WHERE `from` IN (SELECT airport_id FROM airport_geo
*************************** 1. row ***************************
           id: 1
  select_type: NONE
        table: NULL
   partitions: NULL
         type: NULL
possible_keys: NULL
          key: NULL
      key_len: NULL
          ref: NULL
         rows: NULL
     filtered: NULL
        Extra: Using secondary engine RAPID. Use EXPLAIN FORMAT=TREE to show the plan.
1 row in set, 1 warning (0.2304 sec)
```

Note the following:

- The SELECT portion of the INSERT ... SELECT statement is subject to the same MySQL HeatWave requirements and limitations as regular SELECT queries.
- The ON DUPLICATE KEY UPDATE clause is not supported.
- INSERT INTO *some_view* SELECT statements are not offloaded. Setting use_secondary_engine=FORCED does not cause the statement to fail with an error. The statement is executed on the DB System regardless of the use_secondary_engine setting.
- As of MySQL 9.2.0, MySQL HeatWave can offload the SELECT query in an INSERT ... SELECT statement, even if the target table has a trigger.

  In MySQL versions prior to 9.2.0, MySQL HeatWave cannot offload the query when the target table has a trigger.. If you set use_secondary_engine=FORCED, the operation fails and generates an offload error.
- See Section 5.4.1, “Requirements for Running Queries” and Section 11.2.1.9, “Other Limitations”.

What's Next

Learn how to query views.

5.4.7 Query Views

This topic describes how to run queries on views using MySQL HeatWave and verify if those queries are offloaded to the MySQL HeatWave secondary engine for accelerated processing.

Before You Begin

- Review the requirements.
- Connect to your DB System.
- Verify if the query is offloaded to MySQL HeatWave for processing.

- The examples in this topic use the sample database `airportdb`. To learn how to download the sample database, see AirportDB Analytics Quickstart.

Query a View

MySQL HeatWave supports querying views. Ensure that all the tables included in the view are loaded in MySQL HeatWave. Queries executed on views are subject to the same offload prerequisites and limitations as queries executed on tables.

In the following example, a view is created on the `airport` table, described in AirportDB Analytics Quickstart. The example assumes the airport table is loaded in MySQL HeatWave.

```
mysql> CREATE VIEW airport_US AS
SELECT airport.*
FROM airport
JOIN airport_geo ON airport.airport_id = airport_geo.airport_id
WHERE airport_geo.country = 'UNITED STATES';
```

To determine if a query run on a view can be offloaded to MySQL HeatWave for execution, use `EXPLAIN`. If offload is supported, the `Extra` column of `EXPLAIN` output shows `Using secondary engine RAPID`, as in the following example:

```
mysql> EXPLAIN SELECT COUNT(*) FROM airport_US\G
*************************** 1. row ***************************
           id: 1
  select_type: NONE
        table: NULL
   partitions: NULL
         type: NULL
possible_keys: NULL
          key: NULL
      key_len: NULL
          ref: NULL
         rows: NULL
     filtered: NULL
        Extra: Using secondary engine RAPID. Use EXPLAIN FORMAT=TREE to show the plan.
1 row in set, 1 warning (0.2424 sec)
```

What's Next

Learn how to View Query Runtimes and Estimates

5.4.8 Query Materalized Views

This topic describes how to create materialized views on MySQL HeatWave, which is supported as of MySQL 9.5.0.

A materialized view is a database object that stores the results of a query, rather than computing the result dynamically each time the view is accessed.

Before You Begin

- Review the requirements.
- Connect to your DB System.
- Verify if the query is offloaded to MySQL HeatWave for processing.
- The examples in this topic use the sample database `airportdb`. To learn how to download the sample database, see AirportDB Analytics Quickstart.

Create and Query a Materialized View

To create or alter a materialized view, you use the `CREATE VIEW` or `ALTER VIEW` statements.

The following example creates a materialized view for data in the in the `airportdb` database, which is described in the AirportDB Analytics Quickstart. The example assumes all tables are loaded in MySQL HeatWave.

```
mysql> CREATE MATERIALIZED VIEW airport_US
AS
SELECT
    airport.*
FROM
    airport
    JOIN airport_geo ON airport.airport_id = airport_geo.airport_id
WHERE
    airport_geo.country = 'UNITED STATES';
```

The materialized view retrieves information on U.S.-based airports by joining the `airport_id` value in the `airport` and `airport_geo` tables.

Materialization does not occur until you run a query using the materialized view.

To verify the materialized view was successfully created, you can query a sample of the data from the materialized view (`airport_US`), which triggers materialization. See the following example:

```
mysql> SELECT * FROM airportdb.airport_US limit 10;
+------------+------+------+----------------------------+
| airport_id | iata | icao | name                       |
+------------+------+------+----------------------------+
|        263 | AGC  | KAGC | ALLEGHENY CO               |
|       1497 | NULL | KOEB | BRANCH CO MEML             |
|       2850 | NULL | KCUT | CUSTER CO                  |
|       3737 | SKA  | KSKA | FAIRCHILD AFB              |
|       3774 | NULL | KHZR | FALSE RIVER REGIONAL       |
|       3829 | FFM  | KFFM | FERGUS FALLS MUN-MICKELSON |
|       5759 | JDN  | KJDN | JORDAN                     |
|       6662 | LFI  | KLFI | LANGLEY AFB                |
|       7125 | PRC  | KPRC | LOVE                       |
|       8778 | VGT  | KVGT | NORTH LAS VEGAS            |
+------------+------+------+----------------------------+
10 rows in set (0.0511 sec)
```

You can view information from the materialized view by querying the `performance_schema` database.

The following example queries the `rpd_table_id` table for queries with the `airport_US` materialized view. The `ID` value of the query (`165`) is then used to query the `rpd_tables` table.

```
mysql> SELECT * FROM performance_schema.rpd_table_id WHERE NAME='airportdb.airport_US'\G
*************************** 1. row ***************************
                   ID: 165
                 NAME: airportdb.airport_US
          SCHEMA_NAME: airportdb
           TABLE_NAME: airport_US2
MATERIALIZATION_QUERY: select `airportdb`.`airport`.`airport_id` AS `airport_id`,`airportdb`.`airport`.`iat
                       `airportdb`.`airport`.`name` AS `name` from (`airportdb`.`airport` join `airportdb`.
                       where (`airportdb`.`airport_geo`.`country` = 'UNITED STATES')
1 row in set (0.0407 sec)
 mysql> SELECT * FROM performance_schema.rpd_tables WHERE ID='165'\G
 *************************** 1. row ***************************
                        ID: 165
              SNAPSHOT_SCN: 257
             PERSISTED_SCN: 1
                 POOL_TYPE: VOLATILE
       DATA_PLACEMENT_TYPE: RoundRobin
                     NROWS: 2229
             LOAD_STATUS: AVAIL_RPDGSTABSTATE
           LOAD_PROGRESS: 100
              SIZE_BYTES: 46215
      TRANSFORMATION_BYTES: NULL
             QUERY_COUNT: 2
            LAST_QUERIED: 2025-10-06 17:37:20.015999
      LOAD_START_TIMESTAMP: NULL
```

```
         LOAD_END_TIMESTAMP: 2025-10-06 17:23:38.980762
           RECOVERY_SOURCE: NULL
   RECOVERY_START_TIMESTAMP: NULL
     RECOVERY_END_TIMESTAMP: NULL
                  LOAD_TYPE: USER
LOGICAL_PARTS_LOADED_AT_SCN: "NOT_PARTITIONED"
           AUTO_ZMP_COLUMNS: "NO COLUMNS"
                  ACE_MODEL: NOT AVAILABLE
1 row in set (0.0491 sec)
```

The `rpd_table_id` table displays query information for the materialized view in the `MATERIALIZATION QUERY` column.

The `rpd_tables` table shows that the materialized view is in the appropriate pool type for materialized views, `VOLATILE`.

To remove the materialized view, use the `DROP VIEW` statement.

Note the following regarding materialized views:

- If there are any changes to the base tables referenced by a materialized view, MySQL HeatWave re-materializes the view when a query is issued that references the materialized view. This ensures the view displays up-to-date data.
- If a materialized view refers directly to another materialized view, the referred materialized view is treated as a normal view. This means materialized views can only refer to base tables directly, and not other materialized views.
- You cannot directly update a materialized view, so `INSERT`, `UPDATE`, and `DELETE` statements do not work with materialized views.

Automated Substitution of Materialized Views

As of MySQL 9.6.0, MySQL HeatWave automatically identifies if a subtree of a query can be substituted using an existing materialized view that you created earlier.

Consider the following example:

1. Create a materialized view.

   ```
   mysql> CREATE MATERIALIZED VIEW LINEITEM_PART1
   AS SELECT *
   FROM LINEITEM, PART
   WHERE L_PARTKEY = P_PARTKEY AND P_BRAND = "Brand#23";
   ```

2. Run a query referencing the materialized view.

   ```
   mysql> SELECT COUNT(*) FROM LINEITEM_PART1;
   ```

3. Run a query that contains a subtree that can be from the `LINEITEM_PART1` materialized view.

   ```
   mysql> SELECT * FROM LINEITEM, ORDERS, PART
   WHERE L_ORDERKEY = O_ORDERKEY
   AND L_PARTKEY = P_PARTKEY
   AND P_BRAND = ”Brand#23”
   AND O_ORDERSTATUS = ”A”;
   ```

   MySQL HeatWave analyzes the query and makes appropriate substitutions to the query automatically. The following query shows the automated substitution. You are not required to run this query.

   ```
   SELECT * FROM ORDERS, LINEITEM_PART1
   WHERE O_ORDERKEY = L_ORDERKEY
   AND O_ORDERSTATUS = ‘A’;
   ```

4. Query the `rpd_query_stats` table to confirm if automated substitution occurred.

```
mysql> SELECT
JSON_EXTRACT(QKRN_TEXT,'$.qkrnNodesArray[*].nodeSubType', '$.qkrnNodesArray[*].tableId')
FROM performance_schema.rpd_query_stats
ORDER BY MYSQL_QUERY_ID DESC
LIMIT 1;
```

Confirm that `nodeSubType` appears as `Materialized Table`.

Only up-to-date materialized views are considered for automated substitution. A materialized view is up-to-date (fresh) when all base tables that it references have not undergone any changes since the materialized view's last materialization. If any base table undergoes a change, the materialized view is deemed out-of-date (stale). It is re-materialized when a query references it directly in the future.

You can refresh materialized views by running any query that references them. For example:

```
mysql> SELECT COUNT(*) FROM LINEITEM_PART1;
```

rapid_auto_create_materialized_view Variable

MySQL 9.6.0 introduces the `rapid_auto_create_materialized_view` variable. When enabled, the MySQL HeatWave optimizer tries to automatically create materialized views, both from final or intermediate results, from queries that are run in the system. These automatically created materialized views act like a result cache for future queries.

To enable this feature, set the variable by running the following command:

```
mysql> SET SESSION rapid_auto_create_materialized_view=1;
```

You can view automatically created materialized views in the `performance_schema.rpd_tables` and `performance_schema.rpd_table_id` tables.

To confirm the usage of materialized views after running a query, you can run the following command:

```
mysql> SELECT
JSON_EXTRACT(QKRN_TEXT,'$.qkrnNodesArray[*].nodeSubType', '$.qkrnNodesArray[*].tableId')
FROM performance_schema.rpd_query_stats
ORDER BY MYSQL_QUERY_ID DESC
LIMIT 1;
```

To disable `rapid_auto_create_materialized_view`, and delete all automatically created materialized views, run the following command:

```
mysql> SET SESSION rapid_auto_create_materialized_view=0;
```

What's Next

Learn how to View Query Runtimes and Estimates.

5.4.9 View Query Runtimes and Estimates

This topic describes how you can view MySQL HeatWave query runtimes and runtime estimates using the MySQL HeatWave Autopilot Advisor Auto Query Time Estimation feature, or by querying the `performance_schema.rpd_query_stats` table.

Runtime data is useful for query optimization, troubleshooting, and estimating the cost of running a particular query or workload.

MySQL HeatWave query runtime data includes:

- Runtimes for successfully executed queries
- Runtime estimates for `EXPLAIN` queries
- Runtime estimates for queries cancelled using `Ctrl+C`
- Runtime estimates for queries that fail due to an out-of-memory error

Runtime data is available for queries in the MySQL HeatWave query history, which is a non-persistent store of information about the last 1000 executed queries.

Before You Begin

- Review the requirements.
- Connect to your DB System.
- Verify if the query is offloaded to MySQL HeatWave for processing.
- The examples in this topic use the sample database `airportdb`. To learn how to download the sample database, see AirportDB Analytics Quickstart.

View Runtime Data Using Autopilot Advisor

Perform the following steps to view runtime data using Autopilot Advisor Auto Query Time Estimation:

- To view runtime data for all queries in the MySQL HeatWave history:

```
mysql> CALL sys.heatwave_advisor(JSON_OBJECT("query_insights", true));
+------------------------------+
| INITIALIZING HEATWAVE ADVISOR |
+------------------------------+
| Version: 4.15                |
|                              |
| Output Mode: normal          |
| Excluded Queries: 0          |
| Target Schemas: All          |
|                              |
+------------------------------+
6 rows in set (0.2571 sec)

+-----------------------------------------------------------+
| ANALYZING LOADED DATA                                     |
+-----------------------------------------------------------+
| Total 14 tables loaded in HeatWave for 1 schemas          |
| Tables excluded by user: 0 (within target schemas)        |
|                                                           |
| SCHEMA                            TABLES        COLUMNS |
| NAME                              LOADED         LOADED |
| ------                            ------         ------ |
| `airportdb`                           14            106 |
|                                                           |
+-----------------------------------------------------------+
8 rows in set (0.2571 sec)

+-------------------------------------------------------------------------------------------------------------
| QUERY INSIGHTS
+-------------------------------------------------------------------------------------------------------------
| Queries executed on Heatwave: 8
| Session IDs (as filter): None
|
| QUERY-ID  SESSION-ID  QUERY-STRING                                                    EXEC-RUNTIME
| --------  ----------  ------------                                                    ------------
|       11       20083  select count(*) from flight                                         0.010
|       12       20083  select count(*) from flight                                         0.007
|       13       20083  select count(*) from flight                                         0.005
|       14       53204  SELECT airline_id, COUNT(*) AS flight_id           FROM flig...      0.000 (e
|       15       53204  SELECT airline_id, COUNT(*) AS flight_id           FROM flig...      0.022
|       16       53204  SELECT airline_id, COUNT(*) AS flight_id           FROM flig...      0.000 (e
|       17       53204  SELECT airline_id, COUNT(*) AS flight_id           FROM flig...      0.007
|       18       53204  SELECT airline_id, COUNT(*) AS flight_id           FROM flig...      0.006
|
| TOTAL ESTIMATED:   2   EXEC-RUNTIME:       0.000 sec
| TOTAL EXECUTED:    6   EXEC-RUNTIME:       0.057 sec
|
|
| Retrieve detailed query statistics using the query below:
|     SELECT log FROM sys.heatwave_autopilot_report WHERE stage = "QUERY_INSIGHTS" AND type = "info";
```

```
|
+-------------------------------------------------------------------------------------------------
21 rows in set (0.2571 sec)

Query OK, 0 rows affected (0.2571 sec)
```

- To view runtime data for queries executed by the current session only:

```
mysql> CALL sys.heatwave_advisor(JSON_OBJECT("query_insights", true,
          "query_session_id", JSON_ARRAY(connection_id())));
+-------------------------------+
| INITIALIZING HEATWAVE ADVISOR |
+-------------------------------+
| Version: 4.15                 |
|                               |
| Output Mode: normal           |
| Excluded Queries: 0           |
| Target Schemas: All           |
|                               |
+-------------------------------+
6 rows in set (0.2464 sec)

+----------------------------------------------------------+
| ANALYZING LOADED DATA                                    |
+----------------------------------------------------------+
| Total 14 tables loaded in HeatWave for 1 schemas         |
| Tables excluded by user: 0 (within target schemas)       |
|                                                          |
| SCHEMA                            TABLES         COLUMNS |
| NAME                              LOADED          LOADED |
| ------                            ------          ------ |
| `airportdb`                           14             106 |
|                                                          |
+----------------------------------------------------------+
8 rows in set (0.2464 sec)

+-------------------------------------------------------------------------------------------------
| QUERY INSIGHTS
+-------------------------------------------------------------------------------------------------
| Queries executed on Heatwave: 0
| Session IDs (as filter): [56742]
|
| QUERY-ID  SESSION-ID  QUERY-STRING                                                   EXEC-RUNTIME (s)
| --------  ----------  ------------                                                   ----------------
|
| TOTAL ESTIMATED:   0   EXEC-RUNTIME:       0.000 sec
| TOTAL EXECUTED:    0   EXEC-RUNTIME:       0.000 sec
|
|
| Retrieve detailed query statistics using the query below:
|     SELECT log FROM sys.heatwave_autopilot_report WHERE stage = "QUERY_INSIGHTS" AND type = "info";
|
+-------------------------------------------------------------------------------------------------
13 rows in set (0.2464 sec)
```

See: Section 5.8.6, “Auto Query Time Estimation”.

View Runtime Data Using rpd_query_stats Table

Perform the following steps to view runtime data using the `performance_schema.rpd_query_stats` table:

- To view runtime data for all queries in the MySQL HeatWave query history:

```
mysql> SELECT query_id,
          JSON_EXTRACT(JSON_UNQUOTE(qkrn_text->'$**.sessionId'),'$[0]') AS session_id,
          JSON_EXTRACT(JSON_UNQUOTE(qkrn_text->'$**.accumulatedRapidCost'),'$[0]') AS time_in_ns,
          JSON_EXTRACT(JSON_UNQUOTE(qexec_text->'$**.error'),'$[0]') AS error_message
          FROM performance_schema.rpd_query_stats;
+----------+------------+------------+--------------+
```

```
| query_id | session_id | time_in_ns | error_message |
+----------+------------+------------+---------------+
|       11 | 20083      | NULL       | ""            |
|       12 | 20083      | NULL       | ""            |
|       13 | 20083      | NULL       | ""            |
|       14 | 53204      | NULL       | NULL          |
|       15 | 53204      | NULL       | ""            |
|       16 | 53204      | NULL       | NULL          |
|       17 | 53204      | NULL       | ""            |
|       18 | 53204      | NULL       | ""            |
+----------+------------+------------+---------------+
8 rows in set (0.2328 sec)
```

- To view runtime data for a particular MySQL HeatWave query, filtered by query ID:

```
mysql> SELECT query_id,
          JSON_EXTRACT(JSON_UNQUOTE(qkrn_text->'$**.sessionId'),'$[0]') AS session_id,
          JSON_EXTRACT(JSON_UNQUOTE(qkrn_text->'$**.accumulatedRapidCost'),'$[0]') AS time_in_ns,
          JSON_EXTRACT(JSON_UNQUOTE(qexec_text->'$**.error'),'$[0]') AS error_message
          FROM performance_schema.rpd_query_stats
          WHERE query_id = 1;
```

- To view the query ID by querying `performance_schema.rpd_query_stats` table:

```
mysql> SELECT query_id, LEFT(query_text,160)
          FROM performance_schema.rpd_query_stats;
+----------+-----------------------------------------------------+
| query_id | LEFT(query_text,160)                                |
+----------+-----------------------------------------------------+
|       11 | select count(*) from flight                         |
|       12 | select count(*) from flight                         |
|       13 | select count(*) from flight                         |
|       14 | EXPLAIN SELECT airline_id, COUNT(*) AS flight_id |
|             FROM flight                                        |
|             WHERE departure <= DATE '2015-06-09'         |
|             GROUP BY airline_id                                |
|             ORDER BY airline_id                                |
|       15 | SELECT airline_id, COUNT(*) AS flight_id         |
|             FROM flight                                        |
|             WHERE departure <= DATE '2015-06-09'         |
|             GROUP BY airline_id                                |
|             ORDER BY airline_id                                |
|       16 | EXPLAIN SELECT airline_id, COUNT(*) AS flight_id |
|             FROM flight                                        |
|             WHERE flight_id <= 66810                     |
|             GROUP BY airline_id                                |
|             ORDER BY airline_id                                |
|       17 | SELECT airline_id, COUNT(*) AS flight_id         |
|             FROM flight                                        |
|             WHERE flight_id <= 66810                     |
|             GROUP BY airline_id                                |
|             ORDER BY airline_id                                |
|       18 | SELECT airline_id, COUNT(*) AS flight_id         |
|             FROM flight                                        |
|             WHERE flight_id <= 66810                     |
|             GROUP BY airline_id                                |
|             ORDER BY airline_id                                |
+----------+-----------------------------------------------------+
```

What's Next

Learn how to diagnose query offload errors.

5.4.10 Diagnose Query Offload

This topic describes how to debug queries that fail to offload to MySQL HeatWave for execution.

Before You Begin

- Review the requirements.

- Connect to your DB System.
- The examples in this topic use the sample database `airportdb`. To learn how to download the sample database, see AirportDB Analytics Quickstart.

Setting Secondary Engine Definition

By default, `SELECT` queries are offloaded to MySQL HeatWave for execution and fall back to the DB System if that is not possible. To force a query to execute on MySQL HeatWave or fail if that is not possible, set the `use_secondary_engine variable` to `FORCED`. In this mode, a `SELECT` statement returns an error if it cannot be offloaded. The `use_secondary_engine` variable can be set as shown:

Using a `SET` statement before running queries:

```
mysql> SET SESSION use_secondary_engine = FORCED;
```

Using a `SET_VAR` optimizer hint when issuing a query:

```
mysql> SELECT /*+ SET_VAR(use_secondary_engine = FORCED) */ ... FROM ...
```

Enable MySQL Optimizer Trace

If a query fails to offload and you cannot identify the reason, enable tracing and query the `INFORMATION_SCHEMA.OPTIMIZER_TRACE` table to debug the query.

As of MySQL version 9.0.0, the optimizer trace also includes reasons for not offloading a query due to dynamic offload cost calculation.

1. To enable MySQL optimizer trace, set the `optimizer_trace` and `optimizer_trace_offset` variables as shown:

   ```
   mysql> SET SESSION optimizer_trace="enabled=on";
   mysql> SET optimizer_trace_offset=-2;
   ```

   Note

   If you encounter the following error:, when executing long queries

   ```
   ERROR 3141 (22032): Invalid JSON text in argument 1 to function json_extract: "Missing
   ```

   To avoid this error, increase the memory allocated for the optimizer trace, before enabling the trace by executing:

   ```
   SET optimizer_trace_max_mem_size = 1024000000;
   ```

2. Issue the problematic query using `EXPLAIN`. If the query is supported by MySQL HeatWave, the `Extra` column in the `EXPLAIN` output shows the following text: `Using secondary engine RAPID`; otherwise, that text does not appear.

   The following query example uses `INTERVAL()` function to calculate and display the time interval into which the departure hour falls, based on the specified thresholds (6, 12, 18, 24). by MySQL HeatWave:

   This query is not offloaded to MySQL HeatWave.

   ```
   mysql> EXPLAIN SELECT flightno, INTERVAL(HOUR(departure),6,12,18,24)
   FROM flightschedule\G
   *************************** 1. row ***************************
              id: 1
     select_type: SIMPLE
           table: flightschedule
      partitions: NULL
            type: ALL
   possible_keys: NULL
   ```

```
          key: NULL
      key_len: NULL
          ref: NULL
         rows: 9828
     filtered: 100
        Extra: NULL
1 row in set, 1 warning (0.2389 sec)
```

Execute the following `SELECT` statements to detect the error:

```
mysql>  SELECT QUERY, TRACE->'$**.Rapid_Offload_Fails'
FROM INFORMATION_SCHEMA.OPTIMIZER_TRACE\G
*************************** 1. row ***************************
QUERY: EXPLAIN SELECT flightno, INTERVAL(HOUR(departure),6,12,18,24) FROM flightschedule
TRACE->'$**.Rapid_Offload_Fails': [{"Reason": "HW_ER_1014: The function, [INTERVAL(HOUR(departure),6
1 row in set (0.2385 sec)
```

```
mysql> SELECT QUERY, TRACE->'$**.secondary_engine_not_used'
FROM INFORMATION_SCHEMA.OPTIMIZER_TRACE\G
*************************** 1. row ***************************
QUERY: EXPLAIN SELECT flightno, INTERVAL(HOUR(departure),6,12,18,24) FROM flightschedule
TRACE->'$**.secondary_engine_not_used': NULL
1 row in set (0.2385 sec)
```

In order to offload this query, replace the `INTERVAL()` function with multiple IF().

```
mysql> EXPLAIN SELECT flightno, IF(HOUR(departure)<6,0,IF(HOUR(departure)<12,1,IF(HOUR(departure)<18
           *************************** 1. row ***************************
           id: 1
  select_type: NONE
        table: NULL
   partitions: NULL
         type: NULL
possible_keys: NULL
          key: NULL
      key_len: NULL
          ref: NULL
         rows: NULL
     filtered: NULL
        Extra: Using secondary engine RAPID. Use EXPLAIN FORMAT=TREE to show the plan.
```

3. Query the `INFORMATION_SCHEMA.OPTIMIZER_TRACE` table for a failure reason. There are two trace markers for queries that fail to offload:

   - `Rapid_Offload_Fails`
   - `secondary_engine_not_used`

   To query for the `Rapid_Offload_Fails` trace marker, issue this query:

   ```
   mysql> SELECT QUERY, TRACE->'$**.Rapid_Offload_Fails'
            FROM INFORMATION_SCHEMA.OPTIMIZER_TRACE;
   ```

   To query for the `secondary_engine_not_used` trace marker, issue this query:

   ```
   mysql> SELECT QUERY, TRACE->'$**.secondary_engine_not_used'
            FROM INFORMATION_SCHEMA.OPTIMIZER_TRACE;
   ```

   Note

   If the optimizer trace does not return all of the trace information, increase the optimizer trace buffer size.

4. If an `INFORMATION_SCHEMA.OPTIMIZER_TRACE` query trace indicates that a subquery is not yet supported, try unnesting the subquery. For example, the following query contains a subquery and

is not offloaded as indicated by the `EXPLAIN` output, which does not show `Using secondary engine`.

```
mysql> EXPLAIN SELECT COUNT(*)
          FROM orders o
          WHERE o_totalprice> (SELECT AVG(o_totalprice)
          FROM orders
          WHERE o_custkey=o.o_custkey);
*************************** 1. row ***************************
           id: 1
  select_type: PRIMARY
        table: o
   partitions: NULL
         type: ALL
possible_keys: NULL
          key: NULL
      key_len: NULL
          ref: NULL
         rows: 14862970
     filtered: 100.00
        Extra: Using where
*************************** 2. row ***************************
           id: 2
  select_type: DEPENDENT SUBQUERY
        table: orders
   partitions: NULL
         type: ALL
possible_keys: NULL
          key: NULL
      key_len: NULL
          ref: NULL
         rows: 14862970
     filtered: 10.00
        Extra: Using where
2 rows in set, 2 warnings (0.00 sec)
```

Re-write as follows to unnest the subquery so that it can be offloaded.

```
mysql> EXPLAIN SELECT COUNT(*)
          FROM orders o, (SELECT o_custkey, AVG(o_totalprice) a_totalprice
          FROM orders
          GROUP BY o_custkey)a
          WHERE o.o_custkey=a.o_custkey AND o.o_totalprice>a.a_totalprice;
```

The reason reported for a query offload failure depends on the issue or limitation encountered. For common issues, such as unsupported clauses or functions, a specific reason is reported. For undefined issues or unsupported query transformations performed by the optimizer, the following generic reason is reported:

```
mysql> [{"Reason": "Currently unsupported RAPID query compilation scenario"}]
```

For a query that does not meet the query cost threshold for MySQL HeatWave, the following reason is reported, and the query is not offloaded:

```
mysql> [{"Reason": "The estimated query cost does not exceed secondary_engine_cost_threshold."}]
```

If query does not offload due to Dynamic Offload (newer alternative to cost threshold), the following message is displayed.

```
mysql> [{message : "HeatWave dynamic threshold rejects query"}]
```

The query cost threshold prevents queries of little cost from being offloaded to MySQL HeatWave. For information about the query cost threshold, see Section 11.1.1, “MySQL HeatWave Issues”.

To execute such queries using the MySQL HeatWave secondary engine, set the `use_secondary_engine system variable` to `FORCED` state using the following syntax options. By default, this variable is set to `ON`.

```
mysql> SET SESSION use_secondary_engine = 'FORCED';
```

```
Query OK, 0 rows affected (0.3451 sec)
```

For a query that attempts to access a column defined as `NOT SECONDARY`, the following reason is reported:

```
mysql> [{"Reason": "Column risk_assessment is marked as NOT SECONDARY."}]
```

Columns defined as `NOT SECONDARY` are excluded when a table is loaded into MySQL HeatWave. See Section 4.2.6.1, “Exclude Table Columns”.

Diagnosing and Resolving Out-of-Memory (OOM) Errors in MySQL Query Execution

If you encounter out-of-memory errors when running queries:

1. Avoid or rewrite queries that produce a Cartesian product. In the following query, a `JOIN` predicated is not defined between the `supplier` and `nation` tables, which causes the query to select all rows from both tables:

   ```
   mysql> SELECT s_nationkey, s_suppkey, l_comment FROM lineitem, supplier, nation
              WHERE s_suppkey = l_suppkey LIMIT 10;
   ERROR 3015 (HY000): Out of memory in storage engine 'Failure detected in RAPID; query
   execution cannot proceed'.
   ```

   To avoid the Cartesian product, add a relevant predicate between the `supplier` and `nation` tables to filter out rows:

   ```
   mysql> SELECT s_nationkey, s_suppkey, l_comment
              FROM lineitem, supplier, nation
              WHERE s_nationkey = n_nationkey AND s_suppkey = l_suppkey LIMIT 10;
   ```

2. Avoid or rewrite queries that produce a Cartesian product introduced by the MySQL optimizer. Due to lack of quality statistics or non-optimal cost decisions, MySQL optimizer may introduce one or more Cartesian products in a query even if a query has predicates defined among all participating tables. For example:

   ```
   mysql> SELECT o_orderkey, c_custkey, l_shipdate, s_nationkey, s_suppkey, l_comment
              FROM lineitem, supplier, nation, customer, orders
              WHERE c_custkey = o_custkey AND o_orderkey = l_orderkey
              AND c_nationkey = s_nationkey AND c_nationkey = n_nationkey AND c_custkey < 3000000
              LIMIT 10;
   ERROR 3015 (HY000): Out of memory in storage engine 'Failure detected in RAPID;
   query execution cannot proceed'.
   ```

   The `EXPLAIN` plan output shows that there is no common predicate between the first two table entries (`NATION` and `SUPPLIER`).

   ```
   mysql> EXPLAIN SELECT o_orderkey, c_custkey, l_shipdate, s_nationkey, s_suppkey, l_comment
              FROM lineitem, supplier, nation, customer, orders
              WHERE c_custkey = o_custkey AND o_orderkey = l_orderkey AND c_nationkey = s_nationkey
              AND c_nationkey = n_nationkey AND c_custkey < 3000000 LIMIT 10;
   *************************** 1. row ***************************
              id: 1
     select_type: SIMPLE
           table: supplier
      partitions: NULL
            type: ALL
   possible_keys: NULL
             key: NULL
         key_len: NULL
             ref: NULL
            rows: 99626
        filtered: 100.00
           Extra: Using secondary engine RAPID
   *************************** 2. row ***************************
              id: 1
     select_type: SIMPLE
           table: nation
      partitions: NULL
            type: ALL
   ```

```
possible_keys: NULL
          key: NULL
      key_len: NULL
          ref: NULL
         rows: 25
     filtered: 10.00
        Extra: Using where; Using join buffer (hash join); Using secondary engine RAPID
*************************** 3. row ***************************
           id: 1
  select_type: SIMPLE
        table: customer
   partitions: NULL
         type: ALL
possible_keys: NULL
          key: NULL
      key_len: NULL
          ref: NULL
         rows: 1382274
     filtered: 5.00
        Extra: Using where; Using join buffer (hash join); Using secondary engine RAPID
*************************** 4. row ***************************
           id: 1
  select_type: SIMPLE
        table: orders
   partitions: NULL
         type: ALL
possible_keys: NULL
          key: NULL
      key_len: NULL
          ref: NULL
         rows: 14862970
     filtered: 10.00
        Extra: Using where; Using join buffer (hash join); Using secondary engine RAPID
*************************** 5. row ***************************
           id: 1
  select_type: SIMPLE
        table: lineitem
   partitions: NULL
         type: ALL
possible_keys: NULL
          key: NULL
      key_len: NULL
          ref: NULL
         rows: 56834662
     filtered: 10.00
        Extra: Using where; Using join buffer (hash join); Using secondary engine RAPID
```

To force a join order so that there are predicates associated with each pair of tables, add a `STRAIGHT_JOIN` hint. For example:

```
mysql> EXPLAIN SELECT o_orderkey, c_custkey, l_shipdate, s_nationkey, s_suppkey, l_comment
          FROM SUPPLIER STRAIGHT_JOIN CUSTOMER STRAIGHT_JOIN NATION STRAIGHT_JOIN ORDERS
          STRAIGHT_JOIN LINEITEM WHERE c_custkey = o_custkey and o_orderkey = l_orderkey
          AND c_nationkey = s_nationkey AND c_nationkey = n_nationkey AND c_custkey < 3000000
          LIMIT 10;
*************************** 1. row ***************************
           id: 1
  select_type: SIMPLE
        table: supplier
   partitions: NULL
         type: ALL
possible_keys: NULL
          key: NULL
      key_len: NULL
          ref: NULL
         rows: 99626
     filtered: 100.00
        Extra: Using secondary engine RAPID
*************************** 2. row ***************************
           id: 1
  select_type: SIMPLE
        table: customer
```

```
   partitions: NULL
         type: ALL
possible_keys: NULL
          key: NULL
      key_len: NULL
          ref: NULL
         rows: 1382274
     filtered: 5.00
        Extra: Using where; Using join buffer (hash join); Using secondary engine RAPID
*************************** 3. row ***************************
           id: 1
  select_type: SIMPLE
        table: nation
   partitions: NULL
         type: ALL
possible_keys: NULL
          key: NULL
      key_len: NULL
          ref: NULL
         rows: 25
     filtered: 10.00
        Extra: Using where; Using join buffer (hash join); Using secondary engine RAPID
*************************** 4. row ***************************
           id: 1
  select_type: SIMPLE
        table: orders
   partitions: NULL
         type: ALL
possible_keys: NULL
          key: NULL
      key_len: NULL
          ref: NULL
         rows: 14862970
     filtered: 10.00
        Extra: Using where; Using join buffer (hash join); Using secondary engine RAPID
*************************** 5. row ***************************
           id: 1
  select_type: SIMPLE
        table: lineitem
   partitions: NULL
         type: ALL
possible_keys: NULL
          key: NULL
      key_len: NULL
          ref: NULL
         rows: 56834662
     filtered: 10.00
        Extra: Using where; Using join buffer (hash join); Using secondary engine RAPID
```

3. Avoid or rewrite queries that produce a very large result set. This is a common cause of out of memory errors during query processing. Use aggregation functions, a `GROUP BY` clause, or a `LIMIT` clause to reduce the result set size.

4. Avoid or rewrite queries that produce a very large intermediate result set. In certain cases, large result sets can be avoided by adding a `STRAIGHT_JOIN` hint, which enforces a join order in a decreasing of selectiveness.

5. Check the size of your data by performing a cluster size estimate. If your data has grown substantially, the MySQL HeatWave Cluster may require additional nodes.

   - For MySQL HeatWave on OCI, see Generating a Node Count Estimate in the *MySQL HeatWave on OCI Service Guide*.
   - For MySQL HeatWave on AWS, see Estimating Cluster Size with MySQL HeatWave Autopilot in the *MySQL HeatWave on AWS Service Guide*.
   - For MySQL HeatWave for Azure, see Provisioning MySQL HeatWave Nodes in the *MySQL HeatWave for Azure Service Guide*.

6. MySQL HeatWave optimizes for network usage by default. Try running the query with the `MIN_MEM_CONSUMPTION` strategy by setting by setting `rapid_execution_strategy` to `MIN_MEM_CONSUMPTION`. The `rapid_execution_strategy` variable can be set as shown:

   - Using a `SET` statement before running queries:

     ```
     mysql> SET SESSION rapid_execution_strategy = MIN_MEM_CONSUMPTION;
     ```

   - Using a `SET_VAR` optimizer hint when issuing a query:

     ```
     mysql> SELECT /*+ SET_VAR(rapid_execution_strategy = MIN_MEM_CONSUMPTION) */ ... FROM ...
     ```

   - Unloading tables that are not used. These tables consume memory on MySQL HeatWave nodes unnecessarily. See Section 4.4.2, “Unload Data Manually”.

   - Excluding table columns that are not accessed by your queries. These columns consume memory on MySQL HeatWave nodes unnecessarily. This strategy requires reloading data. See Section 4.2.6.1, “Exclude Table Columns”.

7. After running queries, consider using MySQL HeatWave Autopilot Advisor for encoding and data placement recommendations. See Section 5.8, “Optimize Workloads for OLAP”.

What's Next

Learn about Auto Scheduling.

5.4.11 Auto Scheduling

MySQL HeatWave uses a workload-aware, priority-based, automated scheduling mechanism to schedule concurrently issued queries for execution. The scheduling mechanism prioritizes short-running queries but considers wait time in the queue so that costlier queries are eventually scheduled for execution. This scheduling approach reduces query execution wait times overall.

Concurrency referred in this section is specifically for the concurrent query requests to one single MySQL server

When a MySQL HeatWave Cluster is idle, an arriving query is immediately scheduled for execution. It is not queued. A query is queued only if the MySQL HeatWave Cluster is running a preceding query.

A light-weight cost estimate is performed for each query at query compilation time.

Queries cancelled via `Ctrl-C` are removed from the scheduling queue.

To view the MySQL HeatWave query history including query start time, end time, and wait time in the scheduling queue, see Section 5.10, “MySQL HeatWave Monitoring”.

MySQL 9.0.0 introduces a novel scheduler that a MySQL HeatWave Cluster uses to run multiple concurrent queries before placing further queries on the queue. The scheduler can interleave queries inside each MySQL HeatWave node within a MySQL HeatWave Cluster to achieve the following scheduling improvements:

- Maximise query processing throughput.
- Make the query wait time proportional to the query execution time.
- If a query does raise an out of memory error, the scheduler runs the query a second time in isolation.

The scheduler can run up to 12 concurrent queries within a MySQL HeatWave Cluster. The actual number of concurrent queries depends upon the predicted memory requirements, and available memory.

The scheduler runs MySQL HeatWave AutoML, MySQL HeatWave GenAI, MySQL HeatWave Lakehouse and data load queries one at a time.

- The scheduler can run multiple analytical queries if there is no more than one data load query running.
- The scheduler can run multiple data load queries if there is no more than one analytical query running.

What's Next

- Learn more about Auto Query Plan Improvement.
- Learn more about Dynamic Query Offload.

5.4.12 Auto Query Plan Improvement

The *Auto Query Plan Improvement* feature collects and stores query plan statistics in a statistics cache when a query is executed in MySQL HeatWave. When a new query shares query execution plan nodes with previously executed queries, the statistics collected from previously executed queries are used instead of estimated statistics, which improves query execution plans, cost estimations, execution times, and memory efficiency.

Each entry in the cache corresponds to a query execution plan node. A query execution plan may have nodes for table scans, `JOIN`, `GROUP BY` and other operations.

The statistics cache is an LRU structure. When cache capacity is reached, the least recently used entries are evicted from the cache as new entries are added. The number of entries permitted in the statistics cache is 65536, which is enough to store statistics for 4000 to 5000 unique queries of medium complexity. The maximum number of statistics cache entries is defined by the MySQL-managed `rapid_stats_cache_max_entries` setting.

What's Next

Learn more about Dynamic Query Offload.

5.4.13 Dynamic Query Offload

Before MySQL 9.0.0, MySQL HeatWave uses a query cost threshold to choose either INNODB or MySQL HeatWave as the execution engine for the query.

As of MySQL 9.0.0, MySQL HeatWave uses dynamic query offload to choose the optimal execution engine for queries based on query and execution engine characteristics, which achieves improved performance.

Dynamic query offload analyses the query characteristics and execution engine static and dynamic characteristics to choose the best engine for the query, given the current system state. This includes the following:

- Secondary engine change propagation.
- Secondary engine query queue.
- The presence of indexes on the primary engine.

Dynamic query offload introduces a minimal overhead for fast queries. The overhead for more complex queries is also minimal, compared to the time required for query optimisation and subsequent execution.

What's Next

Learn more about modifying tables.

5.5 Modify Tables

When an InnoDB table is loaded into MySQL HeatWave Cluster, DDL operations such as `ALTER TABLE`, `RENAME TABLE`, `TRUNCATE TABLE`, and `DROP TABLE` are permitted. While any of these operations are being executed, queries involving the modified table are not offloaded to the MySQL HeatWave engine; they are processed by InnoDB. If you are explicitly trying to use the secondary engine, these queries generate an offload error.

It is recommended that you avoid expensive queries on a table immediately after performing a DDL operation.

Reload the table into the secondary engine. Queries accessing the table cannot be offloaded until the reload is complete.

To check the status and the progress of the tables present in MySQL HeatWave, run the following query:

```
mysql> SELECT SCHEMA_NAME, TABLE_NAME, LOAD_STATUS, LOAD_PROGRESS, LOAD_TYPE
  FROM performance_schema.rpd_tables NATURAL JOIN performance_schema.rpd_table_id;
+-------------+-------------------+---------------------+---------------+-----------+
| SCHEMA_NAME | TABLE_NAME        | LOAD_STATUS         | LOAD_PROGRESS | LOAD_TYPE |
+-------------+-------------------+---------------------+---------------+-----------+
| airportdb   | flight_temp       | AVAIL_RPDGSTABSTATE |           100 | USER      |
| airportdb   | passengerdetails  | AVAIL_RPDGSTABSTATE |           100 | USER      |
| airportdb   | airport           | AVAIL_RPDGSTABSTATE |           100 | USER      |
+-------------+-------------------+---------------------+---------------+-----------+
```

The following DDL statements do not initiate a data reload in MySQL HeatWave:

- ALTER TABLE *`InnoDbTableName`* SECONDARY_UNLOAD;
- ALTER TABLE *`InnoDbTableName`* SECONDARY_ENGINE = NULL;
- DROP TABLE *`InnoDbTableName`*;

What's Next

Learn how to analyze tables.

5.6 Analyze Tables

As of MySQL 9.3.0, you can use the `ANALYZE TABLE` statement to analyze tables loaded into MySQL HeatWave and update table statistics.

Before You Begin

- Ensure that the `enable_secondary_engine_statistics` system variables is set to `ON`.
- Check if you have the `SELECT` and `INSERT` privileges for the table that you want to analyze. If not, ask your admin user to grant you the required privileges.

Updating Table Statistics

Use the `ANALYZE TABLE` statement to update table statistics.

```
mysql> ANALYZE TABLE tbl_name;
```

For example:

```
mysql> ANALYZE TABLE airline;
```

The statement returns a table similar to the following:

```
+---------+--------------------------+----------+----------+
| Table   | Op                       | Msg_type | Msg_text |
```

```
+---------+-------------------------+----------+----------+
| airline | analyze                 | status   | OK       |
| airline | analyze secondary engine | status   | OK       |
+---------+-------------------------+----------+----------+
```

The details of columns available in this table are described in ANALYZE TABLE Output.

When you run the `ANALYZE TABLE` statement for a table loaded in MySQL HeatWave, the analysis is performed on MySQL HeatWave. The output contains an additional row showing the results of the analysis on the secondary engine.

When using `ANALYZE TABLE`, keep the following in mind:

- The `ANALYZE TABLE` statement builds statistics based on the table loaded in MySQL HeatWave. Therefore, it takes into account change propagation updates that are applied to the table when `ANALYZE TABLE` runs.
- When a query runs for a table, the statistics collected by `ANALYZE TABLE` is different from the statistics collected by MySQL HeatWave. The optimizer uses both types of statistics when possible, and neither fully replaces the other.
- You can consider `ANALYZE TABLE` statistics as a more powerful version of column histograms. Therefore, they can help query optimization when a table is being filtered using ranges, equalities, inequalities, and `IN()` or `NOT IN()` with constants (for example, `column1 > 20 AND column1 < 50 OR column2 = 'A' OR column3 NOT IN [1,2,3]`).

ACE Statistics

The `ANALYZE TABLE` statement updates the Advanced Cardinality Estimation (ACE) statistics for tables loaded in MySQL HeatWave. ACE statistics models are a more powerful version of column histograms and are used by secondary engine to perform query optimization.

If the `enable_secondary_engine_statistics` system variable is set to `ON`, the secondary engine builds an ACE statistics model for a table when the `ANALYZE TABLE` statement is run on the table with or without the `UPDATE HISTOGRAM` clause.

The secondary engine does not build an ACE statistics model for a table in the following scenarios:

- The table does not support ACE statistics, or if it is an empty table.
- An ACE statistics model already exists for a table and no updates were made to the table since the last time the ACE statistics model was built.

When the secondary engine creates the ACE statistics model, the `ACE_MODEL` column is updated in the performance schema table `rpd_tables`.

5.7 Run Tasks Asynchronously

As of MySQL 9.3.1, you can run SQL statements as asynchronous tasks that run in the background. This lets your current session to continue without waiting for these SQL statements or tasks to complete. It is particularly helpful for long-running tasks such as data processing, report generation, or any workload that would normally block the session if run synchronously. You can run such tasks independent of the session where you initiate them and your session or application stays responsive while the task continues to run in the background.

For applications that have multiple application users, and there is no direct mapping between MySQL users and application users, you can use the application-level variants of the asynchronous task routines. Application-level asynchronous task routines let you filter and track tasks for individual application users using the application-managed user ID.

This topic includes the following sections:

- Before You Begin

- Asynchronous Task Components
- Update Maximum Task Limits
- Run SQL Statements Asynchronously
- What's Next

To learn about the limitations of asynchronous tasks, see Section 11.2.1.10, “Asynchronous Task Limitations”.

Before You Begin

- Ensure that you have the following roles and the roles are active:
  - To initiate and manage your own asynchronous tasks, you require the `mysql_task_user` role.
  - To manage all the other users' tasks, backup and restore data, or change the maximum task limits, you require the `mysql_task_admin` role.

  Learn how to grant and activate a role.
- Ensure that you are connected to the DB System.

Asynchronous Task Components

The `mysql_tasks` schema includes the following components:

- Task metadata: stores information about each task, including the task ID, task name, name of the MySQL user who created the task, MySQL connection ID used to create the task, task type, application-managed user ID used to create the task, and any additional metadata related to the task.
- Logs and status: stores details about task progress and log messages.
- APIs: stores the routines for creating, monitoring, and terminating asynchronous tasks.

The `mysql_tasks.config` table stores the following key values that are used to define maximum task limits:

- The `limits.maximumPreparedStmtAsyncTasks` key specifies the maximum number of SQL statements that can run concurrently within the MySQL HeatWave Cluster. This value can be between `1` and `128`.
- The `limits.maximumHeatwaveLoadingTasks` key specifies the maximum number of `VECTOR_STORE_LOAD` tasks that can run concurrently for loading unstructured data into the MySQL HeatWave Cluster. This value can be between `1` and `10`.

  The value specified for the `limits.maximumHeatwaveLoadingTasks` key cannot exceed the value of the `limits.maximumPreparedStmtAsyncTasks` key.

Update Maximum Task Limits

To view or update the maximum task limits, perform the following steps:

- View the maximum number of concurrent tasks you can run:

```
mysql> SELECT * from mysql_tasks.config;
```

  The output is similar to the following:

```
+----+-------------------------------------------------------------------------------+
| id | data                                                                          |
+----+-------------------------------------------------------------------------------+
```

```
|  1 | {"limits": {"maximumHeatwaveLoadingTasks": 5, "maximumPreparedStmtAsyncTasks": 100}} |
+----+--------------------------------------------------------------------------------------+
```

- Update the maximum number of concurrent task limits:

```
mysql> UPDATE mysql_tasks.config
SET data = '{"limits": {"maximumHeatwaveLoadingTasks": MaxVectoStoreLoadTasks, "maximumPreparedStmtAs
WHERE id = 1;
```

Replace *`MaxVectoStoreLoadTasks`* with the maximum limit for `VECTOR_STORE_LOAD` tasks and *`MaxSQLTasks`* with the maximum limit for SQL statements. For example:

```
mysql> UPDATE mysql_tasks.config
SET data = '{"limits": {"maximumHeatwaveLoadingTasks": 10, "maximumPreparedStmtAsyncTasks": 10}}'
WHERE id = 1;
```

Run SQL Statements Asynchronously

Perform the following steps:

1. To run SQL statements asynchronously, use the `execute_prepared_stmt_async` routine:

```
mysql> CALL mysql_tasks.execute_prepared_stmt_async ('SQLStatements', 'DBName', 'TaskName', 'TaskDat
```

Replace the following:

- *`SQLStatements`*: one or more SQL statements separated by a semi-colon (`;`).
- *`DBName`*: database name to run the SQL statements. If you are using a specific database in the current session, you can set this to `NULL` to continue using the current session database.
- *`TaskName`*: name for the asynchronous task. If set to `NULL`, `execute_prepared_stmt_async` is used as the default task name.
- *`TaskData`*: any additional metadata related to the task represented as JSON data. If there is no metadata, set to `NULL`.

The statement returns a unique task identifier (task ID) for the asynchronous task that is created, which is stored in the `@task_id` variable.

For example:

```
mysql> CALL mysql_tasks.execute_prepared_stmt_async('DO SLEEP(5);
SELECT JSON_OBJECT("Async Task completed at", now()) into @task_result', 'demo_db', 'DemoAsyncTask',
```

In this example:

- The first SQL statement `DO SLEEP(5)` simulates a time-consuming task. This can be any long-running task.
- The second statement generates an output that is stored in the reserved session variable `@task_result`. You can optionally use the reserved session variable `@task_result` to add a custom task status message for the asynchronous task, or to store an output generated by an SQL statement that you are running asynchronously. The value that you want to store in `@task_result` must be in JSON format.
- The task ID generated for the asynchronous task that is returned by the routine is stored in the session variable `@task_id`.

2. View the task ID that is stored in the session variable `@task_id`:

```
mysql> SELECT @task_id;
```

The output is similar to the following:

```
17efb56b-52a1-11f0-bca4-020017205654
```

This task ID is independent of the session. You can use it to monitor or terminate the task from any SQL session that you own.

3. View the asynchronous task status brief:

```
mysql> SELECT JSON_PRETTY(mysql_tasks.task_status_brief(@task_id));
```

The output is similar to the following:

```
| {
  "data": {
    "Async Task completed at": "2025-06-26 14:17:38.000000"
  },
  "status": "COMPLETED",
  "message": "Execution finished.",
  "progress": 100
} |
```

Notice that the JSON value stored in `@task_result` is included in the task status `data` field.

Note

Tasks that are completed, canceled, or encountered an error are automatically removed after six days.

To asynchronously run SQL statements on an application level using an application-managed user ID, use the `execute_prepared_stmt_from_app_async` routine. This routine includes additional parameters that you can use to customise the format for storing task status and log data. Similarly, to view the status brief of an application-level asynchronous task created by an application user, use the `app_task_status_brief` routine.

What's Next

Learn more about Asynchronous Task Routines.

5.8 Optimize Workloads for OLAP

Workload optimization for online analytical processing, OLAP, includes using dictionary encoding for certain string columns and defining data placement keys to optimize for `JOIN` and `GROUP BY` query performance. Apply these optimizations manually, or use MySQL HeatWave Autopilot Advisor, which includes the following:

- Auto Encoding recommends string column encodings that minimize the required cluster size and improve query performance.
- Auto Data Placement recommends data placement keys that optimize `JOIN` and `GROUP BY` query performance.
- Auto Query Time Estimation provides runtime information for successfully executed queries and runtime estimates for `EXPLAIN` queries, queries cancelled with `Ctrl+C`, and queries that fail due to out of memory errors. Runtime data is useful for query optimization, troubleshooting, and estimating the cost of running a particular query or workload.
- Unload Advisor

  Recommends tables to unload, reducing MySQL HeatWave memory usage. The recommendations are based upon when the tables were last queried.

Advisor is workload-aware and provides recommendations based on machine learning models, data analysis, and MySQL HeatWave query history. Advisor analyzes the last 1000 successfully executed MySQL HeatWave queries.

Advisor is implemented as a stored procedure named `heatwave_advisor`, which resides in the MySQL `sys` schema.

To run Advisor for OLAP workloads, the MySQL HeatWave Cluster must be active, and the user must have the following MySQL privileges:

- The `PROCESS` privilege.
- The `EXECUTE` privilege on the `sys` schema.
- The `SELECT` privilege on the Performance Schema.

5.8.1 String Column Encoding

MySQL HeatWave supports two string column encoding types:

- Variable-length encoding: This is the default encoding type. It applies to string columns by default when tables are loaded with the exception of string columns defined explicitly as dictionary-encoded columns.
- Dictionary encoding: This is suited for string columns with a low number of distinct values relative to the cardinality of the table.

To use dictionary encoding, you must define the encoding type explicitly for individual string columns. See Section 5.8.1.4, “Encoding String Columns”.

5.8.1.1 Variable-length Encoding

Variable-length encoding is automatically applied when tables are loaded into MySQL HeatWave with the exception of string columns defined explicitly as dictionary-encoded columns. . You can define the encoding type while creating the table or later modify it using `ALTER TABLE` statement.

`VARLEN` columns can be declared as NULL.

Variable-length (`VARLEN`) encoding has the following characteristics:

- Supports all character sets and collation types supported by the DB System, and expressions with non-boolean types. User defined character sets are not supported.
- Minimizes the amount of data stored for string columns by efficiently storing variable length column values.
- Permits more operations involving string columns to be offloaded than dictionary encoding.
- Is more efficient than dictionary encoding with respect to storage and processing of string columns with a high number of distinct values relative to the cardinality of the table.

For information about MySQL HeatWave column limits and how they relate to VARLEN encoded columns, see Column Limits.

VARLEN Column Memory Requirements

- For MySQL HeatWave nodes, a `VARLEN` encoded column value requires enough memory for the data plus two bytes for length information. Internal fragmentation or headers can affect the actual amount of memory required.
- For DB System node, there is no memory requirement, apart from a small memory footprint for metadata.

VARLEN Supported Functions, and Operators

For supported functions and operators, refer to Section 5.3, “Supported Functions and Operators”.

VARLEN Supported Filters

VARLEN supports the following filters:

- Column-to-column filters, excluding the `<=>` filter.
- Column-to-constant filters, excluding the `<=>` filter.

VARLEN Supported Relational Operators

- `GROUP BY`
- `JOIN`
- `LIMIT`
- `ORDER BY`

VARLEN Encoding and Performance

- The presence of `VARLEN` encoded `VARCHAR` or `CHAR` columns does not affect table load performance.
- Table load and change propagation operations perform more slowly on `VARLEN` encoded `TEXT` type columns than on `VARLEN` encoded `VARCHAR` columns.
- There are two main differences with respect to MySQL HeatWave result processing for variable-length encoding compared to dictionary encoding:
  - A dictionary decode operation is not required, which means that fewer CPU cycles are required.
  - Because `VARLEN` encoded columns use a larger number of bytes than dictionary-encoded columns, the network cost for sending results from MySQL HeatWave to the DB System is greater.

What's Next

Learn more about Dictionary Encoding.

5.8.1.2 Dictionary Encoding

Dictionary encoding (`SORTED`) has the following characteristics:

- Best suited to string columns with a low number of distinct values relative to the cardinality of the table. Dictionary encoding reduces the space required for column values on the MySQL HeatWave nodes but requires space on the DB System node for dictionaries.
- Supports `GROUP BY` and `ORDER BY` operations on string columns.
- Supports a subset of the operations supported by variable-length encoding such as `LIKE` with prefix expressions, and comparison with the exact same column. Dictionary-encoded columns cannot be compared in any way with other columns or constants, or with other dictionary-encoded columns.
- Does not support:
  - `JOIN` operations
  - `LIKE` predicates.
  - Operations that use string operators. Queries that use string operators on dictionary-encoded string columns are not offloaded.
  - Comparison with variable-length encoded columns.

- The dictionaries required to decode dictionary-encoded string columns must fit in DB System node memory. Dictionary size depends on the size of the column and the number of distinct values. Load operations for tables with dictionary-encoded string columns that have a high number of distinct values can fail if there is not enough available memory on the DB System node.

What's Next

Learn more about Column Limits.

5.8.1.3 Column Limits

MySQL HeatWave has the following column limits:

- As of MySQL 9.2.2, the maximum column width is 4192192 bytes.

  In previous versions of MySQL, the maximum column width is 65532 bytes.

  The maximum column widths supported for different string and text data types are as follows:

  Table 5.14 Supported Column Widths

String/Text Datatype	Column Width Supported from MySQL 9.2.2	Column Width Supported Prior to MySQL 9.2.2
`MEDIUMTEXT`	4192192 bytes	65532 bytes
`LONGTEXT`	4192192 bytes	65532 bytes
`JSON`	4192192 bytes	65532 bytes
`TEXT`	65535 bytes	65532 bytes
`CHAR`	65532 bytes	65532 bytes
`VARCHAR`	65532 bytes	65532 bytes
`VECTOR`	65532 bytes	65532 bytes

  Note

  The maximum row width is 4 MB. This includes the size of all the data and column metadata stored in the row. Any table with multiple columns wherein the aggregate data size is greater than 4 MB cannot be loaded to MySQL HeatWave.

  This applies to intermediate relations and final query results as well.

- The maximum column limit for base relations, tables as loaded into MySQL HeatWave, is 1017.
- The column limit for intermediate relations (intermediate tables used by MySQL HeatWave when processing queries) is 1800.
- The actual column limit when running queries depends on factors such as MySQL limits, protocol limits, the total number of columns, column types, and column widths.

What's Next

Learn how to encode string columns.

5.8.1.4 Encoding String Columns

This topic describes how to encode string columns to improve the performance of queries that access them.

This topic contains the following sections:

- Before You Begin

- Supported Encoding Types
- Select the Encoding Type
- Define String Column Encoding
- View String Column Encoding
- What's Next

Before You Begin

- Review the requirements.
- Connect to your DB System.
- The examples in this topic use the sample database `airportdb`. To learn how to download the sample database, see AirportDB Analytics Quickstart.

Supported Encoding Types

MySQL HeatWave supports Variable-length encoding(`VARLEN`) and Dictionary encoding (`SORTED`).

The following table provides an overview of encoding type characteristics:

Table 5.15 Column Encoding Type Characteristics

Encoding Type	Expression, Filter, Function, and Operator Support	Best Suited To	Space Required On
`Variable-length (VARLEN)`	Supports `JOIN` operations, string functions and operators, and `LIKE` predicates.	Columns with a high number of distinct values	MySQL HeatWave nodes
`Dictionary (SORTED)`	Supports `GROUP BY` and `ORDER BY` operations on string columns	Columns with a low number of distinct values	DB System node

For additional information about string column encoding, see Section 5.8.1, “String Column Encoding”.

Select the Encoding Type

By default, when tables are loaded into MySQL HeatWave, `CHAR`, `VARCHAR`, and `TEXT` type columns are encoded using variable-length encoding. To use dictionary encoding, define the `RAPID_COLUMN=ENCODING=SORTED` keyword string in a column comment before loading the table.

Select one of the encoding type, based on the query pattern and the cardinality of the column.

Note

The keyword string must be uppercase. Lowercase and mixed-case strings are ignored.

- Variable-length encoding(`VARLEN`)
  - To run `JOIN` operations involving string columns or use string functions and operators and, `LIKE` predicates.
  - To encode columns with a high number of distinct values, such as `comment` columns.
  - When you require more space for column values on the MySQL HeatWave nodes.

- Dictionary encoding (`SORTED`)
  - To run expression, filter, function, and operator support than dictionary encoding.
  - To encode columns with a low number of distinct values, such as `country` columns.
  - When you require space on the DB System node for dictionaries.

Define String Column Encoding

You can define the keyword string in a `CREATE TABLE` or `ALTER TABLE` statement, as shown:

```
mysql> CREATE TABLE airline_info (name VARCHAR(100)
          COMMENT 'RAPID_COLUMN=ENCODING=SORTED');
Query OK, 0 rows affected (0.2363 sec)
```

```
mysql> ALTER TABLE airline_info MODIFY name VARCHAR(100)
          COMMENT 'RAPID_COLUMN=ENCODING=SORTED';
Query OK, 0 rows affected (0.2284 sec)
```

If necessary, you can specify variable-length encoding explicitly using the `RAPID_COLUMN=ENCODING=VARLEN` keyword string.

Note

Other information is permitted in column comments. For example, it is permitted for a column description to be specified alongside a column encoding keyword string:

`COMMENT 'column_description RAPID_COLUMN=ENCODING=SORTED'`

Tip

For string column encoding recommendations, use MySQL HeatWave Autopilot Advisor after loading tables into MySQL HeatWave and running queries. For more information, see Section 5.8, “Optimize Workloads for OLAP”.

To modify or remove a string column encoding, refer to the procedure described in Section 5.5, “Modify Tables”.

View String Column Encoding

- To identify explicitly encoded string columns in tables on the DB System, query the `COLUMN_COMMENT` column of the `INFORMATION_SCHEMA.COLUMNS` table. For example:

```
mysql>  SELECT COLUMN_NAME, COLUMN_COMMENT FROM INFORMATION_SCHEMA.COLUMNS WHERE TABLE_NAME LIKE 'air
+-------------+------------------------------+
| COLUMN_NAME | COLUMN_COMMENT               |
+-------------+------------------------------+
| name        | RAPID_COLUMN=ENCODING=SORTED |
+-------------+------------------------------+
```

  You can also view explicitly defined column encodings for an individual table using `SHOW CREATE TABLE`.

- To view the dictionary size for dictionary-encoded columns, in bytes:

```
mysql> USE performance_schema;
mysql> SELECT rpd_table_id.TABLE_NAME, rpd_columns.COLUMN_ID, rpd_columns.DICT_SIZE_BYTES
          FROM rpd_table_id, rpd_columns
          WHERE rpd_table_id.ID = rpd_columns.TABLE_ID AND rpd_columns.DICT_SIZE_BYTES > 0
          ORDER BY rpd_table_id.TABLE_NAME;
---------------+-----------+-----------------+
| TABLE_NAME    | COLUMN_ID | DICT_SIZE_BYTES |
+--------------+-----------+-----------------+
| airline_info |         3 |        25165912 |
+--------------+-----------+-----------------+
```

What's Next

Data Placement Keys.

5.8.2 Data Placement Keys

When data is loaded into MySQL HeatWave, it is partitioned by the table primary key and sliced horizontally for distribution among MySQL HeatWave nodes by default. However, this may not be optimal for queries involving `JOIN` or `GROUP BY` operations as they incur costs due to data redistribution among MySQL HeatWave nodes at query execution time

In case of such queries, the data placement key feature permits partitioning data by `JOIN` or `GROUP BY` key columns instead of primary keys. To use this feature, define data placement keys on the most frequently used `JOIN` keys and the keys of the longest running queries.

Tip

For data placement key recommendations, use MySQL HeatWave Autopilot Advisor after loading tables into MySQL HeatWave and running queries. For more information, see Section 5.8, “Optimize Workloads for OLAP”.

This topic includes the following sections:

- Before You Begin
- Define a Data Placement Key
- View Data Placement Keys
- Verify Data Placement Partition Usage for JOIN and GROUP BY Queries
- Important Notes
- What's Next

Before You Begin

- Review the requirements.
- Connect to your MySQL DB System.
- The examples in this topic use the sample database `airportdb`. To learn how to download the sample database, see AirportDB Analytics Quickstart.

Define a Data Placement Key

Defining a data placement key requires adding a column comment with the data placement keyword string:

```
RAPID_COLUMN=DATA_PLACEMENT_KEY=N
```

where *`N`* is an index value that defines the priority order of data placement keys. The rules for assigning a data placement index are:

- The index must start with 1.
- Index values range from 1 to 16.
- Each column must have an unique index value. For example, you cannot assign the index value of 2 to more than one column in the same table.
- Always assign consecutive index values, without gaps.For example, if you define a data placement key column with an index value of 3, there must also be two other data placement key columns with index values of 1 and 2, respectively.

The following example shows multiple columns defined as data placement keys. Although a primary key is defined, data is partitioned by the data placement keys, which are prioritized over the primary key.

```
mysql> CREATE TABLE airline_discount(
id INT PRIMARY KEY,
date DATE COMMENT 'RAPID_COLUMN=DATA_PLACEMENT_KEY=1',
discount FLOAT COMMENT 'RAPID_COLUMN=DATA_PLACEMENT_KEY=2');
Query OK, 0 rows affected (0.2448 sec)
```

When defining multiple columns as data placement keys, prioritize the keys according to query cost. For example, assign `DATA_PLACEMENT_KEY=1` to the key of the costliest query, and `DATA_PLACEMENT_KEY=2` to the key of the second costliest query, and so on.

Note

Other information is permitted in column comments. For example, it is permitted to specify a column description alongside a data placement keyword string:

```
COMMENT 'column_description RAPID_COLUMN=DATA_PLACEMENT_KEY=1'
```

View Data Placement Keys

To view the data placement type of the tables loaded into MySQL HeatWave, `PrimaryKey` means the data placement key is using the primary key of the table:

```
mysql> SELECT SCHEMA_NAME, TABLE_NAME, DATA_PLACEMENT_TYPE, LOAD_STATUS
FROM performance_schema.rpd_table_id
JOIN performance_schema.rpd_tables USING (ID);
```

You can also view data placement keys for an individual table using SHOW CREATE TABLE.

For information about data placement key index values, see Define a Data Placement Key.

Verify Data Placement Partition Usage for JOIN and GROUP BY Queries

To determine whether a `JOIN` or `GROUP BY` query used data placement partitions, examine the `QEP_TEXT` column in the `performance_schema.rpd_query_stats` table. This column contains `prepart` (pre-partitioning) data, which indicates how the query was distributed across MySQL HeatWave nodes.

For `GROUP BY` operations, the `prepart` data is displayed as:

```
"prepart": #
```

Where, # represents the number of nodes.

For `JOIN` operations, the `prepart` data appears as:

```
"prepart": [#, #]
```

where `[#, #]` represents the number of MySQL HeatWave nodes used in the left and the right branches respectively. If either value is greater than 1, that branch used data placement partitions. If both values are 1 (i.e., "prepart": [1, 1]), then data placement partitions is not used both the branches.

To query `QEP_TEXT` `prepart` data for the last executed query:

```
mysql> SELECT CONCAT( '"prepart":[', (JSON_EXTRACT(QEP_TEXT->>"$**.prepart", '$[0][0]')),
          "," ,(JSON_EXTRACT(QEP_TEXT->>"$**.prepart", '$[0][1]')) , ']' )
          FROM performance_schema.rpd_query_stats
          WHERE query_id = (select max(query_id)
          FROM performance_schema.rpd_query_stats);
+-------------------------------------------------------------------------+
| concat( '"prepart":[', (JSON_EXTRACT(QEP_TEXT->>"$**.prepart", '$[0][0]')), |
|"," ,(JSON_EXTRACT(QEP_TEXT->>"$**.prepart", '$[0][1]')) , ']' )          |
+-------------------------------------------------------------------------+
```

```
| "prepart":[2,2]                                                                |
+--------------------------------------------------------------------------------+
```

Refer to the following example to understand how to use MySQL HeatWave data placement keys to improve query performance by controlling data distribution and verifying execution plan efficiency.

1. To begin with, check whether the target table (airline_discount) has any existing data placement keys, using SELECT statement:

```
mysql> SELECT COLUMN_NAME, COLUMN_COMMENT
      FROM INFORMATION_SCHEMA.COLUMNS
WHERE
      TABLE_NAME LIKE 'airline_discount'
AND
      COLUMN_COMMENT LIKE '%DATA_PLACEMENT_KEY%';/
   add output
```

2. Modify the table (airline_discount) to add (airline_discount)the from and to columns as placement keys, so that MySQL HeatWave can locate related data across nodes and reduce data movement during joins.

```
mysql> ALTER TABLE flight
      MODIFY `from` smallint NOT NULL COMMENT 'RAPID_COLUMN=DATA_PLACEMENT_KEY=1',
      MODIFY `to` smallint NOT NULL COMMENT 'RAPID_COLUMN=DATA_PLACEMENT_KEY=2';
Add output
```

3. Use the following query to inspect the number of partitions used in the most recent offloaded query:

```
mysql> SELECT QEP_TEXT->>"$**.nPreparts"  FROM performance_schema.rpd_query_stats
       WHERE query_id = (select max(query_id) FROM performance_schema.rpd_query_stats);

              Add output
```

   The `nPreparts` value represents the number of partitions involved. A lower `nPreparts` value indicates more efficient data placement and reduced inter-node communication.

Important Notes

- `JOIN` and `GROUP BY` query optimizations are only applied if at least one of the `JOIN` or `GROUP BY` relations has a key that matches the defined data placement key.
- If you execute a `JOIN` operation with or without the `JOIN` and `GROUP BY` query optimization, a compilation-time cost model determines how the query is executed. The cost model uses estimated statistics.
- A data placement key cannot be defined on a dictionary-encoded string column but are permitted on variable-length encoded columns. MySQL HeatWave applies variable-length encoding to string columns by default. See Section 5.8.1.4, “Encoding String Columns”.
- A data placement key can only be defined on a column with a supported data type. See Section 4.2.1, “Supported Data Types for MySQL HeatWave”.
- A data placement key column cannot be defined as a `NOT SECONDARY` column. See Section 4.2.6.1, “Exclude Table Columns”.

What's Next

Learn about Autopilot Advisor.

5.8.3 About MySQL HeatWave Autopilot Advisor

The MySQL HeatWave Autopilot improves MySQL database performance without manual tuning. It uses machine learning to study how the database is used and automatically recommends or applies optimizations such as indexing and data compression. This automation helps save time and ensures better efficiency, especially when dealing with complex analytical (OLAP) workloads.

What's Next

- Learn about the following MySQL HeatWave Autopilot Advisor features:
  - Auto encoding
  - Auto data placement
  - Auto query time estimation.
  - Unload advisor.
- Learn more about MySQL HeatWave Autopilot Advisor Syntax.

5.8.4 Auto Encoding

Auto Encoding recommends string column encodings. Using the right string column encodings can reduce the amount of memory required on MySQL HeatWave nodes and improve query performance. MySQL HeatWave supports two string column encoding types: variable-length and dictionary. MySQL HeatWave applies variable-length encoding to string columns by default when data is loaded. Auto Encoding generates string column encoding recommendations by analyzing column data, MySQL HeatWave query history, query performance data, and available memory on the MySQL node. For more information about string column encoding, see Section 5.8.1.4, “Encoding String Columns”.

This topic contains the following sections:

- Run Auto Encoding
- Usage Notes
- Auto Encoding Examples
- What's Next

Run Auto Encoding

To enable Auto Encoding, specify the `auto_enc` option in `recommend` mode. See Section 10.1.5, “HEATWAVE_ADVISOR”.

Note

To run Advisor for both encoding and data placement recommendations, run Auto Encoding first, apply the recommended encodings, rerun the queries, and then run Auto Data Placement. This sequence allows data placement performance benefits to be calculated with string column encodings in place, which provides for greater accuracy from Advisor internal models.

For Advisor to provide string column encoding recommendations, tables must be loaded in MySQL HeatWave and a query history must be available. Run the queries that you intend to use or run a representative set of queries. Failing to do so can affect query offload after Auto Encoding recommendations are implemented due to query constraints associated with dictionary encoding. For dictionary encoding limitations, see Section 5.8.1.2, “Dictionary Encoding”.

In the following example, Auto Encoding is run in `recommend` mode, which analyzes column data, checks the amount of memory on the MySQL node, and provides encoding recommendations intended to reduce the amount of space required on MySQL HeatWave nodes and optimize query performance. There is no target schema specified, so Auto Encoding runs on all schemas loaded in MySQL HeatWave.

```
mysql> CALL sys.heatwave_advisor(JSON_OBJECT('auto_enc',JSON_OBJECT('mode','recommend')));
```

The `fixed_enc` option can be used in `recommend` mode to specify an encoding for specific columns. These columns are excluded from consideration when Auto Encoding generates recommendations.

Manually encoded columns are also excluded from consideration. (For manual encoding instructions, see Section 5.8.1.4, “Encoding String Columns”.)

```
mysql> CALL sys.heatwave_advisor(JSON_OBJECT('auto_enc',JSON_OBJECT('mode','recommend','fixed_enc',
         JSON_OBJECT('tpch.CUSTOMER.C_ADDRESS','varlen'))));
```

Advisor output provides information about each stage of Advisor execution, including recommended column encodings and estimated MySQL HeatWave Cluster memory savings.

```
mysql> CALL sys.heatwave_advisor(JSON_OBJECT('target_schema',JSON_ARRAY('tpch_1024'),
       'auto_enc',JSON_OBJECT('mode','recommend')));
+------------------------------+
| INITIALIZING HEATWAVE ADVISOR |
+------------------------------+
| Version: 1.12                |
|                              |
| Output Mode: normal          |
| Excluded Queries: 0          |
| Target Schemas: 1            |
|                              |
+------------------------------+
6 rows in set (0.01 sec)
+-----------------------------------------------------------+
| ANALYZING LOADED DATA                                     |
+-----------------------------------------------------------+
| Total 8 tables loaded in HeatWave for 1 schemas           |
| Tables excluded by user: 0 (within target schemas)        |
|                                                           |
| SCHEMA                              TABLES        COLUMNS |
| NAME                                LOADED         LOADED |
| ------                              ------         ------ |
| `tpch_1024`                              8             61 |
|                                                           |
+-----------------------------------------------------------+
8 rows in set (0.15 sec)

+-----------------------------------------------------------------------------------------------+
| ENCODING SUGGESTIONS                                                                          |
+-----------------------------------------------------------------------------------------------+
| Total Auto Encoding suggestions produced for 21 columns                                       |
| Queries executed: 200                                                                         |
|   Total query execution time: 28.82 min                                                       |
|   Most recent query executed on: Tuesday 8th June 2021 14:42:13                               |
|   Oldest query executed on: Tuesday 8th June 2021 14:11:45                                    |
|                                                                                               |
|                                                      CURRENT           SUGGESTED              |
| COLUMN                                                COLUMN              COLUMN              |
| NAME                                                 ENCODING            ENCODING             |
| ------                                               --------           ---------            |
| `tpch_1024`.`CUSTOMER`.`C_ADDRESS`                     VARLEN          DICTIONARY             |
| `tpch_1024`.`CUSTOMER`.`C_COMMENT`                     VARLEN          DICTIONARY             |
| `tpch_1024`.`CUSTOMER`.`C_MKTSEGMENT`                  VARLEN          DICTIONARY             |
| `tpch_1024`.`CUSTOMER`.`C_NAME`                        VARLEN          DICTIONARY             |
| `tpch_1024`.`LINEITEM`.`L_COMMENT`                     VARLEN          DICTIONARY             |
| `tpch_1024`.`LINEITEM`.`L_SHIPINSTRUCT`                VARLEN          DICTIONARY             |
| `tpch_1024`.`LINEITEM`.`L_SHIPMODE`                    VARLEN          DICTIONARY             |
| `tpch_1024`.`NATION`.`N_COMMENT`                       VARLEN          DICTIONARY             |
| `tpch_1024`.`NATION`.`N_NAME`                          VARLEN          DICTIONARY             |
| `tpch_1024`.`ORDERS`.`O_CLERK`                         VARLEN          DICTIONARY             |
| `tpch_1024`.`ORDERS`.`O_ORDERPRIORITY`                 VARLEN          DICTIONARY             |
| `tpch_1024`.`PART`.`P_BRAND`                           VARLEN          DICTIONARY             |
| `tpch_1024`.`PART`.`P_COMMENT`                         VARLEN          DICTIONARY             |
| `tpch_1024`.`PART`.`P_CONTAINER`                       VARLEN          DICTIONARY             |
| `tpch_1024`.`PART`.`P_MFGR`                            VARLEN          DICTIONARY             |
| `tpch_1024`.`PARTSUPP`.`PS_COMMENT`                    VARLEN          DICTIONARY             |
| `tpch_1024`.`REGION`.`R_COMMENT`                       VARLEN          DICTIONARY             |
| `tpch_1024`.`REGION`.`R_NAME`                          VARLEN          DICTIONARY             |
| `tpch_1024`.`SUPPLIER`.`S_ADDRESS`                     VARLEN          DICTIONARY             |
| `tpch_1024`.`SUPPLIER`.`S_NAME`                        VARLEN          DICTIONARY             |
| `tpch_1024`.`SUPPLIER`.`S_PHONE`                       VARLEN          DICTIONARY             |
|                                                                                               |
| Applying the suggested encodings might improve query performance and cluster memory usage.|
```

```
|   Estimated HeatWave Cluster memory savings: 355.60 GiB                                          |
|                                                                                                  |
+--------------------------------------------------------------------------------------------------+
35 rows in set (18.18 sec)

+--------------------------------------------------------------------------------------------------+
| SCRIPT GENERATION                                                                                |
+--------------------------------------------------------------------------------------------------+
| Script generated for applying suggestions for 8 loaded tables                                    |
|                                                                                                  |
| Applying changes will take approximately 1.64 h                                                  |
|                                                                                                  |
| Retrieve script containing 61 generated DDL commands using the query below:                      |
|   SELECT log->>"$.sql" AS "SQL Script" FROM sys.heatwave_advisor_report WHERE type = "sql"|
|   ORDER BY id;                                                                                   |
|                                                                                                  |
| Caution: Executing the generated script will alter the column comment and secondary engine|
| flags in the schema                                                                              |
|                                                                                                  |
+--------------------------------------------------------------------------------------------------+
9 rows in set (18.20 sec)
```

To inspect the load script, which includes the DDL statements required to implement the recommended encodings, query the `heatwave_autopilot_report` table.

```
mysql> SELECT log->>"$.sql" AS "SQL Script"
          FROM sys.heatwave_autopilot_report
          WHERE type = "sql"
          ORDER BY id;
```

To concatenate generated DDL statements into a single string that can be copied and pasted for execution, issue the statements that follow. The `group_concat_max_len` variable sets the result length in bytes for the `GROUP_CONCAT()` function to accommodate a potentially long string. (The default `group_concat_max_len` setting is 1024 bytes.)

```
mysql> SET SESSION group_concat_max_len = 1000000;
mysql> SELECT GROUP_CONCAT(log->>"$.sql" SEPARATOR ' ')
          FROM sys.heatwave_advisor_report
          WHERE type = "sql"
          ORDER BY id;
```

Usage Notes

- Auto Encoding analyzes string columns (`CHAR`, `VARCHAR`, and `TEXT` type columns) of tables that are loaded in MySQL HeatWave. Automatically or manually excluded columns, columns greater than 65532 bytes, and columns with manually defined encodings are excluded from consideration. Auto Encoding also analyzes MySQL HeatWave query history to identify query constraints that preclude the use of dictionary encoding. Dictionary-encoded columns are not supported in `JOIN` operations, with string functions and operators, or in `LIKE` predicates. For dictionary encoding limitations, see Section 5.8.1.2, “Dictionary Encoding”.
- The time required to generate encoding recommendations depends on the number of queries to be analyzed, the number of operators, and the complexity of each query.
- Encoding recommendations for the same table may differ after changes to data or data statistics. For example, changes to table cardinality or the number of distinct values in a column can affect recommendations.
- Auto Encoding does not generate recommendations for a given table if existing encodings do not require modification.
- Auto Encoding only recommends dictionary encoding if it is expected to reduce the amount of memory required on MySQL HeatWave nodes.
- If there is not enough MySQL node memory for the dictionaries of all columns that would benefit from dictionary encoding, the columns estimated to save the most memory are recommended for dictionary encoding.

- Auto Encoding uses the current state of tables loaded in MySQL HeatWave when generating recommendations. Concurrent change propagation activity is not considered.
- Encoding recommendations are based on estimates and are therefore not guaranteed to reduce the memory required on MySQL HeatWave nodes or improve query performance.

Auto Encoding Examples

- Running Auto Encoding to generate string column encoding recommendations for the `airportdb` schema:

```
mysql> CALL sys.heatwave_advisor(JSON_OBJECT('target_schema',JSON_ARRAY('tpch'),
          'auto_enc',JSON_OBJECT('mode','recommend')));
```

- Running Auto Encoding with the `fixed_enc` option to force variable-length encoding for the `tpch.CUSTOMER.C_ADDRESS` column. Columns specified by the `fixed_enc` option are excluded from consideration by the Auto Encoding feature.

```
mysql> CALL sys.heatwave_advisor(JSON_OBJECT('target_schema',JSON_ARRAY('tpch'),
          'auto_enc',JSON_OBJECT('mode','recommend','fixed_enc',
           JSON_OBJECT('tpch.CUSTOMER.C_ADDRESS','varlen'))));
```

What's Next

Learn more about Auto Data Placement.

5.8.5 Auto Data Placement

Auto Data Placement generates data placement key recommendations. Data placement keys are used to partition table data among MySQL HeatWave nodes when loading tables. Partitioning table data by `JOIN` and `GROUP BY` key columns can improve query performance by avoiding costs associated with redistributing data among MySQL HeatWave nodes at query execution time. The Data Placement Advisor generates data placement key recommendations by analyzing table statistics and MySQL HeatWave query history. For more information about data placement keys, see Section 5.8.2, “Data Placement Keys”.

This topic contains the following sections:

- Run Auto Data Placement
- Usage Notes
- Auto Data Placement Examples
- What's Next

Run Auto Data Placement

Note

To run Advisor for both encoding and data placement recommendations, run Auto Encoding first, apply the recommended encodings, rerun the queries, and then run Auto Data Placement. This sequence allows data placement performance benefits to be calculated with string column encodings in place, which provides for greater accuracy from Advisor internal models.

For Advisor to generate data placement recommendations:

- Tables must be loaded in MySQL HeatWave.
- There must be a query history with at least 5 queries. A query is counted if it includes a `JOIN` on tables loaded in the MySQL HeatWave Cluster or `GROUP BY` keys. A query executed on a table that is no longer loaded or that was reloaded since the query was run is not counted.

To view the query history, query the `performance_schema.rpd_query_stats` table. For example:

```
mysql> SELECT query_id, LEFT(query_text,160)
         FROM performance_schema.rpd_query_stats;
*************************** 1. row ***************************
           query_id: 14
LEFT(query_text,160): SELECT country from airport_geo
*************************** 2. row ***************************
           query_id: 15
LEFT(query_text,160): CREATE TABLE flight_from_US SELECT * FROM flight WHERE `from` IN (SELECT airpor
*************************** 3. row ***************************
           query_id: 16
LEFT(query_text,160): CREATE TABLE flight_from_US SELECT * FROM flight WHERE `from` IN (SELECT airpor
*************************** 4. row ***************************
           query_id: 17
LEFT(query_text,160): CREATE TABLE flight_from_US SELECT * FROM flight WHERE `from` IN (SELECT airpor
...
...
9 rows in set (0.2179 sec)
```

For the most accurate data placement recommendations, run Advisor on one schema at a time. In the following example, Advisor is run on the `airportdb` schema using the `target_schema` option. No other *`options`* are specified, which means that the default option settings are used.

```
mysql> CALL sys.heatwave_advisor(JSON_OBJECT('target_schema',JSON_ARRAY('tpch_1024')));
```

Advisor output provides information about each stage of Advisor execution. The data placement suggestion output shows suggested data placement keys and the estimated performance benefit of applying the keys.

When you run the MySQL HeatWave Advisor, it provides a query to retrieve the generated DDL statements needed to implement the recommended data placement keys. You cannot add or modify data placement keys without reloading the table. Therefore, the Advisor creates DDL statements to unload the table, apply the new keys, and then reload the table.

```
mysql> CALL sys.heatwave_advisor(JSON_OBJECT('target_schema',JSON_ARRAY('tpch_1024')));
+------------------------------+
| INITIALIZING HEATWAVE ADVISOR |
+------------------------------+
| Version: 1.12                |
|                              |
| Output Mode: normal          |
| Excluded Queries: 0          |
| Target Schemas: 1            |
|                              |
+------------------------------+
6 rows in set (0.01 sec)

+-----------------------------------------------------------+
| ANALYZING LOADED DATA                                     |
+-----------------------------------------------------------+
| Total 8 tables loaded in HeatWave for 1 schemas           |
| Tables excluded by user: 0 (within target schemas)        |
|                                                           |
| SCHEMA                            TABLES        COLUMNS   |
| NAME                              LOADED         LOADED   |
| ------                            ------         ------   |
| `tpch_1024`                            8             61   |
|                                                           |
+-----------------------------------------------------------+
8 rows in set (0.02 sec)

+-------------------------------------------------------------------------+
| AUTO DATA PLACEMENT                                                     |
+-------------------------------------------------------------------------+
| Auto Data Placement Configuration:                                      |
|   Minimum benefit threshold: 1%                                         |
| Producing Data Placement suggestions for current setup:                 |
```

```
|   Tables Loaded: 8                                                    |
|   Queries used: 189                                                   |
|     Total query execution time: 22.75 min                             |
|     Most recent query executed on: Tuesday 8th June 2021 16:29:02     |
|     Oldest query executed on: Tuesday 8th June 2021 16:05:43          |
|   HeatWave cluster size: 5 nodes                                      |
|                                                                       |
| All possible Data Placement combinations based on query history: 120 |
| Explored Data Placement combinations after pruning: 90                |
|                                                                       |
+-----------------------------------------------------------------------+
16 rows in set (12.38 sec)

+-----------------------------------------------------------------------------------------+
| DATA PLACEMENT SUGGESTIONS                                                              |
+-----------------------------------------------------------------------------------------+
| Total Data Placement suggestions produced for 2 tables                                  |
|                                                                                         |
| TABLE                                          DATA PLACEMENT           DATA PLACEMENT |
| NAME                                              CURRENT KEY            SUGGESTED KEY |
| ------                                         --------------           -------------- |
| `tpch_1024`.`LINEITEM`              L_ORDERKEY, L_LINENUMBER               L_ORDERKEY |
| `tpch_1024`.`SUPPLIER`                             S_SUPPKEY              S_NATIONKEY |
|                                                                                         |
| Expected benefit after applying Data Placement suggestions                              |
|   Runtime saving: 6.17 min                                                              |
|   Performance benefit: 27%                                                              |
|                                                                                         |
+-----------------------------------------------------------------------------------------+
12 rows in set (16.42 sec)

+-----------------------------------------------------------------------------------------------+
| SCRIPT GENERATION                                                                             |
+-----------------------------------------------------------------------------------------------+
| Script generated for applying suggestions for 2 loaded tables                                 |
|                                                                                               |
| Applying changes takes approximately 1.18 h                                                   |
|                                                                                               |
| Retrieve script containing 12 generated DDL commands using the query below:                   |
|   SELECT log->>"$.sql" AS "SQL Script" FROM sys.heatwave_advisor_report WHERE type = "sql"|
|   ORDER BY id;                                                                                |
|                                                                                               |
| Caution: Executing the generated script alters the column comment and secondary engine        |
| flags in the schema                                                                           |
|                                                                                               |
+-----------------------------------------------------------------------------------------------+
9 rows in set (16.43 sec)


SELECT log->>"$.sql" AS "SQL Script" FROM sys.heatwave_advisor_report WHERE type = "sql"
 ORDER BY id;
+-----------------------------------------------------------------------------------------------+
| SQL Script                                                                                    |
+-----------------------------------------------------------------------------------------------+
| SET SESSION innodb_parallel_read_threads = 48;                                                |
| ALTER TABLE `tpch_1024`.`LINEITEM` SECONDARY_UNLOAD;                                          |
| ALTER TABLE `tpch_1024`.`LINEITEM` SECONDARY_ENGINE=NULL;                                     |
| ALTER TABLE `tpch_1024`.`LINEITEM` MODIFY `L_ORDERKEY` bigint NOT NULL COMMENT                |
|  'RAPID_COLUMN=DATA_PLACEMENT_KEY=1';                                                         |
| ALTER TABLE `tpch_1024`.`LINEITEM` SECONDARY_ENGINE=RAPID;                                    |
| ALTER TABLE `tpch_1024`.`LINEITEM` SECONDARY_LOAD;                                            |
| SET SESSION innodb_parallel_read_threads = 48;                                                |
| ALTER TABLE `tpch_1024`.`SUPPLIER` SECONDARY_UNLOAD;                                          |
| ALTER TABLE `tpch_1024`.`SUPPLIER` SECONDARY_ENGINE=NULL;                                     |
| ALTER TABLE `tpch_1024`.`SUPPLIER` MODIFY `S_NATIONKEY` int NOT NULL COMMENT                  |
|  'RAPID_COLUMN=DATA_PLACEMENT_KEY=1';                                                         |
| ALTER TABLE `tpch_1024`.`SUPPLIER` SECONDARY_ENGINE=RAPID;                                    |
| ALTER TABLE `tpch_1024`.`SUPPLIER` SECONDARY_LOAD;                                            |
+-----------------------------------------------------------------------------------------------+
12 rows in set (0.00 sec)
```

Usage Notes

- If a table already has data placement keys or columns are customized before running Advisor, Advisor may generate DDL statements for removing previously defined data placement keys.
- Advisor provides recommendations only if data placement keys are estimated to improve query performance. If not, an information message is returned and no recommendations are provided.
- Advisor provides data placement key recommendations based on approximate models. Recommendations are therefore not guaranteed to improve query performance.

Auto Data Placement Examples

- Invoking Advisor without any *`options`* runs the Data Placement Advisor with the default option settings.

```
mysql> CALL sys.heatwave_advisor(NULL);
```

- Running Advisor with only the `target_schema` option runs the Data Placement Advisor on the specified schemas with the default option settings.

```
mysql> CALL sys.heatwave_advisor(JSON_OBJECT("target_schema",JSON_ARRAY("tpch","employees")));
```

- Run Advisor with the data placement `max_combinations` and `benefit_threshold` parameters. For information about these *`options`*, see Section 10.1.5, “HEATWAVE_ADVISOR”.

```
mysql> CALL sys.heatwave_advisor(JSON_OBJECT("auto_dp",
         JSON_OBJECT("max_combinations",100,"benefit_threshold",20)));
```

- The following example shows how to view the MySQL HeatWave query history by querying the `performance_schema.rpd_query_stats` table, and how to exclude specific queries from Data Placement Advisor analysis using the `exclude_query` option:

```
mysql> SELECT query_id, LEFT(query_text,160)
         FROM performance_schema.rpd_query_stats;

mysql> CALL sys.heatwave_advisor(JSON_OBJECT("target_schema",
         JSON_ARRAY("tpch"),"exclude_query",JSON_ARRAY(1,11,12,14)));
```

- This example demonstrates how to invoke the Data Placement Advisor with *`options`* specified in a variable:

```
mysql> SET @options = JSON_OBJECT(
           "target_schema", JSON_ARRAY("analytics45","sample_schema"),
           "exclude_query", JSON_ARRAY(12,24),
           "auto_dp", JSON_OBJECT(
                  "benefit_threshold", 12.5,
                  "max_combinations", 100
));

mysql> CALL sys.heatwave_advisor( @options );
```

- This example demonstrates how to invoke Advisor in silent output mode, which is useful if the output is consumed by a script, for example. Auto Data Placement is run by default if no option such as `auto_enc` or `query_insights` is specified.

```
mysql> CALL sys.heatwave_advisor(JSON_OBJECT("output","silent"));
```

What's Next

Learn more about Auto Query Time Estimation.

5.8.6 Auto Query Time Estimation

Auto Query Time Estimation provides:

- Runtimes for successfully executed queries
- Runtime estimates for `EXPLAIN` queries.
- Runtime estimates for queries cancelled using `Ctrl+C`.
- Runtime estimates for queries that fail due to an out-of-memory error.

Runtime data can be used for query optimization, troubleshooting, or to estimate the cost of running a particular query or workload on MySQL HeatWave.

This topic contains the following sections:

- Run Auto Query Time Estimation
- Auto Query Time Estimation Data
- Auto Query Time Estimation Examples
- What's Next

Run Auto Query Time Estimation

For Auto Query Time Estimation to provide runtime data, a query history must be available. Auto Query Time Estimation can provide runtime data for up to 1000 queries, which is the MySQL HeatWave query history limit. To view the current MySQL HeatWave query history, query the `performance_schema.rpd_query_stats` table:

```
mysql> SELECT query_id, LEFT(query_text,160)
         FROM performance_schema.rpd_query_stats;
*************************** 1. row ***************************
           query_id: 14
LEFT(query_text,160): SELECT country from airport_geo
*************************** 2. row ***************************
           query_id: 15
LEFT(query_text,160): CREATE TABLE flight_from_US SELECT * FROM flight WHERE `from` IN (SELECT airport_id F
*************************** 3. row ***************************
           query_id: 16
LEFT(query_text,160): CREATE TABLE flight_from_US SELECT * FROM flight WHERE `from` IN (SELECT airport_id F
...
...
9 rows in set (0.2190 sec)
```

The following example shows how to retrieve runtime data for the entire query history using Auto Query Time Estimation. In this example, there are three queries in the query history: a successfully executed query, a query that failed due to an out of memory error, and a query that was cancelled using `Ctrl+C`. For an explanation of Auto Query Time Estimation data, see Auto Query Time Estimation Data.

```
mysql> CALL sys.heatwave_advisor(JSON_OBJECT('query_insights', true));
+------------------------------+
| INITIALIZING HEATWAVE ADVISOR |
+------------------------------+
| Version: 4.31                |
|                              |
| Output Mode: normal          |
| Excluded Queries: 0          |
| Target Schemas: All          |
|                              |
+------------------------------+
6 rows in set (0.2276 sec)

+-------------------------------------------------------+
| ANALYZING LOADED DATA                                 |
+-------------------------------------------------------+
| Total 14 tables loaded in HeatWave for 1 schemas      |
| Tables excluded by user: 0 (within target schemas)    |
```

```
|                                                          |
| SCHEMA                            TABLES        COLUMNS |
| NAME                              LOADED         LOADED |
| ------                            ------         ------ |
| `airportdb`                           14            106 |
|                                                          |
+----------------------------------------------------------+
8 rows in set (0.2276 sec)

+-------------------------------------------------------------------------------------------------------------------
| QUERY INSIGHTS
+-------------------------------------------------------------------------------------------------------------------
| Queries executed on Heatwave: 9
| Session IDs (as filter): None
|
| QUERY-ID  SESSION-ID  QUERY-STRING                                                          EXEC-RUNTIME (
| --------  ----------  ------------                                                          --------------
|       14        5326  SELECT country from airport_geo                                          0.454
|       15        5740  CREATE TABLE flight_from_US SELECT * FROM flight WHERE `from...      0.576
|       16        5748  CREATE TABLE flight_from_US SELECT * FROM flight WHERE `from...      0.556
|       17        5755  CREATE TABLE flight_from_US SELECT * FROM flight WHERE `from...      0.614
|       18        5916  CREATE TABLE flight_from_London SELECT * FROM flight WHERE `...      0.130
|       19        5326  SELECT COUNTRY from airport_geo                                       0.003
|       20        6396  CREATE TABLE flight_from_INDIA SELECT * FROM flight WHERE `f...      0.082
|       21        6501  CREATE TABLE flight_from_SPAIN SELECT * FROM flight WHERE `f...      0.041
|       22        6505  CREATE TABLE flight_from_NORWAY SELECT * FROM flight WHERE `...      0.045
|
| TOTAL ESTIMATED:   0   EXEC-RUNTIME:       0.000 sec
| TOTAL EXECUTED:    9   EXEC-RUNTIME:       2.501 sec
|
|
| Retrieve detailed query statistics using the query below:
|     SELECT log FROM sys.heatwave_autopilot_report WHERE stage = "QUERY_INSIGHTS" AND type = "info";
|
+-------------------------------------------------------------------------------------------------------------------
22 rows in set (0.2276 sec)
mysql> SELECT log FROM sys.heatwave_advisor_report
          WHERE stage = "QUERY_INSIGHTS"
          AND type = "info";
*************************** 1. row ***************************
log: {"comment": "", "query_id": 14, "query_text": "SELECT country from airport_geo", "session_id": 532
*************************** 2. row ***************************
log: {"comment": "", "query_id": 15, "query_text": "CREATE TABLE flight_from_US SELECT * FROM flight WH
*************************** 3. row ***************************
log: {"comment": "", "query_id": 16, "query_text": "CREATE TABLE flight_from_US SELECT * FROM flight WH
*************************** 4. row ***************************
log: {"comment": "", "query_id": 17, "query_text": "CREATE TABLE flight_from_US SELECT * FROM flight WH
*************************** 5. row ***************************
log: {"comment": "", "query_id": 18, "query_text": "CREATE TABLE flight_from_London SELECT * FROM fligh
...
...
9 rows in set (0.2169 sec)
```

Auto Query Time Estimation Data

Auto Query Time Estimation provides the following data:

- `QUERY-ID`

  The query ID.

- `SESSION-ID`

  The session ID that issued the query.

- `QUERY-STRING`

  The query string. `EXPLAIN`, if specified, is not displayed in the query string.

- `EXEC-RUNTIME`

The query execution runtime in seconds. Runtime estimates are differentiated from actual runtimes by the appearance of the following text adjacent to the runtime: `(est.)`. Actual runtimes are shown for successfully executed queries. Runtime estimates are shown for `EXPLAIN` queries, queries cancelled by `Ctrl+C`, and queries that fail with an out-of-memory error.

- `COMMENT`

  Comments associated with the query. Comments may include:

  - `Explain`: The query was run with `EXPLAIN`.
  - `Operation was interrupted by the user`: The query was successfully offloaded to MySQL HeatWave but was interrupted by a Ctrl+C key combination.
  - `Out of memory error during query execution in RAPID`: The query was successfully offloaded to MySQL HeatWave but failed due to an out-of-memory error.

- `TOTAL-ESTIMATED` and `EXEC-RUNTIME`

  The total number of queries with runtime estimates and total execution runtime (estimated).

- `TOTAL-EXECUTED` and `EXEC-RUNTIME`

  The total number of successfully executed queries and total execution runtime (actual).

- `Retrieve detailed statistics using the query below`

  The query retrieves detailed statistics from the `heatwave_autopilot_report` table. For an example of the detailed statistics, see Run Auto Query Time Estimation.

Auto Query Time Estimation data is available in machine readable format for use in scripts. Auto Query Time Estimation data is also available in `JSON` format or SQL table format with queries to the `heatwave_autopilot_report` table. See Section 5.8.8, “Autopilot Report Table”.

Auto Query Time Estimation Examples

- To view runtime data for all queries in the MySQL HeatWave query history for a particular schema:

```
mysql> CALL sys.heatwave_advisor(JSON_OBJECT('target_schema',JSON_ARRAY('tpch'),
          'query_insights',true));
```

- To view runtime data for queries issued by the current session:

```
mysql> CALL sys.heatwave_advisor(JSON_OBJECT('query_insights',true,
          'query_session_id', JSON_ARRAY(connection_id())));
```

- To view runtime data for queries issued by a particular session:

```
mysql> CALL sys.heatwave_advisor(JSON_OBJECT('query_insights', true,
          'query_session_id', JSON_ARRAY(8)));
```

- This example demonstrates how to invoke the Auto Query Time Estimation in silent output mode, which is useful if the output is consumed by a script, for example.

```
mysql> CALL sys.heatwave_advisor(JSON_OBJECT('query_insights',true,'output','silent'));
```

What's Next

Learn more about Unload Advisor.

5.8.7 Unload Advisor

Unload Advisor recommends tables to unload, reducing MySQL HeatWave memory usage. The recommendations are based upon when the tables were last queried.

Run Unload Advisor

For Advisor to recommend tables to unload, the tables must be loaded in MySQL HeatWave and a query history must be available.

To enable Unload Advisor, specify the `auto_unload` option in `recommend` mode. See Section 10.1.5, “HEATWAVE_ADVISOR”.

Use the `exclude_list` option to define a list of schemas and tables to exclude from Unload Advisor.

Use the `last_queried_hours` option to only recommend unloading tables that were not queried during this past number of hours. The default is `24` hours.

Set `memory_gain_ascending` to `true` to rank the unload table suggestions in ascending order based on the table size. The default is `false`.

Use the `limit_tables` option to limit the number of unload table suggestions, based on the order imposed by `memory_gain_ascending`. The default is `10`.

Unload Advisor Examples

- To view recommendations for tables to unload, for a particular schema:

```
mysql> CALL sys.heatwave_advisor(JSON_OBJECT('auto_unload',JSON_OBJECT('mode','recommend')));
+-----------------------------------------------------------------------------------------------------------------
| Using:
|  "last_queried_hours": 24
|  "memory_gain_ascending": FALSE
|  "limit_tables": 10
| Number of excluded tables: 0
|
| SCHEMA                   TABLE
| NAME                     NAME                              REASON
| ---------------          -------------------------         ----------------------------------------
| `airportdb`              `weatherdata`                     LAST QUERIED ON "2025-05-27 10:39:59"
| `airportdb`              `flight`                          LAST QUERIED ON "2025-05-28 07:17:00"
| `airportdb`              `airline`                         LAST QUERIED ON "2025-05-27 10:40:00"
| `airportdb`              `airplane`                        LAST QUERIED ON "2025-05-27 10:40:11"
| ...
| ...
+-----------------------------------------------------------------------------------------------------------------
19 rows in set (0.2171 sec)
```

What's Next

Learn more about Autopilot Report Table.

5.8.8 Autopilot Report Table

The `heatwave_autopilot_report` table is a temporary table. It contains data from the last execution of Advisor or Auto Parallel Load. Data is only available for the current session and is lost when the session terminates or when the server shuts down.

When Advisor or Auto Parallel Load run, they send detailed output to the `heatwave_autopilot_report` table in the `sys` schema. This includes execution logs and generated load scripts.

As of MySQL 8.0.32, MySQL HeatWave uses the `heatwave_autopilot_report` table. All versions prior to MySQL 8.0.32 use the `heatwave_advisor_report` and `heatwave_load_report` tables for reporting Autopilot Advisor and load activities. For more information, refer to Advisor Report Table.

Autopilot Report Table Query Examples

After the user initiates calls to Advisor or Auto Parallel Load, query the `heatwave_autopilot_report` table to review the generated reports and logs, as shown in the following examples:

- View warning information:

```
mysql> SELECT log FROM sys.heatwave_autopilot_report WHERE type="warn";
```

- View error information if Advisor or Auto Parallel Load stop unexpectedly:

```
mysql> SELECT log FROM sys.heatwave_autopilot_report WHERE type="error";
```

- View the generated DDL statements for Advisor recommendations, or to see the commands that would be executed by Auto Parallel Load in `normal` mode:

```
mysql> SELECT log->>"$.sql" AS "SQL Script"
          FROM sys.heatwave_autopilot_report
          WHERE type = "sql"
          ORDER BY id;
```

- Concatenate Advisor or Auto Parallel Load generated DDL statements into a single string to copy and paste for execution. The `group_concat_max_len` variable sets the result length in bytes for the `GROUP_CONCAT()` function to accommodate a potentially long string. The default `group_concat_max_len` setting is 1024 bytes.

```
mysql> SET SESSION group_concat_max_len = 1000000;
mysql> SELECT GROUP_CONCAT(log->>"$.sql" SEPARATOR ' ')
          FROM sys.heatwave_autopilot_report
          WHERE type = "sql"
          ORDER BY id;
```

- For Advisor, retrieve Auto Query Time Estimation data in `JSON` format:

```
mysql> SELECT log
          FROM sys.heatwave_autopilot_report
          WHERE stage = "QUERY_INSIGHTS" AND type = "info";
```

- For Advisor, retrieve Auto Query Time Estimation data in SQL table format:

```
mysql> SELECT log->>"$.query_id" AS query_id,
          log->>"$.session_id" AS session_id,
          log->>"$.query_text" AS query_text,
          log->>"$.runtime_estimated_ms" AS runtime_estimated_ms,
          log->>"$.runtime_executed_ms" AS runtime_executed_ms,
          log->>"$.comment" AS comment
          FROM sys.heatwave_autopilot_report
          WHERE stage = "QUERY_INSIGHTS" AND type = "info"
          ORDER BY id;
```

- For Auto Parallel Load, view the number of load commands generated:

```
mysql> SELECT Count(*) AS "Total Load Commands Generated"
          FROM sys.heatwave_autopilot_report
          WHERE type = "sql" ORDER BY id;
```

- For Auto Parallel Load, view load script data for a particular table:

```
mysql> SELECT log->>"$.sql"
          FROM sys.heatwave_autopilot_report
          WHERE type="sql" AND log->>"$.schema_name" = "db0" AND log->>"$.table_name" = "tbl"
          ORDER BY id;
```

What's Next

Learn more about Advisor Report Table.

5.8.9 Advisor Report Table

From MySQL version 8.0.32 onwards, the `heatwave_advisor_report` table, is deprecated and replaced with the `heatwave_autopilot_report` table in the `sys` schema. See Autopilot Report Table.

When MySQL runs Advisor, it sends detailed output to the `heatwave_advisor_report` table in the `sys` schema.

The `heatwave_advisor_report` table is a temporary table. It contains data from the last execution of Advisor. Data is only available for the current session and is lost when the session terminates or when the server is shut down.

Advisor Report Table Query Examples

Query the `heatwave_advisor_report` table after the user invokes the Advisor, as in the following examples:

- View Advisor warning information:

```
mysql> SELECT log FROM sys.heatwave_advisor_report WHERE type="warn";
```

- View error information if Advisor stops unexpectedly:

```
mysql> SELECT log FROM sys.heatwave_advisor_report WHERE type="error";
```

- View the generated DDL statements for Advisor recommendations:

```
mysql> SELECT log->>"$.sql" AS "SQL Script"
         FROM sys.heatwave_advisor_report
         WHERE type = "sql"
         ORDER BY id;
```

- Concatenate Advisor generated DDL statements into a single string to copy and paste for execution. The `group_concat_max_len` variable sets the result length in bytes for the `GROUP_CONCAT()` function to accommodate a potentially long string. The default `group_concat_max_len` setting is 1024 bytes.

```
mysql> SET SESSION group_concat_max_len = 1000000;
mysql> SELECT GROUP_CONCAT(log->>"$.sql" SEPARATOR ' ')
         FROM sys.heatwave_advisor_report
         WHERE type = "sql"
         ORDER BY id;
```

- Retrieve Auto Query Time Estimation data in `JSON` format:

```
mysql> SELECT log
         FROM sys.heatwave_advisor_report
         WHERE stage = "QUERY_INSIGHTS" AND type = "info";
```

- Retrieve Auto Query Time Estimation data in SQL table format:

```
mysql> SELECT log->>"$.query_id" AS query_id,
         log->>"$.session_id" AS session_id,
         log->>"$.query_text" AS query_text,
         log->>"$.runtime_estimated_ms" AS runtime_estimated_ms,
         log->>"$.runtime_executed_ms" AS runtime_executed_ms,
         log->>"$.comment" AS comment
         FROM sys.heatwave_advisor_report
         WHERE stage = "QUERY_INSIGHTS" AND type = "info"
         ORDER BY id;
```

What's Next

Learn how to Optimize Workloads for OLTP.

5.9 Optimize Workloads for OLTP

Online transaction processing, OLTP, does not require a MySQL HeatWave Cluster, nor the secondary engine. OLTP only requires the MySQL InnoDB primary engine.

5.9.1 Autopilot Indexing

Autopilot Indexing obtains workloads from the statement digest history stored in the Performance Schema, evaluates them, and identifies potential indexes that can help improve workload performance.

Indexes can improve the query performance of `SELECT` statements, but index management can increase the overhead of DML operations, `INSERT`, `UPDATE`, and `DELETE`. Autopilot Indexing can generate secondary index recommendations that optimize the overall workload performance.

As of MySQL 9.0.0, MySQL HeatWave Autopilot Advisor includes Autopilot Indexing that can make secondary index suggestions to improve workload performance.

Autopilot Indexing generates recommendations to add and drop indexes and also includes:

- An estimate of the overall workload performance benefit,
- An estimate of the overall storage footprint impact.
- A DDL statement, an explanation, and an estimated storage impact for each index recommendation.
- An estimate of the index creation time and the estimated performance gain of the top 5 queries of each create index recommendation.

To run Autopilot Indexing for OLTP workloads, the user must have the following MySQL privileges:

- The `PROCESS` privilege.
- The `EXECUTE` privilege on the `sys` schema.
- The `SELECT` privilege on the Performance Schema.

To view Autopilot Indexing command-line help:

```
mysql> CALL sys.autopilot_index_advisor(JSON_OBJECT('output','help'));
```

5.9.1.1 Autopilot Indexing Syntax

Autopilot Indexing uses the following syntax:

```
mysql> CALL sys.autopilot_index_advisor ([options]);

 options: {
     JSON_OBJECT('key','value'[,'key','value'] ...)
        'key','value':
        ['output',{'normal'|'silent'|'help'}]
        ['target_schema',JSON_ARRAY({'schema_name'['schema_name']}]
}
```

Autopilot Indexing *`options`* are specified as key-value pairs in `JSON` format. Options include:

- `output`: Defines how Autopilot Indexing produces output. Permitted values are:
  - `normal`: The default. Produces summarized output and sends it to `stdout` and to the `autopilot_index_advisor_report` table. See Section 5.9.1.4, “Autopilot Index Advisor Report Table”.
  - `silent`: Sends output to the `autopilot_index_advisor_report` table only. See Section 5.9.1.4, “Autopilot Index Advisor Report Table”. The `silent` output type is useful if human-readable output is not required; when the output is consumed by a script, for example.
  - `help`: Displays Autopilot Indexing command-line help.
- `target_schema`: Defines one or more schemas for Advisor to analyze. The list is specified as a `JSON` array. If a target schema is not specified, Advisor analyzes all user defined schemas. When

a target schema is specified, Advisor generates recommendations for tables belonging to the target schema. For the most accurate recommendations, specify one schema at a time. Only run Advisor on multiple schemas if the queries access tables in multiple schemas.

5.9.1.2 Run Autopilot Indexing

Autopilot Indexing provides recommendations for schemas with a representative workload that has at least five queries in the SQL statement digest history in the Performance Schema.

Autopilot Indexing only evaluates SQL statements in the SQL statement digest history that access existing tables. Autopilot Indexing does evaluate past SQL statements that access a table that has been dropped and recreated.

When the workload changes, invoke Autopilot Indexing again to update index recommendations.

Autopilot Indexing recommends indexes to create and drop. Rather than drop the suggested indexes, make them invisible first, and then drop them later after confirmation that there is no impact to any user queries. For instance, if a query uses an index hint, and that index is dropped, then the query fails to execute. This includes `FORCE INDEX`, `USE INDEX`, and `IGNORE INDEX`, see Index Hints.

Autopilot Indexing does not evaluate the following:

- System schemas.
- Small InnoDB tables, with `innodb_page_size` less than 16KB, or less than 1,000 rows.
- Statements executed from stored routines and prepared statements.
- Columns with the following data types:
  - `BINARY`
  - `VARBINARY`
  - `TINYBLOB`
  - `BLOB`
  - `MEDIUMBLOB`
  - `LONGBLOB`
  - `JSON`
  - `TEXT`
  - `TINYTEXT`
  - `MEDIUMTEXT`
  - `LONGTEXT`

Autopilot Indexing does not make recommendations for the following:

- The creation of primary keys.
- Indexes that enforce foreign key constraints.
- Functional indexes.
- Indexes on partitioned tables.
- Indexes that speed up `LIKE` predicates.

5.9.1.3 Autopilot Indexing Examples

To evaluate all user defined schemas:

```
mysql> CALL sys.autopilot_index_advisor(NULL);
```

To evaluate selected schemas:

```
mysql> CALL  sys.autopilot_index_advisor(JSON_OBJECT('target_schema',JSON_ARRAY('schema1','schema2', ...)))
```

An example of the output that includes the following information:

- New indexes to create.
- Existing indexes to drop.
- The reason, expected storage footprint, and performance impact of each suggestion.
- The overall expected performance benefit and expected change to the storage footprint.
- An estimate of the time to create an index.

```
+-----------------------------------------------------------------------------------------------------------------------
| INDEX SUGGESTIONS
+-----------------------------------------------------------------------------------------------------------------------
| Total Index suggestions: 5
| Statements analyzed:     519798
|
| SUGGEST           TABLE                                  INDEXED                                  REASO
| ACTION            NAME                                   COLUMNS                                  SUGGE
| -------           -----                                  -------                                  -----
| CREATE            `world`.`city`                         `Population`                             Missi
| CREATE            `world`.`state`                        `Area`, `Population`                     Missi
| CREATE            `world`.`country`                      `Area`                                   Missi
| CREATE            `world`.`continent`                    `ContinentCode`                          Missi
| DROP              `world`.`countrylanguage`              `CountryCode`                            Unuse
|
| Expected performance benefit after applying all Index suggestions: 97.3%
| Expected storage footprint after applying all Index suggestions:  + 236.98 MiB
|        64.00 KiB freed up by dropping indexes.
|       237.05 MiB required for creating indexes.
|       NOTE: Indexes are be stored efficiently at time of creation.
|             To accommodate efficient future inserts, size may double.
| Expected time for applying all Index creation suggestions:         25 s
|
+-----------------------------------------------------------------------------------------------------------------------
```

To clear the statement digest in the Performance Schema and start collecting statements for a new workload:

```
mysql> CALL sys.ps_truncate_all_tables(FALSE);
```

See Section 5.9.1.4, “Autopilot Index Advisor Report Table”.

5.9.1.4 Autopilot Index Advisor Report Table

When Autopilot Indexing runs, it sends detailed output to the `autopilot_index_advisor_report` table in the `sys` schema. This includes execution logs and generated load scripts.

The `autopilot_index_advisor_report` table is a temporary table that contains data from the last execution of Autopilot Indexing. Data is only available for the current session and is lost when the session terminates or when the server shuts down.

Index Advisor Report Table Query Examples

Query the `autopilot_index_advisor_report` table after calls to Autopilot Indexing, as in the following examples:

- To view the DDL statements for the index suggestions:

```
mysql> SELECT log->>"$.sql" AS "SQL Script"
        FROM sys.autopilot_index_advisor_report
        WHERE type = "sql"
        ORDER BY id;
+-----------------------------------------------------------------------------------------+
| SQL Script                                                                              |
+-----------------------------------------------------------------------------------------+
| CREATE INDEX `autoidx_tab1108_col2` ON `world`.`city` ( `Population` );                 |
| CREATE INDEX `autoidx_tab1110_col3_col2` ON `world`.`state` ( `Area`, `Population` ); |
| CREATE INDEX `autoidx_tab1104_col1` ON `world`.`country` ( `Area` );                    |
| CREATE INDEX `autoidx_tab1109_col5` ON `world`.`continent` ( `ContinentCode` );         |
| DROP INDEX `countrylanguage_idx4` ON `world`.`countrylanguage` ( `CountryCode` );       |
+-----------------------------------------------------------------------------------------+
5 rows in set (0.00 sec)
```

Then execute the script to implement the suggestions.

- To view explanations for Autopilot Indexing recommendations:

```
mysql> SELECT JSON_PRETTY(log) AS "Explanations"
        FROM sys.autopilot_index_advisor_report
        WHERE type = "explain"
        ORDER BY id;
+-----------------------------------------------------------------------------------------------------------------
| Explanations
+-----------------------------------------------------------------------------------------------------------------
| {
  "SQL": "CREATE INDEX `autoidx_tab1108_col2` ON `world`.`city` ( `Population` );",
  "explanation": [
    {
      "reason": "Covering Index",
      "query_text": "SELECT `Name` FROM `city` WHERE `Population` = ?",
      "estimated_gain": "700.0x"
    }
  ],
  "est_create_time": "5.79 s"
} |
+-----------------------------------------------------------------------------------------------------------------
1 row in set (0.01 sec)
```

The explanation includes the top 5 queries that benefit from the index, together with the reason and estimated gain of each query. It also includes the estimated time to create the index.

What's Next

Learn how to Monitor MySQL HeatWave.

5.10 MySQL HeatWave Monitoring

This section provides queries that you can use to monitor MySQL HeatWave.

5.10.1 Query Execution Monitoring

- To view the number of queries offloaded to the MySQL HeatWave Cluster for execution since the last time MySQL HeatWave Cluster was started:

```
mysql> SELECT VARIABLE_VALUE
          FROM performance_schema.global_status
          WHERE VARIABLE_NAME = 'rapid_query_offload_count';
+----------------+
| VARIABLE_VALUE |
+----------------+
| 62             |
+----------------+
```

- The Performance Schema statement event tables (see Performance Schema Statement Event Tables), and the `performance_schema.threads` and `performance_schema.processlist`

tables include an `EXECUTION_ENGINE` column that indicates whether a query was processed on the `PRIMARY` or `SECONDARY` engine, where the primary engine is InnoDB and the secondary engine is MySQL HeatWave. The `sys.processlist` and `sys.x$processlist` views in the MySQL `sys` Schema also include an `execution_engine` column.

This query shows the schema, the first 50 characters of the query, and the execution engine that processed the query:

```
mysql> SELECT CURRENT_SCHEMA, LEFT(DIGEST_TEXT, 50), EXECUTION_ENGINE
          FROM performance_schema.events_statements_history
          WHERE CURRENT_SCHEMA='tpch';
+---------------+----------------------------------------------------+------------------+
| CURRENT_SCHEMA | LEFT(DIGEST_TEXT, 50)                             | EXECUTION_ENGINE |
+---------------+----------------------------------------------------+------------------+
| tpch           | SELECT COUNT(*) FROM tpch.LINEITEM                | SECONDARY        |
+---------------+----------------------------------------------------+------------------+
```

- The Performance Schema statement summary tables (see Statement Summary Tables) include a `COUNT_SECONDARY` column that indicates the number of times a query was processed on the `SECONDARY` engine (MySQL HeatWave).

  This query retrieves the total number of secondary engine execution events from the `events_statements_summary_by_digest` table:

```
mysql> SELECT SUM(COUNT_SECONDARY)
          FROM performance_schema.events_statements_summary_by_digest;
+----------------------+
| SUM(COUNT_SECONDARY) |
+----------------------+
|                   25 |
+----------------------+
```

  This query counts all engine execution events for a particular schema and shows how many occurred on the primary engine (InnoDB) and how many occurred on the secondary engine (MySQL HeatWave):

```
mysql> SELECT SUM(COUNT_STAR) AS TOTAL_EXECUTIONS, SUM(COUNT_STAR) - SUM(COUNT_SECONDARY)
          AS PRIMARY_ENGINE, SUM(COUNT_SECONDARY) AS SECONDARY_ENGINE
          FROM performance_schema.events_statements_summary_by_digest
          WHERE SCHEMA_NAME LIKE 'tpch';
*************************** 1. row ***************************
TOTAL_EXECUTIONS: 25
  PRIMARY_ENGINE: 5
SECONDARY_ENGINE: 20
```

5.10.2 Query History and Statistics Monitoring

- To view the MySQL HeatWave query history including query start time, end time, and wait time in the scheduling queue, as discussed in Section 5.4.11, “Auto Scheduling”.

```
mysql> SELECT QUERY_ID,
          CONNECTION_ID,
          QUERY_START,
          QUERY_END,
          QUEUE_WAIT,
          SUBTIME(
              SUBTIME(QUERY_END, SEC_TO_TIME(RPD_EXEC / 1000)),
              SEC_TO_TIME(GET_RESULT / 1000)
          ) AS EXEC_START
      FROM (
              SELECT QUERY_ID,
                  STR_TO_DATE(
                      JSON_UNQUOTE(
                          JSON_EXTRACT(QEXEC_TEXT->>"$**.queryStartTime", '$[0]')
                      ),
                      '%Y-%m-%d %H:%i:%s.%f'
                  ) AS QUERY_START,
                  JSON_EXTRACT(QEXEC_TEXT->>"$**.timeBetweenMakePushedJoinAndRpdExec", '$[0]')
```

```
                AS QUEUE_WAIT,
                STR_TO_DATE(
                    JSON_UNQUOTE(
                        JSON_EXTRACT(QEXEC_TEXT->>"$**.queryEndTime", '$[0]')
                    ),
                    '%Y-%m-%d %H:%i:%s.%f'
                ) AS QUERY_END,
                JSON_EXTRACT(QEXEC_TEXT->>"$**.rpdExec.msec", '$[0]') AS RPD_EXEC,
                JSON_EXTRACT(QEXEC_TEXT->>"$**.getResults.msec", '$[0]') AS GET_RESULT,
                JSON_EXTRACT(QEXEC_TEXT->>"$**.thread", '$[0]') AS CONNECTION_ID
            FROM performance_schema.rpd_query_stats
        ) tmp;
```

The query returns the following data:

- `QUERY_ID`

  The ID assigned to the query by MySQL HeatWave. IDs are assigned in first in first out (FIFO) order.

- `CONNECTION_ID`

  The connection ID of the client that issued the query.

- `QUERY_START`

  The time the query was issued.

- `QUERY_END`

  The time the query finished executing.

- `QUEUE_WAIT`

  The amount of time the query waited in the scheduling queue.

- `EXEC_START`

  The time that MySQL HeatWave started executing the query.

- To view the number of records in the `rpd_query_stats` table. The `rpd_query_stats` table stores query compilation and execution statistics (the query history) for the last 1000 successfully executed queries.

```
mysql> SELECT COUNT(*) FROM performance_schema.rpd_query_stats;
+----------+
| count(*) |
+----------+
|     1000 |
+----------+
```

- To view query IDs for the first and last successfully executed queries:

```
mysql> SELECT MIN(QUERY_ID), MAX(QUERY_ID) FROM performance_schema.rpd_query_stats;
+---------------+---------------+
| MIN(QUERY_ID) | MAX(QUERY_ID) |
+---------------+---------------+
|             2 |          1001 |
+---------------+---------------+
```

- To view the query count for a table and the last time the table was queried:

```
mysql> USE performance_schema;
mysql> SELECT rpd_table_id.TABLE_NAME, rpd_tables.NROWS, rpd_tables.QUERY_COUNT,
          rpd_tables.LAST_QUERIED FROM rpd_table_id, rpd_tables
          WHERE rpd_table_id.ID = rpd_tables.ID;
+-----------+---------+-------------+---------------------------+
```

```
| TABLE_NAME | NROWS   | QUERY_COUNT | LAST_QUERIED               |
+------------+---------+-------------+----------------------------+
| orders     | 1500000 |           1 | 2021-12-06 14:32:59.868141 |
+------------+---------+-------------+----------------------------+
```

5.10.3 Scanned Data Monitoring

MySQL HeatWave tracks the amount of base relation data scanned by all queries collectively and by individual queries. Base relation data depends on the table schema, number of rows in each table, and DML operations on a MySQL table. Features like Auto Zonemap and MySQL table partitions also impact the base relation data scanned by a query. When one of these features is present, the actual bytes scanned are typically lower than a full base relation scan. The data scanned is calculated in megabytes (MB) based on the in-memory allocation.

To view a cumulative total of data scanned (in MB) by all successfully executed MySQL HeatWave queries from the time the MySQL HeatWave Cluster was last started, query the `hw_data_scanned` global status variable. For example:

```
mysql> SHOW GLOBAL STATUS LIKE 'hw_data_scanned';
+-----------------+-------+
| Variable_name   | Value |
+-----------------+-------+
| hw_data_scanned | 66    |
+-----------------+-------+
```

The cumulative total does not include data scanned by failed queries, queries that were not offloaded to MySQL HeatWave, or `EXPLAIN` queries.

The `hw_data_scanned` value is reset to 0 only when the MySQL HeatWave Cluster is restarted.

If a subset of MySQL HeatWave nodes go offline, MySQL HeatWave retains the cumulative total of scanned data as long as the MySQL HeatWave Cluster remains in an active state. When the MySQL HeatWave Cluster becomes fully operational and starts processing queries again, MySQL HeatWave resumes tracking the amount of data scanned, adding to the cumulative total.

```
mysql> SELECT query_id,
          JSON_EXTRACT(JSON_UNQUOTE(qexec_text->'$**.error'),'$[0]') AS error_message
          FROM performance_schema.rpd_query_stats;
+----------+------------+--------------+---------------+
| query_id | session_id | data_scanned | error_message |
+----------+------------+--------------+---------------+
|        1 | 8          | 66           | ""            |
+----------+------------+--------------+---------------+
```

The example above retrieves any error message associated with the query ID. If a query fails or was interrupted, the number of bytes scanned by the failed or interrupted query and the associated error message are returned, as shown in the following examples:

```
mysql> SELECT query_id,
          JSON_EXTRACT(JSON_UNQUOTE(qexec_text->'$**.error'),'$[0]') AS error_message
          FROM performance_schema.rpd_query_stats;
+----------+------------+--------------+----------------------------------------+
| query_id | session_id | data_scanned | error_message                          |
+----------+------------+--------------+----------------------------------------+
|        1 | 8          | 461          | "Operation was interrupted by the user." |
+----------+------------+--------------+----------------------------------------+
```

```
mysql> SELECT query_id,
          JSON_EXTRACT(JSON_UNQUOTE(qexec_text->'$**.error'),'$[0]') AS error_message
          FROM performance_schema.rpd_query_stats;
+----------+------------+--------------+-----------------------------------------------+
| query_id | session_id | data_scanned | error_message                                 |
+----------+------------+--------------+-----------------------------------------------+
|        1 | 8          | 987          | "Out of memory error during query execution in|
|          |            |              |  RAPID."                                      |
+----------+------------+--------------+-----------------------------------------------+
```

Chapter 6 Train and Use Machine Learning Models

Table of Contents

6.1 About MySQL HeatWave AutoML 289
6.1.1 MySQL HeatWave AutoML Ease of Use 290
6.1.2 MySQL HeatWave AutoML Workflow 290
6.1.3 MySQL HeatWave AutoML Learning Types 291
6.1.4 Oracle AutoML 292
6.2 Additional MySQL HeatWave AutoML Requirements 293
6.3 MySQL HeatWave AutoML Privileges 294
6.4 Supported Data Types for MySQL HeatWave AutoML 295
6.5 Create a Machine Learning Model 296
6.5.1 Prepare Data 296
6.5.2 Train a Model 300
6.5.3 Load a Model 301
6.5.4 Generate Predictions 303
6.5.5 Generate Model Explanations 309
6.5.6 Generate Prediction Explanations 312
6.5.7 Score a Model 319
6.6 Learn About MySQL HeatWave AutoML with Oracle Cloud Infrastructure Generative AI 321
6.7 Machine Learning Use Cases 329
6.7.1 Classify Data 329
6.7.2 Perform Regression Analysis 337
6.7.3 Generate Forecasts 344
6.7.4 Detect Anomalies 353
6.7.5 Generate Recommendations 370
6.7.6 Topic Modeling 392
6.7.7 Train a Model in an Interim Isolated Environment 397
6.8 Use MySQL HeatWave AutoML with Lakehouse 402
6.9 Manage Machine Learning Models 408
6.9.1 The Model Catalog 408
6.9.2 Work with Model Handles 415
6.9.3 Unload a Model 418
6.9.4 View Model Details 420
6.9.5 Delete a Model 422
6.9.6 Share a Model 423
6.9.7 Manage External ONNX Models 426
6.9.8 Analyze Data Drift 440
6.10 Track Progress for MySQL HeatWave AutoML Routines 446
6.11 Monitor the Status of MySQL HeatWave AutoML 450

This chapter describes how to create and manage machine learning models with MySQL HeatWave AutoML.

6.1 About MySQL HeatWave AutoML

MySQL HeatWave AutoML makes it easy to use machine learning (ML), whether you are a novice user or an experienced ML practitioner. You provide the data, and MySQL HeatWave AutoML analyzes the characteristics of the data and creates an optimized machine learning model that you can use to generate predictions and explanations. An ML model makes predictions by identifying patterns in your data and applying those patterns to unseen data. MySQL HeatWave AutoML explanations help you understand how these predictions are made, such as which features of a dataset contribute most to a prediction. You can score machine learning models to get a better understanding of the quality of the model and its ability to generate reliable predictions.

With MySQL HeatWave AutoML, you can do the following:

- Classify Data
- Perform Regression Analysis
- Generate Forecasts
- Detect Anomalies
- Generate Recommendations
- Topic Modeling

6.1.1 MySQL HeatWave AutoML Ease of Use

MySQL HeatWave AutoML is purpose-built for ease of use. It requires no machine learning expertise, specialized tools, or algorithms. With MySQL HeatWave AutoML and a set of training data in a MySQL HeatWave DB system, you can train a predictive machine learning model with a single call to the `ML_TRAIN` SQL routine.

For example:

```
CALL sys.ML_TRAIN('heatwaveml_bench.census_train', 'revenue', NULL, @census_model);
```

The `ML_TRAIN` routine leverages Oracle AutoML technology to automate training of machine learning models. Learn more about Oracle AutoML.

You can use a model created by `ML_TRAIN` with other MySQL HeatWave AutoML routines to generate predictions and explanations. For example, the following call to the `ML_PREDICT_TABLE` routine generates predictions for a table of input data:

```
CALL sys.ML_PREDICT_TABLE('heatwaveml_bench.census_test', @census_model,
'heatwaveml_bench.census_predictions', NULL);
```

All MySQL HeatWave AutoML operations are initiated by running `CALL` or `SELECT` statements, which can be easily integrated into your applications. MySQL HeatWave AutoML routines reside in the MySQL `sys` schema and can be run from any MySQL client or application that is connected to a DB System with a MySQL HeatWave Cluster. Learn more about MySQL HeatWave AutoML Routines.

In addition, with MySQL HeatWave AutoML, there is no need to move or reformat your data. Data and machine learning models never leave the MySQL HeatWave Service, which saves you time and effort while keeping your data and models secure.

To start using MySQL HeatWave AutoML with sample datasets, see Machine Learning Use Cases.

What's Next

- Learn more about the following:
  - MySQL HeatWave AutoML Learning Types
  - MySQL HeatWave AutoML Workflow
  - Oracle AutoML
- Learn how to Create a Machine Learning Model.

6.1.2 MySQL HeatWave AutoML Workflow

A typical MySQL HeatWave AutoML workflow is described below:

1. When you run the `ML_TRAIN` routine, MySQL HeatWave AutoML retrieves the data to use for training. The data can originate from either DB System tables or external Lakehouse tables. The training data is then distributed across the MySQL HeatWave Cluster, which performs machine learning computation in parallel. See Train a Model.

2. MySQL HeatWave AutoML analyzes the training data, trains an optimized machine learning model, and stores the model in a model catalog on the DB System. See Model Catalog.
3. MySQL HeatWave AutoML `ML_PREDICT_*` and `ML_EXPLAIN_*` routines use the trained model to generate predictions and explanations on test or unseen data. See Generate Predictions and Generate Explanations.
4. Predictions and explanations are returned to the DB System and to the user or application that issued the query.

Optionally, the `ML_SCORE` routine can be used to compute the quality of a model to ensure that predictions and explanations are reliable. See Score a Model.

To start using MySQL HeatWave AutoML with sample datasets, see Machine Learning Use Cases.

What's Next

- Learn more about the following:
  - MySQL HeatWave AutoML Learning Types
  - MySQL HeatWave AutoML Ease of Use
  - Oracle AutoML
- Learn how to Create a Machine Learning Model.

6.1.3 MySQL HeatWave AutoML Learning Types

MySQL HeatWave AutoML supports the following types of machine learning: supervised, unsupervised, and semi-supervised.

Supervised Learning

Supervised learning creates a machine learning model by analyzing a labeled dataset to learn patterns. This means that the dataset has values associated with the column (the label) that the machine learning model eventually generates predictions for. The model is able to to predict labels based on the features of the dataset. For example, a census and income dataset may have features such as age, education, occupation, and country that you can use to predict the income of an individual (the label). The income label in this dataset already has values that the machine learning model uses for training.

Once a machine learning model is trained, it can be used on unseen data, where the label is unknown, to make predictions. In a business setting, predictive models have a variety of possible applications such as predicting customer churn, approving or rejecting credit applications, predicting customer wait times, and so on.

See Labeled Data and Unlabeled Data to learn more.

Unsupervised Learning

MySQL HeatWave AutoML supports unsupervised learning for forecasting, anomaly detection and topic modeling use cases. This type of learning requires no labeled data. This means that the column (the label) the machine learning model eventually generates predictions for has no values in the dataset for training. For example, a dataset of credit card transactions that you use for anomaly detection has a column indicating if the transaction is anomalous or normal, but the column has no data (unlabeled). See Generate Forecasts, Detect Anomalies, and Topic Modeling to learn more.

Semi-Supervised Learning

MySQL 9.0.1-u1 introduces support for semi-supervised learning when running anomaly detection. This type of machine learning algorithm uses a specific set of labeled data along with unlabeled data to detect anomalies. The dataset for this type of model must have a column whose only allowed

values are 0 (normal), 1, (anomalous), and NULL (unlabeled). All rows in the dataset are used to train the unsupervised component, while the rows with a value different than NULL are used to train the supervised component. See Detect Anomalies and Anomaly Detection Model Types to learn more.

What's Next

- Learn more about the following:
  - MySQL HeatWave AutoML Ease of Use
  - MySQL HeatWave AutoML Workflow
  - Oracle AutoML
- Learn how to Create a Machine Learning Model.
- Review Machine Learning Use Cases to create machine learning models with sample datasets.

6.1.4 Oracle AutoML

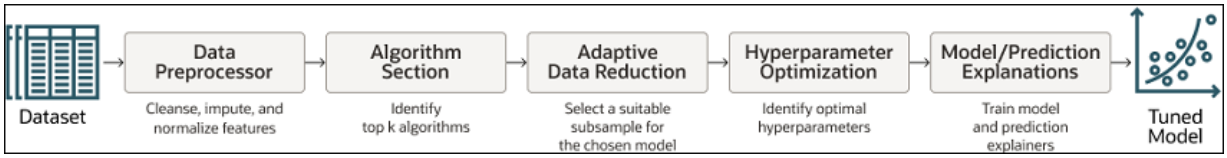
The MySQL HeatWave AutoML `ML_TRAIN` routine leverages Oracle AutoML technology to automate the process of training a machine learning model. Oracle AutoML replaces the laborious and time consuming tasks of the data analyst, whose workflow is as follows:

1. Selecting a model from a large number of viable candidate models.
2. For each model, tuning hyperparameters.
3. Selecting only predictive features to speed up the pipeline and reduce over-fitting.
4. Ensuring the model performs well on unseen data (also called generalization).

Oracle AutoML automates this workflow, providing you with an optimal model given a time budget. When you run the MySQL HeatWave AutoML `ML_TRAIN` routine, that triggers the Oracle AutoML pipeline to run the following stages in a single command:

- Data pre-processing
- Algorithm selection
- Adaptive data reduction
- Hyperparameter optimization
- Model and prediction explanations

Figure 6.1 Oracle AutoML Pipeline



Oracle AutoML also produces high quality models very efficiently, which is achieved through a scalable design and intelligent choices that reduce trials at each stage in the pipeline.

- *Scalable design*: The Oracle AutoML pipeline is able to exploit both MySQL HeatWave internode and intranode parallelism, which improves scalability and reduces runtime.
- *Intelligent choices reduce trials in each stage*: Algorithms and parameters are chosen based on dataset characteristics, which ensures that the model is accurate and efficiently selected. This is achieved using meta-learning throughout the pipeline.

For additional information about Oracle AutoML, refer to Yakovlev, Anatoly, et al. "Oracle AutoML: A Fast and Predictive AutoML Pipeline." Proceedings of the VLDB Endowment 13.12 (2020): 3166-3180.

What's Next

- Learn more about the following:
  - MySQL HeatWave AutoML Learning Types
  - MySQL HeatWave Ease of Use
  - MySQL HeatWave AutoML Workflow
- Learn how to Create a Machine Learning Model.
- Review Machine Learning Use Cases to create machine learning models with sample datasets.

6.2 Additional MySQL HeatWave AutoML Requirements

Before You Begin

- Review Requirements for using MySQL HeatWave.

Note

MySQL HeatWave AutoML is supported on DB Systems with 32 or less MySQL HeatWave Cluster nodes only.

Model and Table Sizes

- The table used to train a model cannot exceed 10 GB, 100 million rows, or 1017 columns.
- Refer to the appropriate MySQL version for maximum MySQL HeatWave AutoML model sizes.
  - Before MySQL 9.0.0: The maximum model size is 900MB.
  - MySQL 9.0.0 and later: The shape you set for the MySQL HeatWave cluster in the DB system defines the total memory available to train a model and for all loaded models. For imported models, we recommend individual models have a size of 4GB or less. To query all loaded models and relevant model sizes, see ML_MODEL_ACTIVE.

    Refer to the following to learn more:
    - For MySQL HeatWave on OCI, see Supported Shapes.
    - For MySQL HeatWave on AWS, see Supported Shapes.
    - For MySQL HeatWave on Azure, see Provisioning Oracle MySQL HeatWave.

Data Requirements

- Each dataset must reside in a single table on the DB System. MySQL HeatWave AutoML routines operate on a single table. See Load and Manage Data in MySQL HeatWave.
- Table columns must use supported data types. See Supported Data Types for MySQL HeatWave AutoML to learn more.
- NaN (Not a Number) values are not recognized by MySQL and should be replaced by `NULL`.
- Refer to the following requirements for specific machine learning models.

- Classification models: Must have at least two distinct values, and each distinct value should appear in at least five rows.
- Regression models: The target column must be numeric.

Note

The `ML_TRAIN` routine ignores columns missing more than 20% of its values and columns with the same value in each row. Missing values in numerical columns are replaced with the average value of the column, standardized to a mean of 0 and with a standard deviation of 1. Missing values in categorical columns are replaced with the most frequent value, and either one-hot or ordinal encoding is used to convert categorical values to numeric values. The input data as it exists in the MySQL database is not modified by `ML_TRAIN`.

MySQL User Names

To use MySQL HeatWave AutoML, ensure that the MySQL user name that trains a model does not have a period character ("."). For example, a user named `'joesmith'@'%'` is permitted to train a model, but a user named `'joe.smith'@'%'` is not. The model catalog schema created by the `ML_TRAIN` procedure incorporates the user name in the schema name (for example, `ML_SCHEMA_joesmith`), and a period is not a permitted schema name character.

What's Next

- Learn more about the following:
  - MySQL HeatWave AutoML Privileges
  - Supported Data Types for MySQL HeatWave AutoML
- Learn how to Create a Machine Learning Model.
- Review Machine Learning Use Cases to create machine learning models with sample datasets.

6.3 MySQL HeatWave AutoML Privileges

To use MySQL HeatWave AutoML, ask the admin user to grant you the following privileges. Replace *`user_name`* and *`database_name`* in the commands with the appropriate user name and database name.

Database Privileges

You need the following privileges to access the database that stores the input tables (training datasets).

```
mysql> GRANT SELECT, ALTER ON database_name.* TO 'user_name'@'%';
```

You need the following privileges to access the database that stores the output tables of generated predictions and explanations.

```
mysql> GRANT CREATE, DROP, INSERT, SELECT, ALTER, DELETE, UPDATE ON database_name.* TO 'user_name'@'%';
```

Lakehouse and Tracking/Monitoring Privileges

You need the following privileges to access external Lakehouse tables that have machine learning datasets, and to track/monitor the status of AutoML and AutoML routines..

```
mysql> GRANT SELECT ON performance_schema.rpd_tables TO 'user_name'@'%';
mysql> GRANT SELECT ON performance_schema.rpd_table_id TO 'user_name'@'%';
mysql> GRANT SELECT ON performance_schema.rpd_query_stats TO 'user_name'@'%';
mysql> GRANT SELECT ON performance_schema.rpd_ml_stats TO 'user_name'@'%';
```

Model Catalog Privileges

You need the following privileges to access machine learning models from the model catalog.

```
mysql> GRANT SELECT, INSERT, CREATE, ALTER, UPDATE, DELETE, DROP, GRANT OPTION ON ML_SCHEMA_user_name.*
```

System Privileges

You need the following privileges for the system database where MySQL HeatWave routines reside.

```
mysql> GRANT SELECT, EXECUTE ON sys.* TO 'user_name'@'%';
```

What's Next

- Learn more about the following:
  - Additional MySQL HeatWave AutoML Requirements
  - Supported Data Types for MySQL HeatWave AutoML
- Learn how to Create a Machine Learning Model.
- Review Machine Learning Use Cases to create machine learning models with sample datasets.

6.4 Supported Data Types for MySQL HeatWave AutoML

MySQL HeatWave AutoML supports the following data types.

Numeric Data Types

- `DECIMAL`
- `DOUBLE`
- `FLOAT`
- `INT`
- `INT UNSIGNED`
- `TINYINT`
- `TINYINT UNSIGNED`
- `SMALLINT`
- `SMALLINT UNSIGNED`
- `MEDIUMINT`
- `MEDIUMINT UNSIGNED`
- `BIGINT`
- `BIGINT UNSIGNED`

Temporal Data Types

- `DATE`
- `TIME`
- `DATETIME`
- `TIMESTAMP`

- `YEAR`

String and Text Data Types

- `VARCHAR`
- `CHAR`
- `TINYTEXT`
- `TEXT`
- `MEDIUMTEXT`
- `LONGTEXT`

Data Type Limitations

MySQL HeatWave AutoML uses TfidfVectorizer to pre-process `TINYTEXT`, `TEXT`, `MEDIUMTEXT`, and `LONGTEXT`, and appends the results to the data set. MySQL HeatWave AutoML has the following limitations for text usage:

- The `ML_PREDICT_TABLE` `ml_results` column contains the prediction results and the data. This combination must be fewer than 65,532 characters.
- MySQL HeatWave AutoML only supports datasets in the English language.
- MySQL HeatWave AutoML does not support text columns with `NULL` values.
- MySQL HeatWave AutoML does not support a text target column.
- MySQL HeatWave AutoML does not support recommendation tasks with a text column.
- For the forecasting task, `endogenous_variables` cannot be text.

What's Next

- Learn more about the following:
  - Additional MySQL HeatWave AutoML Requirements
  - MySQL HeatWave AutoML Privileges
- Learn how to Create a Machine Learning Model.
- Review Machine Learning Use Cases to create machine learning models with sample datasets.

6.5 Create a Machine Learning Model

The topics in this section go through the process of training and using a machine learning model.

Before going through these tasks, make sure to Review Requirements for MySQL HeatWave and Additional MySQL HeatWave AutoML Requirements.

To start using MySQL HeatWave AutoML with sample datasets, see Machine Learning Use Cases.

6.5.1 Prepare Data

Review the following topics to learn more about preparing data for machine learning models.

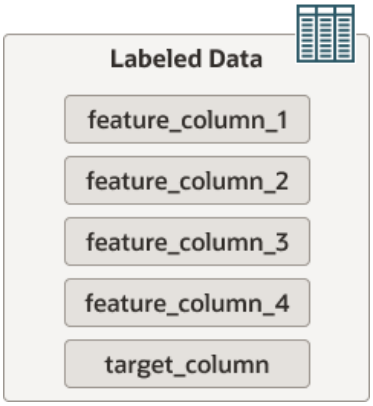
6.5.1.1 Overview of Preparing Data

MySQL HeatWave AutoML works with labeled and unlabeled data to train and score machine learning models.

Labeled Data

Labeled data is data that has values associated with it. It has feature columns and a target column (the *label*), as illustrated in the following diagram:

Figure 6.2 Labeled Data



Feature columns contain the input variables used to train the machine learning model. The target column contains *ground truth values* or, in other words, the correct answers. This dataset can be considered the *training dataset*.

A labeled dataset with ground truth values is also used to score a model (compute its accuracy and reliability). This dataset should have the same columns as the *training dataset* but with a different set of data. This dataset can be considered the *validation dataset*.

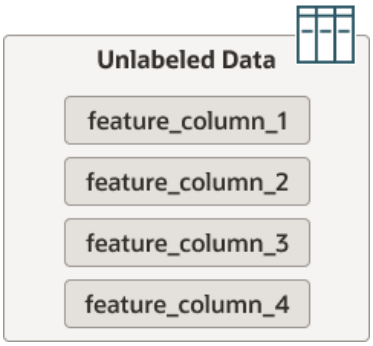
Labeled Data Example

A table of data for bank customers can be a labeled dataset. The feature columns in the table have data related to job, marital status, education, and city of residence. The target column has the approval status of a loan application, `Yes` or `No`. You can use some of the data in this table to train a classification machine learning model. You can also use the data in the table that wasn't used for training to score the trained machine learning model.

Unlabeled Data

Unlabeled data has feature columns but no target column (no answers), as illustrated below:

Figure 6.3 Unlabeled Data



If you are training a machine learning model that does not require labeled data, such as models for topic modeling or anomaly detection, you use unlabeled data. MySQL HeatWave AutoML also uses unlabeled data to generate predictions and explanations. It must have exactly the same feature columns as the training dataset but no target column. This type of dataset can be considered the *test dataset*. Test data starts as labeled data, but the label is not considered when the machine learning model generates predictions and explanations. This allows you to compare the generated predictions and explanations with the real values in the dataset before you start using “unseen data”.

The “unseen data” that you eventually use with your model to make predictions is also unlabeled data. Like the *test dataset*, unseen data must have exactly the same feature columns as the training dataset but no target column.

Unlabeled Data Example

A table of data for credit card transactions can be an unlabeled dataset. The feature columns in the table have data related to the amount of the purchase and the location of the purchase. Because there is no column identifying any transactions as anomalous or normal, it is unlabeled data. MySQL HeatWave AutoML can train an anomaly detection model on the unlabeled data to try and find unusual patterns in the data. A different set of labeled data identifying anomalies in credit cards transactions can be used to score the trained model.

Example Datasets

To start using AutoML with sample datasets, see Machine Learning Use Cases. Alternatively, navigate to the *MySQL HeatWave AutoML examples and performance benchmarks* GitHub repository at https://github.com/oracle-samples/heatwave-ml.

What's Next

- Learn how to Prepare Training and Testing Datasets.
- Learn how to Train a Model.

6.5.1.2 Prepare Training and Testing Datasets

As of MySQL 9.4.1, you can automatically create training and testing datasets with the `TRAIN_TEST_SPLIT` routine.

Before You Begin

- Review the Requirements.
- Get the Required Privileges to use MySQL HeatWave AutoML.
- Review the Data Types Supported For Machine Learning Tasks.
- Load structured data from the DB system or External Sources.

Overview

The `TRAIN_TEST_SPLIT` routine takes your datasets and prepares new tables for training and testing machine learning models. Two new tables in the same database are created with the following names:

- `[original_table_name]_train`
- `[original_table_name]_test`

The split of the data between training and testing datasets depends on the machine learning task.

- Classification: A stratified split of data. For each class in the dataset, 80% of the samples go into the training dataset, and the remaining go into the testing dataset. If the number of samples in the 80% subset is fewer than five, then five samples are inserted into the training dataset.
- Regression: A random split of data.
- Forecasting: A time-based split of data. The data is inserted in order according to `datetime_index` values. The first 80% of the samples go into the training dataset. The remaining samples go into the testing dataset.
- Unsupervised anomaly detection: A random split of data. 80% of the samples go into the training dataset, and the remaining samples go into the testing dataset.

- Semi-supervised anomaly detection: A stratified split of data.
- Anomaly detection for log data: A split of data based on primary key values. The first 80% of the samples go into the training dataset. The remaining samples go into the testing dataset. Review requirements when running Anomaly Detection for Logs.
- Recommendations: A random split of data.
- Topic modeling: A random split of data.

Parameters to Prepare Training and Testing Datasets

To run the `TRAIN_TEST_SPLIT` routine, you use the following parameters:

- `table_name`: You must provide the fully qualified name of the table that contains the dataset to split (*`schema_name.table_name`*).
- `target_column_name`: Classification and semi-supervised anomaly detection tasks require a target column. All other tasks do not require a target column. If a target column is not required, you can set this parameter to `NULL`.
- `options`: Set the following options as needed as key-value pairs in JSON object format. If no options are needed, set this to `NULL`.
  - `task`: Set the appropriate machine learning task: `classification`, `regression`, `forecasting`, `anomaly_detection`, `log_anomaly_detection`, `recommendation`, or `topic_modeling`. If the machine learning task is not set, the default task is `classification`.
  - `datetime_index`: Required for forecasting tasks. The column that has datetime values.

    The following data types for this column are supported:

    - `DATETIME`
    - `TIMESTAMP`
    - `DATE`
    - `TIME`
    - `YEAR`
  - `semisupervised`: If running an anomaly detection task, set this to `true` for semi-supervised learning, or `false` for unsupervised learning. If this is set to `NULL`, then the default value of `false` is selected.

TRAIN_TEST_SPLIT Example

To automatically generate a training and testing dataset:

1. Run the `TRAIN_TEST_SPLIT` routine.

   ```
   mysql> CALL sys.TRAIN_TEST_SPLIT('table_name', 'target_column_name', options);
   ```

   Replace *`table_name`*, *`target_column_name`*, and *`options`* with your own values. For example:

   ```
   mysql> CALL sys.TRAIN_TEST_SPLIT('data_files_db.data_files_1', 'class', JSON_OBJECT('task', 'classif
   ```

2. Confirm the two datasets are created ([original_table_name]_train and [original_table_name]_test) by querying the tables in the database.

   ```
   mysql> SHOW TABLES;
   +------------------------+
   | Tables_in_data_files_db |
   ```

```
+-------------------------+
| data_files_1            |
| data_files_1_test       |
| data_files_1_train      |
+-------------------------+
```

What's Next

- Learn how to Train a Model.

6.5.2 Train a Model

Run the `ML_TRAIN` routine on a training dataset to produce a trained machine learning model.

Before You Begin

- Review how to Prepare Data.
- Review Additional MySQL HeatWave AutoML Requirements.

ML_TRAIN Overview

`ML_TRAIN` supports training of the following models:

- Classification: Assign items to defined categories.
- Regression: Generate a prediction based on the relationship between a dependent variable and one or more independent variables.
- Forecasting: Use a timeseries dataset to generate forecasting predictions.
- Anomaly Detection: Detect unusual patterns in data.
- Recommendation: Generate user and product recommendations.
- Topic Modeling: Generate words and similar expressions that best characterize a set of documents (As of MySQL 9.0.1-u1).

The training dataset used with `ML_TRAIN` must reside in a table on the DB System.

`ML_TRAIN` stores machine learning models in the `MODEL_CATALOG` table. See The Model Catalog to learn more.

The time required to train a model can take a few minutes to a few hours depending on the following:

- The number of rows and columns in the dataset. MySQL HeatWave AutoML supports tables up to 10 GB in size with a maximum of 100 million rows and or 1017 columns.
- The specified `ML_TRAIN` parameters.
- The size of the MySQL HeatWave Cluster.

To learn more about `ML_TRAIN` requirements and options, see ML_TRAIN or Machine Learning Use Cases.

The quality and reliability of a trained model can be assessed using the `ML_SCORE` routine. For more information, see Score a Model. `ML_TRAIN` displays the following message if a trained model has a low score: `Model Has a low training score, expect low quality model explanations`.

ML_TRAIN Example

Before training a model, it is good practice to define your own model handle instead of automatically generating one. This allows you to easily remember the model handle for future routines on the trained model instead of having to query it, or depending on the session variable that can no longer be used when the current connection terminates. See Define Model Handle to learn more.

To train a machine learning model:

1. Optionally, set the value of the session variable, which sets the model handle to this same value.

```
mysql> SET @variable = 'model_handle';
```

Replace *`@variable`* and *`model_handle`* with your own definitions. For example:

```
mysql> SET @census_model = 'census_test';
```

The model handle is set to `census_test`.

2. Run the `ML_TRAIN` routine.

```
mysql> CALL sys.ML_TRAIN('table_name', 'target_column_name', JSON_OBJECT('task', 'task_name'), @vari
```

Replace *`table_name`*, *`target_column_name`*, *`task_name`*, and *`variable`* with your own values.

The following example runs `ML_TRAIN` on the `census_data.census_train` training dataset.

```
mysql> CALL sys.ML_TRAIN('census_data.census_train', 'revenue', JSON_OBJECT('task', 'classification'
```

Where:

- `census_data.census_train` is the fully qualified name of the table that contains the training dataset (*`schema_name.table_name`*).
- `revenue` is the name of the target column, which contains ground truth values.
- `JSON_OBJECT('task', 'classification')` specifies the machine learning task type.
- `@census_model` is the session variable previously set that defines the model handle to the name defined by the user: `census_test`. If you do not define the model handle before training the model, the model handle is automatically generated, and the session variable only stores the model handle for the duration of the connection. User variables are written as *`@var_name`*. Any valid name for a user-defined variable is permitted. See Work with Model Handles to learn more.

3. When the training completes, query the model catalog for the model handle and the name of the trained table to confirm the model handle is correctly set. Replace `user1` with your own user name.

```
mysql> SELECT model_handle, train_table_name FROM ML_SCHEMA_user1.MODEL_CATALOG;
+-------------------------------------------------------+---------------------------------+
| model_handle                                          | train_table_name                |
+-------------------------------------------------------+---------------------------------+
| census_test                                           | census_data.census_train        |
+-------------------------------------------------------+---------------------------------+
1 row in set (0.0450 sec)
```

Tip

When done working with a trained model, it is good practice to unload it. See Unload a Model.

What's Next

- For details on all training options and to view more examples for task-specific models, see ML_TRAIN.
- Learn how to Load a Model.

6.5.3 Load a Model

You must load a machine learning model from the model catalog into MySQL HeatWave before running MySQL HeatWave AutoML routines other than `ML_TRAIN`. A model remains loaded and

can be called repetitively by MySQL HeatWave AutoML routines until it is unloaded using the `ML_MODEL_UNLOAD` routine, or until the MySQL HeatWave Cluster is restarted.

A model can only be loaded by the MySQL user that created the model unless you grant access to other users. For more information, see Grant Other Users Access to a Model.

MySQL 9.0.0 introduces support for large models that changes how MySQL HeatWave AutoML stores models. See The Model Object Catalog Table. `ML_MODEL_LOAD` upgrades older models.

For `ML_MODEL_LOAD` parameter descriptions, see ML_MODEL_LOAD.

Before You Begin

- Review how to Train a Model.

Load a Model with The Session Variable

After training a model, you set a session variable for the model handle that you can use until the current connection ends.

The following example loads a MySQL HeatWave AutoML model from the model catalog by using the session variable

```
mysql> CALL sys.ML_MODEL_LOAD(@census_model, NULL);
```

Where:

- `@census_model` is the session variable that contains the model handle.
- `NULL` is specified in place of the user name of the model owner. You are not required to specify a user name.

Load a Model Handle with The Defined Model Handle Name

Before training a machine learning model, it is good practice to define a model handle name instead of automatically generating one. This allows you to easily remember the model handle for future routines on the trained model instead of having to query it, or depending on the session variable that can no longer be used when the current connection terminates. See ML_TRAIN Example.

The following example uses the defined model handle name to load the model.

```
mysql> CALL sys.ML_MODEL_LOAD('census_test', NULL);
```

Load the Model with The Automatically Generated Model Handle

If you do not define a model handle name before training a machine learning model, it is automatically generated. If the connection for the session variable of a model handle ends, you need to load the model with the model name.

1. Query the model handle, model owner, and the trained table name from the model catalog table. Replace `user1` with your own user name.

```
mysql> SELECT model_handle, model_owner, train_table_name FROM ML_SCHEMA_user1.MODEL_CATALOG;
+-------------------------------------------------+-------------+--------------------------------+
| model_handle                                    | model_owner | train_table_name               |
+-------------------------------------------------+-------------+--------------------------------+
| census_data.census_train_admin_1745261646953    | admin       | census_data.census_train       |
| census_data.census_train_admin_1745334557047    | admin       | census_data.census_train       |
| census_data.census_train_admin_1745336500455    | admin       | census_data.census_train       |
+-------------------------------------------------+-------------+--------------------------------+
3 rows in set (0.0431 sec)
```

2. Copy the appropriate `model_handle` and use it to load the machine learning model.

```
mysql> CALL sys.ML_MODEL_LOAD('census_data.census_train_user1_1745261646953', NULL);
```

Verify Model is Loaded

You have the option to verify that model is loaded by using the ML_MODEL_ACTIVE routine.

The following example verifies the model previously loaded is active.

1. Run ML_MODEL_ACTIVE on all active and loaded models and assign a session variable.

```
mysql> CALL sys.ML_MODEL_ACTIVE('all', @variable);
```

Replace *variable* with your own value. For example:

```
mysql> CALL sys.ML_MODEL_ACTIVE('all', @models);
```

2. Query the session variable previously created. Replace *models* with your own value.

```
mysql> SELECT @models;
+-----------------------------------------------------------------------------------------------------------
| @models
+-----------------------------------------------------------------------------------------------------------
| [{"total model size(bytes)": 388948}, {"admin": [{"census_test": {"format": "HWMLv2.0", "model_siz
+-----------------------------------------------------------------------------------------------------------
1 row in set (0.0431 sec)
```

The output displays the loaded model with information on the user that trained the model, the size of the model, the model handle, and its format.

What's Next

- For details on all model load options, see ML_MODEL_LOAD.
- Learn how to Generate Predictions.

6.5.4 Generate Predictions

Predictions are generated by running ML_PREDICT_ROW or ML_PREDICT_TABLE on trained models. The row or table of data must have the same feature columns as the data used to train the model. If the target column exists in the data to run predictions on, it is not considered during prediction generation.

ML_PREDICT_ROW generates predictions for one or more rows of data. ML_PREDICT_TABLE generates predictions for an entire table of data and saves the results to an output table.

6.5.4.1 Generate Predictions for a Row of Data

ML_PREDICT_ROW generates predictions for one or more rows of data specified in JSON format. You invoke the routine with the SELECT statement.

This topic has the following sections.

- Before You Begin
- Prepare to Generate a Row Prediction
- Input Data to Generate a Row Prediction
- Generate Predictions on One or More Rows of Data
- What's Next

Before You Begin

- Review the following:
  - Prepare Data

- Train a Model
- Load a Model

Prepare to Generate a Row Prediction

Before running `ML_PREDICT_ROW`, you must train, and then load the model you want to use.

1. The following example trains a dataset with the classification machine learning task.

```
mysql> CALL sys.ML_TRAIN('census_data.census_train', 'revenue', JSON_OBJECT('task', 'classification'), (
```

2. The following example loads the trained model.

```
mysql> CALL sys.ML_MODEL_LOAD(@census_model, NULL);
```

For more information about training and loading models, see Train a Model and Load a Model.

After training and loading the model, you can generate predictions on one or more rows of data. For parameter and option descriptions, see ML_PREDICT_ROW.

Input Data to Generate a Row Prediction

One way to generate predictions on row data is to manually enter the row data into a session variable, and then generate a prediction by specifying the session variable.

1. Define values for each column to predict. The column names must match the feature column names in the trained table.

```
mysql> SET @variable = (JSON_OBJECT("column_name", value, "column_name", value, ...), model_handle, opt
```

In the following example, create the `@row_input` session variable and enter the data to predict into the session variable.

```
mysql> SET @row_input = JSON_OBJECT(
          "age", 25,
          "workclass", "Private",
          "fnlwgt", 226802,
          "education", "11th",
          "education-num", 7,
          "marital-status", "Never-married",
          "occupation", "Machine-op-inspct",
          "relationship", "Own-child",
          "race", "Black",
          "sex", "Male",
          "capital-gain", 0,
          "capital-loss", 0,
          "hours-per-week", 40,
          "native-country", "United-States");
```

2. Run `ML_PREDICT_ROW` and specify the session variable set previously. Optionally, use `\G` to display information in an easily readable format.

```
mysql> SELECT sys.ML_PREDICT_ROW(@variable, ...), model_handle, options);
```

Replace *`variable`*, *`model_handle`*, and *`options`* with your own values. For example:

```
mysql> SELECT sys.ML_PREDICT_ROW(@row_input, @census_model, NULL)\G
*************************** 1. row ***************************
sys.ML_PREDICT_ROW(@row_input, @census_model, NULL):
{
    "age": 25,
    "sex": "Male",
    "race": "Black",
    "fnlwgt": 226802,
    "education": "11th",
    "workclass": "Private",
```

```
    "Prediction": "<=50K",
    "ml_results": {
        "predictions": {
            "revenue": "<=50K"
        },
        "probabilities": {
            ">50K": 0.0032,
            "<=50K": 0.9968
        }
    },
    "occupation": "Machine-op-inspct",
    "capital-gain": 0,
    "capital-loss": 0,
    "relationship": "Own-child",
    "education-num": 7,
    "hours-per-week": 40,
    "marital-status": "Never-married",
    "native-country": "United-States"
}
1 row in set (2.2218 sec)
```

Where:

- `@row_input` is a session variable containing a row of unlabeled data. The data is specified in `JSON` key-value format. The column names must match the feature column names in the training dataset.
- `@census_model` is the session variable that contains the model handle. Learn more about Model Handles.
- `NULL` sets no options to the routine.

The prediction on the data is that the revenue is `<=50K with a probability of 99.7%.`.

Generate Predictions on One or More Rows of Data

Another way to generate predictions is to create a `JSON_OBJECT` with specified columns and labels, and then generate predictions on one or more rows of data in the table.

```
mysql> SELECT sys.ML_PREDICT_ROW(JSON_OBJECT("output_col_name", schema.`input_col_name`, "output_col_na
                                                model_handle, options) FROM input_table_name LIMIT N;
```

The following example specifies the table and columns to use for the prediction and assigns output labels for each table-column pair. No options are set with `NULL`. It also defines to predict the top two rows of the table. Optionally, use `\G` to display information in an easily readable format.

```
mysql> SELECT sys.ML_PREDICT_ROW(JSON_OBJECT(
 "age", census_train.`age`,
 "workclass", census_train.`workclass`,
 "fnlwgt", census_train.`fnlwgt`,
 "education", census_train.`education`,
 "education-num", census_train.`education-num`,
 "marital-status", census_train.`marital-status`,
 "occupation", census_train.`occupation`,
 "relationship", census_train.`relationship`,
 "race", census_train.`race`,
 "sex", census_train.`sex`,
 "capital-gain", census_train.`capital-gain`,
 "capital-loss", census_train.`capital-loss`,
 "hours-per-week", census_train.`hours-per-week`,
 "native-country", census_train.`native-country`),
 @census_model, NULL)FROM census_data.census_train LIMIT 2\G
*************************** 1. row ***************************
sys.ML_PREDICT_ROW(JSON_OBJECT(
"age", census_train.`age`,
"workclass", census_train.`workclass`,
"fnlwgt", census_train.`fnlwgt`,
"education", census_train.`education`,
"education-num", census_train.`education-num`,
```

```
"marital-status", census_train.`marita: {
                                        "age": 62,
                                        "sex": "Female",
                                        "race": "White",
                                        "fnlwgt": 123582,
                                        "education": "10th",
                                        "workclass": "Private",
                                        "Prediction": "<=50K",
                                        "ml_results": {
                                            "predictions": {
                                                "revenue": "<=50K"
                                            },
                                            "probabilities": {
                                                ">50K": 0.0106,
                                                "<=50K": 0.9894
                                            }
                                        },
                                        "occupation": "Other-service",
                                        "capital-gain": 0,
                                        "capital-loss": 0,
                                        "relationship": "Unmarried",
                                        "education-num": 6,
                                        "hours-per-week": 40,
                                        "marital-status": "Divorced",
                                        "native-country": "United-States"
                                    }
*************************** 2. row ***************************
sys.ML_PREDICT_ROW(JSON_OBJECT(
"age", census_train.`age`,
"workclass", census_train.`workclass`,
"fnlwgt", census_train.`fnlwgt`,
"education", census_train.`education`,
"education-num", census_train.`education-num`,
"marital-status", census_train.`marita: {
                                        "age": 32,
                                        "sex": "Female",
                                        "race": "White",
                                        "fnlwgt": 174215,
                                        "education": "Bachelors",
                                        "workclass": "Federal-gov",
                                        "Prediction": "<=50K",
                                        "ml_results": {
                                            "predictions": {
                                                "revenue": "<=50K"
                                            },
                                            "probabilities": {
                                                ">50K": 0.3249,
                                                "<=50K": 0.6751
                                            }
                                        },
                                        "occupation": "Exec-managerial",
                                        "capital-gain": 0,
                                        "capital-loss": 0,
                                        "relationship": "Not-in-family",
                                        "education-num": 13,
                                        "hours-per-week": 60,
                                        "marital-status": "Never-married",
                                        "native-country": "United-States"
                                    }
2 rows in set (9.6548 sec)
```

The output generates revenue predictions for the four rows of data.

What's Next

- Review ML_PREDICT_ROW for parameter descriptions and options.
- After generating predictions for a row of data, learn how to Generate Explanations for a Row of Data to get insight into which features have the most influence on the predictions.
- Learn how to Generate Predictions for a Table.

- Learn how to Score a Model to get insight into the quality of the model.

6.5.4.2 Generate Predictions for a Table

`ML_PREDICT_TABLE` generates predictions for an entire table of trained data. Predictions are performed in parallel.

`ML_PREDICT_TABLE` is a compute intensive process. Depending on your MySQL version, we recommend the following:

- Before MySQL 9.4.1, limit operations to batches of rows by splitting large tables into smaller tables by using the `batch_size` option.
- For MySQL 9.4.1 and later, if `ML_PREDICT_TABLE` takes a long time to complete, manually limit input tables to a maximum of 1,000 rows.

See ML_PREDICT_TABLE and Track Progress for MySQL HeatWave AutoML Routines to learn more.

Before You Begin

- Review the following:
  - Prepare Data
  - Train a Model
  - Load a Model

Input Tables and Output Tables

Before MySQL 9.4.1, the output table of predictions must be a new table with a unique name.

As of MySQL 9.4.1, you can specify the output table and the input table as the same table if all the following conditions are met:

- The input table is not a Lakehouse external table.
- The input table does not have the columns that are created for the output table when generating predictions. Output columns are specific to each machine learning task. Some of these columns include:
  - `Prediction`
  - `ml_results`
- The input table does not have a primary key, and it does not have a column named `_4aad19ca6e_pk_id`. This is because `ML_PREDICT_TABLE` adds a column as the primary key with the name `_4aad19ca6e_pk_id` to the output table.
- The input table was not trained with the `log_anomaly_detection` task.

If you specify the output table and the input table as the same name, the predictions are inserted into the input table.

Prepare to Generate Predictions for a Table

Before running `ML_PREDICT_TABLE`, you must train, and then load the model you want to use.

1. The following example trains a dataset with the classification machine learning task.

```
mysql> CALL sys.ML_TRAIN('census_data.census_train', 'revenue', JSON_OBJECT('task', 'classification'
```

2. The following example loads the trained model.

```
mysql> CALL sys.ML_MODEL_LOAD(@census_model, NULL);
```

For more information about training and loading models, see Train a Model and Load a Model.

After training and loading the model, you can generate predictions for a table of data. For parameter and option descriptions, see ML_PREDICT_TABLE.

Generate Predictions for a Table

To generate predictions for a table, define the input table, the model handle, the output table, and any additional options.

```
mysql> CALL sys.ML_PREDICT_TABLE(table_name, model_handle, output_table_name, [options]);
```

The following example generates predictions for the entire table in the trained and loaded model.

```
mysql> CALL sys.ML_PREDICT_TABLE('census_data.census_train', @census_model, 'census_data.census_train_predi
```

Where:

- `census_data.census_train` is the fully qualified name of the test dataset table (*`schema_name.table_name`*). The table must have the same feature column names as the training dataset. The target column is not required. If it present in the table, it is not considered when generating predictions.
- `@census_model` is the session variable that contains the model handle. Learn more about Model Handles.
- `census_data.census_train_predictions` is the output table where predictions are stored. A fully qualified table name must be specified (*`schema_name.table_name`*).
- `NULL` sets no options to the routine.

When the output table is created, you can query a sample of the table to review predictions.

```
mysql> SELECT * FROM table_name LIMIT N;
```

Replace *`table_name`* with your own table name, and *`N`* with the number of rows from the table you want to view.

The following example queries the top five rows of the output table.

```
mysql> SELECT * FROM census_train_predictions LIMIT 5;
+------------------+-----+------------------+--------+-------------+---------------+--------------------+
| _4aad19ca6e_pk_id | age | workclass        | fnlwgt | education   | education-num | marital-status     |
+------------------+-----+------------------+--------+-------------+---------------+--------------------+
|                1 |  37 | Private          |  99146 | Bachelors   |            13 | Married-civ-spouse |
|                2 |  34 | Private          |  27409 | 9th         |             5 | Married-civ-spouse |
|                3 |  30 | Private          | 299507 | Assoc-acdm  |            12 | Separated          |
|                4 |  62 | Self-emp-not-inc | 102631 | Some-college |            10 | Widowed            |
|                5 |  51 | Private          | 153486 | Some-college |            10 | Married-civ-spouse |
+------------------+-----+------------------+--------+-------------+---------------+--------------------+
5 rows in set (0.0014 sec)
```

The predictions and associated probabilities are displayed in the `ml_results` column. You can compare the predicted revenue values with the real revenue values in the table. If needed, you can refine and train different sets of data to try and generate more reliable predictions.

What's Next

- Review ML_PREDICT_TABLE for parameter descriptions and options.
- After generating predictions on a table, learn how to Generate Explanations on a table to get insights into which features have the most influence on the predictions.
- Learn how to Generate Predictions for a Row of Data.
- Learn how to Score a Model to get insight into the quality of the model.

6.5.5 Generate Model Explanations

After the `ML_TRAIN` routine, use the `ML_EXPLAIN` routine to train model explainers for MySQL HeatWave AutoML. By default, the `ML_TRAIN` routine trains the Permutation Importance model explainer.

This topic has the following sections.

- Before You Begin
- Explanations Overview
- Model Explainers
- Unsupported Model Types
- Prepare to Generate a Model Explanation
- Retrieve the Default Permutation Importance Explanation
- Generate a Model Explanation
- What's Next

Before You Begin

- Review the following:
  - Prepare Data
  - Train a Model
  - Load a Model

Explanations Overview

Explanations help you understand which features have the most influence on a prediction. Feature importance is presented as an attribution value. A positive value indicates that a feature contributed toward the prediction. A negative value can have different interpretations depending on the specific model explainer used for the model. For example, a negative value for the permutation importance explainer means that the feature is not important.

Model Explainers

Model explainers are used when you run the `ML_EXPLAIN` routine to explain what the model learned from the training dataset. The model explainer provides a list of feature importances to show what features the model considered important based on the entire training dataset. The `ML_EXPLAIN` routine can train these model explainers:

- The Permutation Importance model explainer, specified as `permutation_importance`, is the default model explainer. `ML_TRAIN` generates this model explainer when it runs.
- The Partial Dependence model explainer, specified as `partial_dependence`, shows how changing the values of one or more columns changes the value that the model predicts. When you train this model explainer, you need to specify some additional options. See ML_EXPLAIN to learn more.
- The SHAP model explainer, specified as `shap`, produces feature importance values based on Shapley values.
- The Fast SHAP model explainer, specified as `fast_shap`, is a subsampling version of the SHAP model explainer, which usually has a faster runtime.

The model explanation is stored in the model catalog along with the machine learning model in the `model_explanation` column. See The Model Catalog. If you run `ML_EXPLAIN` again for the same model handle and model explainer, the field is overwritten with the new result.

Unsupported Model Types

You cannot generate model explanations for the following model types:

- Forecasting
- Recommendation
- Anomaly detection
- Anomaly detection for logs
- Topic modeling

Prepare to Generate a Model Explanation

Before running `ML_EXPLAIN`, you must train, and then load the model you want to use.

1. The following example trains a dataset with the classification machine learning task.

```
mysql> CALL sys.ML_TRAIN('census_data.census_train', 'revenue', JSON_OBJECT('task', 'classification'), (
```

2. The following example loads the trained model.

```
mysql> CALL sys.ML_MODEL_LOAD(@census_model, NULL);
```

For more information about training and loading models, see Train a Model and Load a Model.

After training and loading the model, you can generate model explanations. For option and parameter descriptions, see ML_EXPLAIN.

Retrieve the Default Permutation Importance Explanation

After training and loading a model, you can retrieve the default model explanation using the `permutation_importance` explainer from the model catalog. See The Model Catalog.

```
mysql> SELECT column FROM ML_SCHEMA_user1.MODEL_CATALOG WHERE model_handle=model_handle;
```

The following example retrieves the model explainer column from the model catalog of the previously trained model. The `JSON_PRETTY` parameter displays the output in an easily readable format.

```
mysql> SELECT JSON_PRETTY(model_explanation) FROM ML_SCHEMA_user1.MODEL_CATALOG WHERE model_handle=@census_
+-----------------------------------------------------------------------------------------------------------
| JSON_PRETTY(model_explanation)
+-----------------------------------------------------------------------------------------------------------
| {
  "permutation_importance": {
    "age": 0.0292,
    "sex": 0.0023,
    "race": 0.0019,
    "fnlwgt": 0.0038,
    "education": 0.0008,
    "workclass": 0.0068,
    "occupation": 0.0223,
    "capital-gain": 0.0479,
    "capital-loss": 0.0117,
    "relationship": 0.0234,
    "education-num": 0.0352,
    "hours-per-week": 0.0148,
    "marital-status": 0.024,
    "native-country": 0.0
  }
} |
+-----------------------------------------------------------------------------------------------------------
```

```
1 row in set (0.0427 sec)
```

Replace `user1` and `@census_model` with your own user name and session variable.

The explanation displays values of permutation importance for each column.

Generate a Model Explanation

To generate a model explanation, run the `ML_EXPLAIN` routine.

```
mysql> CALL sys.ML_EXPLAIN ('table_name', 'target_column_name', model_handle, [options]);
```

The following example generates a model explanation on the trained and loaded model with the `shap` model explainer.

```
mysql> CALL sys.ML_EXPLAIN('census_data.census_train', 'revenue', @census_model, JSON_OBJECT('model_exp
```

Where:

- `census_data.census_train` is the fully qualified name of the table that contains the training dataset (*`schema_name.table_name`*).
- `revenue` is the name of the target column, which contains ground truth values.
- `@census_model` is the session variable for the trained model.
- `model_explainer` is set to `shap` for the SHAP model explainer.

After running `ML_EXPLAIN`, you can view the model explanation in the Model Catalog. See The Model Catalog. The following example views the model explanation for the previous command. It provides values for each column representing importance values with the `shap` explainer.

```
mysql> SELECT JSON_PRETTY(model_explanation) FROM ML_SCHEMA_user1.MODEL_CATALOG WHERE model_handle=@cen
+-----------------------------------------------------------------------------------------------------------
| JSON_PRETTY(model_explanation)
+-----------------------------------------------------------------------------------------------------------
| {
  "shap": {
    "age": 0.0467,
    "sex": 0.033,
    "race": 0.0155,
    "fnlwgt": 0.0185,
    "education": 0.016,
    "workclass": 0.0255,
    "occupation": 0.0001,
    "capital-gain": 0.0217,
    "capital-loss": 0.0001,
    "relationship": 0.0426,
    "education-num": 0.0186,
    "hours-per-week": 0.0148,
    "marital-status": 0.024,
    "native-country": 0.0
  },
  "permutation_importance": {
    "age": -0.0057,
    "sex": 0.0002,
    "race": 0.0001,
    "fnlwgt": 0.0103,
    "education": 0.0108,
    "workclass": 0.0189,
    "occupation": 0.0,
    "capital-gain": 0.0304,
    "capital-loss": 0.0,
    "relationship": 0.0195,
    "education-num": 0.0152,
    "hours-per-week": 0.0235,
    "marital-status": 0.0099,
    "native-country": 0.0
   }
} |
```

```
+-----------------------------------------------------------------------------------------------------------------
1 row in set (0.0427 sec)
```

What's Next

- Review ML_EXPLAIN for parameter descriptions and options.
- Learn how to Generate Prediction Explanations.
- Learn more about the The Model Catalog.

6.5.6 Generate Prediction Explanations

Prediction explanations are generated by running `ML_EXPLAIN_ROW` or `ML_EXPLAIN_TABLE` on unlabeled data. The data must have the same feature columns as the data used to train the model. The target column is not required.

Prediction explanations are similar to model explanations, but rather than explain the whole model, prediction explanations explain predictions for individual rows of data. See Explanations Overview to learn more.

You can train the following prediction explainers:

- The Permutation Importance prediction explainer, specified as `permutation_importance`, is the default prediction explainer, which explains the prediction for a single row or table. Right after training and loading a model, you can run `ML_EXPLAIN_ROW` and `ML_EXPLAIN_TABLE` with this prediction explainer directly without having to run `ML_EXPLAIN` first.
- The SHAP prediction explainer, specified as `shap`, uses feature importance values to explain the prediction for a single row or table. To run this prediction explainer with `ML_EXPLAIN_ROW` and `ML_EXPLAIN_TABLE`, you must run `ML_EXPLAIN` first.

`ML_EXPLAIN_ROW` generates explanations for one or more rows of data. `ML_EXPLAIN_TABLE` generates explanations on an entire table of data and saves the results to an output table. `ML_EXPLAIN_*` routines limit explanations to the 100 most relevant features.

6.5.6.1 Generate Prediction Explanations for a Row of Data

`ML_EXPLAIN_ROW` explains predictions for one or more rows of unlabeled data. You invoke the routine by using a `SELECT` statement.

This topic has the following sections.

- Before You Begin
- Unsupported Model Types
- Prepare to Generate a Row Explanation
- Generate a Row Prediction Explanation with the Default Permutation Importance Explainer
- Generate a Row Prediction Explanation with the SHAP Explainer
- What's Next

Before You Begin

- Review the following:
  - Prepare Data
  - Train a Model
  - Load a Model

Unsupported Model Types

You cannot generate prediction explanations on a row of data for the following model types:

- Forecasting
- Recommendation
- Anomaly detection
- Anomaly detection for logs
- Topic modeling

Prepare to Generate a Row Explanation

Before running `ML_EXPLAIN_ROW`, you must train, and then load the model you want to use.

1. The following example trains a dataset with the classification machine learning task.

```
mysql> CALL sys.ML_TRAIN('census_data.census_train', 'revenue', JSON_OBJECT('task', 'classification'
```

2. The following example loads the trained model.

```
mysql> CALL sys.ML_MODEL_LOAD(@census_model, NULL);
```

For more information about training and loading models, see Train a Model and Load a Model.

After training and loading the model, you can generate prediction explanations for one or more rows. For parameter and option descriptions, see ML_EXPLAIN_ROW.

Generate a Row Prediction Explanation with the Default Permutation Importance Explainer

After training and loading a model, you can run `ML_EXPLAIN_ROW` to generate a row prediction explanation with the default Permutation Importance explainer. However, if you train the `shap` prediction explainer with `ML_EXPLAIN`, you need to run `ML_EXPLAIN` again with the `permutation_importance` explainer before running `ML_EXPLAIN_ROW` with the same explainer.

The following example enters a row of data to explain into a session variable. The session variable is then used in the `ML_EXPLAIN_ROW` routine.

1. Define values for each column to predict. The column names must match the feature column names in the trained table.

```
mysql> SET @variable = (JSON_OBJECT("column_name", value, "column_name", value, ...), model_handle,
```

In the following example, assign the data to analyze into the `@row_input` session variable.

```
mysql> SET @row_input = JSON_OBJECT(
          "age", 31,
          "workclass", "Private",
          "fnlwgt", 45781,
          "education", "Masters",
          "education-num", 14,
          "marital-status", "Married-civ-spouse",
          "occupation", "Prof-specialty",
          "relationship", "Not-in-family",
          "race", "White",
          "sex", "Female",
          "capital-gain", 14084,
          "capital-loss", 2042,
          "hours-per-week", 40,
          "native-country", "India");
```

2. Run the `ML_EXPLAIN_ROW` routine.

```
mysql> SELECT sys.ML_EXPLAIN_ROW(input_data, model_handle, [options]);
```

In the following example, include the session variable previously created. Optionally, use `\G` to display the output in an easily readable format. The output is similar to the following:

```
mysql> SELECT sys.ML_EXPLAIN_ROW(@row_input, @census_model, JSON_OBJECT('prediction_explainer', 'permuta
*************************** 1. row ***************************
sys.ML_EXPLAIN_ROW(@row_input, @census_model,
          JSON_OBJECT('prediction_explainer', 'permutation_importance')):
          {
                "age": 31,
                "sex": "Female",
                "race": "White",
                "Notes": "capital-gain (14084) had the largest impact towards predicting >50K",
                "fnlwgt": 45781,
                "education": "Masters",
                "workclass": "Private",
                "Prediction": ">50K",
                "ml_results": {
                   "notes": "capital-gain (14084) had the largest impact towards predicting >50K",
                   "predictions": {
                       "revenue": ">50K"
                    },
                    "attributions": {
                        "age": 0.34,
                        "sex": 0,
                        "race": 0,
                        "fnlwgt": 0,
                        "education": 0,
                        "workclass": 0,
                        "occupation": 0,
                        "capital-gain": 0.97,
                        "capital-loss": 0,
                        "relationship": 0,
                        "education-num": 0.04,
                        "hours-per-week": 0,
                        "marital-status": 0
                    }
                },
                "occupation": "Prof-specialty",
                "capital-gain": 14084,
                "capital-loss": 2042,
                "relationship": "Not-in-family",
                "education-num": 14,
                "hours-per-week": 40,
                "marital-status": "Married-civ-spouse",
                "native-country": "India",
                "age_attribution": 0.34,
                "sex_attribution": 0,
                "race_attribution": 0,
                "fnlwgt_attribution": 0,
                "education_attribution": 0,
                "workclass_attribution": 0,
                "occupation_attribution": 0,
                "capital-gain_attribution": 0.97,
                "capital-loss_attribution": 0,
                "relationship_attribution": 0,
                "education-num_attribution": 0.04,
                "hours-per-week_attribution": 0,
                "marital-status_attribution": 0
          }
1 row in set (6.3072 sec)
```

The output provides an explanation on the column that had the largest impact towards the prediction, and the column that contributed the most against the prediction.

Generate a Row Prediction Explanation with the SHAP Explainer

To generate a row prediction explanation with the SHAP explainer, you must first run the SHAP explainer with `ML_EXPLAIN`.

1. Run the ML_EXPLAIN routine.

```
mysql> CALL sys.ML_EXPLAIN ('table_name', 'target_column_name', model_handle, [options]);
```

The following example runs the shap explainer.

```
mysql> CALL sys.ML_EXPLAIN('census_data.census_train', 'revenue', @census_model, JSON_OBJECT('predic
```

Where:

- census_data.census_train is the fully qualified name of the table that contains the training dataset (*schema_name.table_name*).
- revenue is the name of the target column, which contains ground truth values.
- @census_model is the session variable for the trained model.
- prediction_explainer is set to shap for the SHAP prediction explainer.

2. Define values for each column to predict. The column names must match the feature column names in the trained table.

```
mysql> SET @variable = (JSON_OBJECT("column_name", value, "column_name", value, ...), model_handle,
```

In the following example, assign the data to analyze into the @row_input session variable.

```
mysql> SET @row_input = JSON_OBJECT(
          "age", 25,
          "workclass", "Private",
          "fnlwgt", 226802,
          "education", "11th",
          "education-num", 7,
          "marital-status", "Never-married",
          "occupation", "Machine-op-inspct",
          "relationship", "Own-child",
          "race", "Black",
          "sex", "Male",
          "capital-gain", 0,
          "capital-loss", 0,
          "hours-per-week", 40,
          "native-country", "United-States");
```

3. Run the ML_EXPLAIN_ROW routine.

```
mysql> SELECT sys.ML_EXPLAIN_ROW(input_data, model_handle, [options]);
```

In the following example run the same shap prediction explainer. Optionally, use \G to display the output in an easily readable format.

```
mysql> SELECT sys.ML_EXPLAIN_ROW(@row_input, @census_model, JSON_OBJECT('prediction_explainer', 'sha
*************************** 1. row ***************************
sys.ML_EXPLAIN_ROW(@row_input, @census_model,
  JSON_OBJECT('prediction_explainer', 'shap')):
  {
    "age": 25,
    "sex": "Male",
    "race": "Black",
    "fnlwgt": 226802,
    "education": "11th",
    "workclass": "Private",
    "Prediction": "<=50K",
    "ml_results": {
        "predictions": {
            "revenue": "<=50K"
        },
        "attributions": {
            "age_attribution": 0.03154012309521936,
            "sex_attribution": -0.002995059121088509,
            "race_attribution": 0.0051264089998398765,
```

```
            "fnlwgt_attribution": -0.003139455788215409,
            "education_attribution": 0.0013752672453250653,
            "workclass_attribution": 0,
            "occupation_attribution": 0.020919219303459986,
            "capital-gain_attribution": 0.015089815859614985,
            "capital-loss_attribution": 0.0033537962775555263,
            "relationship_attribution": 0.027744370891787523,
            "education-num_attribution": 0.0284122832892542,
            "hours-per-week_attribution": 0.009110644648945954,
            "marital-status_attribution": 0.036222463769272406
        }
    },
    "occupation": "Machine-op-inspct",
    "capital-gain": 0,
    "capital-loss": 0,
    "relationship": "Own-child",
    "education-num": 7,
    "hours-per-week": 40,
    "marital-status": "Never-married",
    "native-country": "United-States",
    "age_attribution": 0.0315401231,
    "sex_attribution": -0.0029950591,
    "race_attribution": 0.005126409,
    "fnlwgt_attribution": -0.0031394558,
    "education_attribution": 0.0013752672,
    "workclass_attribution": 0,
    "occupation_attribution": 0.0209192193,
    "capital-gain_attribution": 0.0150898159,
    "capital-loss_attribution": 0.0033537963,
    "relationship_attribution": 0.0277443709,
    "education-num_attribution": 0.0284122833,
    "hours-per-week_attribution": 0.0091106446,
    "marital-status_attribution": 0.0362224638
}
1 row in set (4.3007 sec)
```


The output displays feature importance values for each column.

What's Next

- Review ML_EXPLAIN_ROW for parameter descriptions and options.
- Learn how to Generate Explanations for a Table.
- Learn how to Score a Model to get insight into the quality of the model.

6.5.6.2 Generate Prediction Explanations for a Table

`ML_EXPLAIN_TABLE` explains predictions for an entire table of unlabeled data. Explanations are performed in parallel.

Note

`ML_EXPLAIN_TABLE` is a very memory-intensive process. Depending on your MySQL version, we recommend the following:

- Before MySQL 9.4.1, use the `batch_size` option to limit operations to batches of 10 to 100 rows by splitting large tables into smaller tables. For tables with over ten columns, we suggest a `batch_size` value of `10`.
- As of MySQL 9.4.1, the `batch_size` option is deprecated. Limit the input table to a maximum of 100 rows. If the input table has more than ten columns, limit it to ten rows.

See ML_EXPLAIN_TABLE and Track Progress for MySQL HeatWave AutoML Routines to learn more.

Before You Begin

- Review the following:
  - Prepare Data
  - Train a Model
  - Load a Model

Unsupported Model Types

You cannot generate prediction explanations on a table for the following model types:

- Forecasting
- Recommendation
- Anomaly detection
- Anomaly detection for logs
- Topic modeling

Input Tables and Output Tables

Before MySQL 9.4.1, the output table of predictions must be a new table with a unique name.

As of MySQL 9.4.1, you can specify the output table and the input table as the same table if all the following conditions are met:

- The input table is not a Lakehouse external table.
- The input table does not have the columns that are created for the output table when generating predictions. Output columns are specific to each machine learning task. Some of these columns include:
  - `Prediction`
  - `ml_results`
  - `[input_column_name]_attribution`
- The input table does not have a primary key, and it does not have a column named `_4aad19ca6e_pk_id`. This is because `ML_EXPLAIN_TABLE` adds a column as the primary key with the name `_4aad19ca6e_pk_id` to the output table.

If you specify the output table and the input table as the same name, the predictions are inserted into the input table.

Prepare to Generate Explanations for a Table

Before running `ML_EXPLAIN_TABLE`, you must train, and then load the model you want to use.

1. The following example trains a dataset with the classification machine learning task.

```
mysql> CALL sys.ML_TRAIN('census_data.census_train', 'revenue', JSON_OBJECT('task', 'classification'
```

2. The following example loads the trained model.

```
mysql> CALL sys.ML_MODEL_LOAD(@census_model, NULL);
```

For more information about training and loading models, see Train a Model and Load a Model.

After training and loading the model, you can generate prediction explanations for a table. For parameter and option descriptions, see ML_EXPLAIN_TABLE.

Generate Explanations for a Table with the Default Permutation Importance Explainer

After training and loading a model, you can run ML_EXPLAIN_TABLE to generate a table of prediction explanations with the default Permutation Importance explainer>. However, if you train the shap prediction explainer with ML_EXPLAIN, you need to run ML_EXPLAIN again with the permutation_importance explainer before running ML_EXPLAIN_TABLE with the same explainer.

1. Run the ML_EXPLAIN_TABLE routine.

```
mysql> CALL sys.ML_EXPLAIN_TABLE(table_name, model_handle, output_table_name, [options]);
```

The following example runs ML_EXPLAIN_TABLE with the permutation_importance explainer.

```
mysql> CALL sys.ML_EXPLAIN_TABLE('census_data.census_train', @census_model, 'census_data.census_train_p
                                 JSON_OBJECT('prediction_explainer', 'permutation_importance'));
```

Where:

- census_data.census_train is the fully qualified name of the table that contains the training dataset (*schema_name.table_name*).
- @census_model is the session variable for the trained model.
- census_data.census_train_permutation is the fully qualified name of the output table that contains the explanations (*schema_name.table_name*).
- prediction_explainer is set to permutation_importance for the Permutation Importance prediction explainer.

2. Query the output table to review a sample of the results.

```
mysql> SELECT * FROM table_name LIMIT N;
```

The following example queries the top three rows of the output table.

```
mysql> SELECT * FROM census_train_permutation LIMIT 3;
+-------------------+-----+-----------+--------+------------+---------------+--------------------+-----
| _4aad19ca6e_pk_id | age | workclass | fnlwgt | education  | education-num | marital-status     | occu
+-------------------+-----+-----------+--------+------------+---------------+--------------------+-----
|                 1 |  37 | Private   |  99146 | Bachelors  |            13 | Married-civ-spouse | Exec
|                 2 |  34 | Private   |  27409 | 9th        |             5 | Married-civ-spouse | Craf
|                 3 |  30 | Private   | 299507 | Assoc-acdm |            12 | Separated          | Othe
+-------------------+-----+-----------+--------+------------+---------------+--------------------+-----
```

The results display information on the columns that had the largest impact towards the predictions and the columns that contributed the most against the prediction.

A warning displays if the model is of low quality.

Generate Explanations for a Table with the SHAP Explainer

To generate a table of prediction explanations with the SHAP explainer, you must first run the SHAP explainer with ML_EXPLAIN.

1. Run the ML_EXPLAIN routine.

```
mysql> CALL sys.ML_EXPLAIN ('table_name', 'target_column_name', model_handle, [options]);
```

The following example runs the shap prediction explainer.

```
mysql> CALL sys.ML_EXPLAIN('census_data.census_train', 'revenue', @census_model, JSON_OBJECT('predictio
```

Where:

- `census_data.census_train` is the fully qualified name of the table that contains the training dataset (*`schema_name.table_name`*).
- `revenue` is the name of the target column, which contains ground truth values.
- `@census_model` is the session variable for the trained model.
- `prediction_explainer` is set to `shap` for the SHAP prediction explainer.

2. Run the `ML_EXPLAIN_TABLE` routine.

```
mysql> CALL sys.ML_EXPLAIN_TABLE(table_name, model_handle, output_table_name, [options]);
```

The following example runs the `shap` prediction explainer.

```
mysql> CALL sys.ML_EXPLAIN_TABLE('census_data.census_train', @census_model, 'census_data.census_trai
                                 JSON_OBJECT('prediction_explainer', 'shap'));
```

Where:

- `census_data.census_train` is the fully qualified name of the table that contains the training dataset (*`schema_name.table_name`*).
- `@census_model` is the session variable for the trained model.
- `census_data.census_train_explanations` is the fully qualified name of the output table that contains the explanations (*`schema_name.table_name`*).
- `prediction_explainer` is set to `shap` for the SHAP prediction explainer.

3. Query the output table to review a sample of the results.

```
mysql> SELECT * FROM table_name LIMIT N;
```

The following example queries the top three rows of the output table.

```
mysql> SELECT * FROM census_train_explanations LIMIT 3;
+-------------------+-----+-----------+--------+------------+---------------+-------------------+--
| _4aad19ca6e_pk_id | age | workclass | fnlwgt | education  | education-num | marital-status    | 
+-------------------+-----+-----------+--------+------------+---------------+-------------------+--
|                 1 |  37 | Private   |  99146 | Bachelors  |            13 | Married-civ-spouse | 
|                 2 |  34 | Private   |  27409 | 9th        |             5 | Married-civ-spouse | 
|                 3 |  30 | Private   | 299507 | Assoc-acdm |            12 | Separated          | 
+-------------------+-----+-----------+--------+------------+---------------+-------------------+--
```

The results display feature importance values for each column.

A warning displays if the model is of low quality.

What's Next

- Review ML_EXPLAIN_TABLE for parameter descriptions and options.
- Learn how to Score a Model to get insight into the quality of the model.

6.5.7 Score a Model

`ML_SCORE` scores a model by generating predictions using the feature columns in a labeled dataset as input and comparing the predictions to ground truth values in the target column of the labeled dataset.

You cannot score a model with a topic modeling task type.

Before You Begin

- Review the following:
  - Prepare Data
  - Train a Model
  - Load a Model
  - Generate Predictions
  - Generate Model Explanations
  - Generate Prediction Explanations

ML_SCORE Overview

The dataset used with `ML_SCORE` should have the same feature columns as the dataset used to train the model, but the data sample should be different from the data used to train the model. For example, you might reserve 20 to 30 percent of a labeled dataset for scoring.

`ML_SCORE` returns a computed metric indicating the quality of the model. A value of `None` is reported if a score for the specified or default metric cannot be computed. If an invalid metric is specified, the following error message is reported: `Invalid data for the metric. Score could not be computed.`

Models with a low score can be expected to perform poorly, producing predictions and explanations that cannot be relied upon. A low score typically indicates that the provided feature columns are not a good predictor of the target values. In this case, consider adding more rows or more informative features to the training dataset.

You can also run `ML_SCORE` on the training dataset and a labeled test dataset and compare results to ensure that the test dataset is representative of the training dataset. A high score on a training dataset and low score on a test dataset indicates that the test data set is not representative of the training dataset. In this case, consider adding rows to the training dataset that better represent the test dataset.

MySQL HeatWave AutoML supports a variety of scoring metrics to help you understand how your model performs across a series of benchmarks. The metric you select to score the model must be compatible with the `task` type and the target data. See Optimization and Scoring Metrics.

Prepare to Score a Model

Before running `ML_SCORE`, you must train, and then load the trained model you want to use for scoring.

1. The following example trains a dataset with the classification machine learning task.

```
mysql> CALL sys.ML_TRAIN('census_data.census_train', 'revenue', JSON_OBJECT('task', 'classification'), (
```

2. The following example loads the trained model.

```
mysql> CALL sys.ML_MODEL_LOAD(@census_model, NULL);
```

For more information about training and loading models, see Train a Model and Load a Model.

After training and loading the model, prepare a table of labeled data to score that has a different set of data from the trained model. This is considered the validation dataset. For parameter and option descriptions, see ML_SCORE.

Score a Model

To score a model, run the `ML_SCORE` routine.

```
mysql> CALL sys.ML_SCORE(table_name, target_column_name, model_handle, metric, score, [options]);
```

The following example uses the `accuracy` metric to compute model quality:

```
mysql> CALL sys.ML_SCORE('census_data.census_validate', 'revenue', @census_model, 'accuracy', @score, N
```

Where:

- `census_data.census_validate` is the fully qualified name of the validation dataset table (*`schema_name.table_name`*).
- `revenue` is the name of the target column containing ground truth values.
- `@census_model` is the session variable that contains the model handle.
- `accuracy` is the scoring metric. For other supported scoring metrics, see Optimization and Scoring Metrics.
- `@score` is the user-defined session variable that stores the computed score. The `ML_SCORE` routine populates the variable. User variables are written as *`@var_name`*. The examples in this guide use `@score` as the variable name. Any valid name for a user-defined variable is permitted, for example `@my_score`.
- `NULL` sets no options for the routine. To view available options, see ML_SCORE.

To retrieve the computed score, query the `@score` session variable.

```
mysql> SELECT @score;
+--------------------+
| @score             |
+--------------------+
| 0.8888888955116272 |
+--------------------+
1 row in set (0.0409 sec)
```

Review the score value and determine if the trained model is reliable enough for generating predictions and explanations.

What's Next

- Review ML_SCORE for parameter descriptions and options.
- Review Machine Learning Use Cases to create machine learning models with sample datasets.

6.6 Learn About MySQL HeatWave AutoML with Oracle Cloud Infrastructure Generative AI

Available as of MySQL 9.4.0, the `NL2ML` routine enables you to learn about MySQL HeatWave AutoML by leveraging Oracle Cloud Infrastructure Generative AI (OCI) and asking questions in natural language.

This topic has the following sections.

- Before You Begin
- Overview
- Use Oracle Cloud Infrastructure Generative AI to Learn About MySQL HeatWave AutoML
- Use NL2ML with In-Database LLMs
- View Chat History and Change Model

- NL2ML Use Cases
- What's Next

Before You Begin

- This feature is only available on MySQL HeatWave on OCI. It is not available for MySQL HeatWave on AWS.
- Before MySQL 9.6.0, `NL2ML` requires Oracle Cloud Infrastructure Generative AI, which is available in specific regions. To confirm if your region has Generative AI, see Regions with Generative AI.

  As of MySQL 9.6.0, if your region does not have OCI Generative AI, or you want to bypass OCI Generative AI and instead use in-database LLMs to generate responses, you can set the `skip_generate` option to `true`. See Use NL2ML with In-Database LLMs.
- To use OCI Generative AI to generate responses, you must complete the steps to Authenticate OCI Generative AI Service.
- Review the required tasks to Create a Machine Learning Model.

Overview

MySQL HeatWave AutoML is simple to use, and the introduction of NL2ML (natural language to machine learning) makes it even easier to use. NL2ML provides step-by-step guidelines on how to use MySQL HeatWave AutoML for a particular business problem.

You can use NL2ML to do the following:

- Ask general questions about using MySQL HeatWave AutoML.
- Use NL2ML to generate AutoML queries that you can copy and run. To generate AutoML queries, NL2ML leverages the databases, tables, and table columns in the DB System.

NL2ML uses pretrained foundational models supported by OCI Generative AI.

- Before MySQL 9.6.0, MySQL HeatWave in-database LLMs are not supported. To review available OCI Generative AI models, see Pretrained Foundational Models in Generative AI. For every major MySQL release, NL2ML creates a new vector store from the MySQL HeatWave User Guide documentation. After creating each new vector store, the old vector store gets deleted. Due to the creation of vector stores, some NL2ML calls may take longer to complete.
- As of MySQL 9.6.0, the `skip_generate` option allows `NL2ML` to bypass OCI Generative AI and only provide citations to MySQL HeatWave documentation. You can then use the citations as context to generate model training guidance and SQL queries with in-database LLMs. See Use NL2ML with In-Database LLMs.

MySQL 9.4.1 adds the following updates to `NL2ML`:

- Additional data preparation information for training queries. This includes detailed instructions to do the following:
  - Join your dataset's various tables into a single table.
  - Exclude unnecessary columns.
  - Run the `TRAIN_TEST_SPLIT` routine to automatically prepare training and testing datasets.
- If table and column names are non-descriptive, the ability to scan table and column comments in the table to retrieve additional contextual information for preparing queries.

Use Oracle Cloud Infrastructure Generative AI to Learn About MySQL HeatWave AutoML

Follow these steps to use OCI Generative AI to generate responses to questions about AutoML.

1. Run the `NL2ML` routine.

```
mysql> CALL sys.NL2ML (query, response);
```

   Replace *`query`* with your question in natural language, and *`response`* with the name of the session variable that has the generated answer to your question.

   The following example asks about the size limit of the training table and sets the response session variable to `@output`.

```
mysql> CALL sys.NL2ML("Is there any limit on the size of training table?", @output);
Query OK, 0 rows affected (11.0086 sec)
```

2. Query the response session variable to view the response. The following example queries the response from the previous question.

```
mysql> SELECT @output;
+-----------------------------------------------------------------------------------------------------------------
| @output |
+-----------------------------------------------------------------------------------------------------------------
| {"text": "Yes, there are limits on the size of the training table for HeatWave AutoML models. The
+-----------------------------------------------------------------------------------------------------------------
1 row in set (0.0003 sec)
```

Review additional Syntax Examples.

Use NL2ML with In-Database LLMs

To use `NL2ML` to provide citations to MySQL HeatWave documentation, and then leverage in-database LLMs to generate responses that include appropriate table schemas and commands, do the following:

- Set the `skip_generate` option to `true` with the `@nl2ml_options` session variable. This forces `NL2ML` to bypass OCI Generative AI and only provide relevant citations to MySQL HeatWave documentation.
- Use `ML_RETRIEVE_SCHEMA_METADATA` to retrieve the table schema related to the question asked during the `NL2ML` routine.
- Use `GROUP_CONCAT()` to build a compact context string from the citations provided by `NL2ML`.
- Use `ML_GENERATE` to specify the in-database LLM and generate the response to the question, which includes the citations, context, and retrieved table schema.

See the following example.

To use in-database LLMs with `NL2ML`:

1. Specify the question and set it into a variable (`@input`).

```
mysql> SET @input = "How can I train a model to predict net worth of a singer?";
```

2. Set the `skip_generate` option to `true` with the `@nl2ml_options` session variable.

```
mysql> SET @nl2ml_options = JSON_OBJECT("skip_generate", true);
```

3. Run the `NL2ML` routine and include the previous variable that has the question.

```
mysql> CALL sys.NL2ML(@input, @out);
```

4. View the output generated from the question by selecting the `@out` variable.

```
mysql> SELECT JSON_PRETTY(@out);
JSON_PRETTY(@out)
{
  "citations": [
    {
      "segment": "<segment content>",
      "distance": 0.1023,
      "document_name": "https://.../heatwave-en-ml-9.6.0.pdf",
      "segment_number": <segment number>
    },
    ...
  ],
  "vector_store": [
    "`ML_SCHEMA_userOne`.`hwml_vector_store_heatwave_en_ml_9_6_0`"
  ],
  "retrieval_info": {
    "method": "n_citations",
    "threshold": 0.114
  }
}
```

The output includes citations with the following information:

- `segment`: The relevant excerpts from the MySQL HeatWave documentation.
- `distance`: A value indicating how relevant the segment is to the question asked. A lower value represents a more relevant segment.
- `document_name`: A reference to the MySQL HeatWave documentation.
- `segment_number`: The index number identifying the segment.

5. Use `ML_RETRIEVE_SCHEMA_METADATA` to retrieve the most relevant table schema for the previous question.

```
mysql> CALL sys.ML_RETRIEVE_SCHEMA_METADATA(@input, @retrieved, NULL);
```

6. Retrieve the table schema from the `@retrieved` variable.

```
mysql> SELECT @retrieved;
@retrieved
CREATE TABLE `mlcorpus`.`singer`(
  `Singer_ID` int,
  `Name` varchar,
  `Birth_Year` double,
  `Net_Worth_Millions` double,
  `Citizenship` varchar
);
```

7. Use `GROUP_CONCAT()` to build a compact context string from the citations provided by `NL2ML`.

```
mysql> SELECT GROUP_CONCAT(seg SEPARATOR '\n') INTO @ctx
FROM JSON_TABLE(JSON_EXTRACT(@out,'$.citations'),
'$[*]' COLUMNS (seg LONGTEXT PATH '$.segment')) AS jt;
```

8. Combine the retrieved table schema and citations as the final context.

```
mysql> SET @final_ctx = CONCAT(@ctx, '\n\nRetrieved tables:\n', @retrieved);
```

9. Use `ML_GENERATE` to specify an in-database LLM (for example, `llama3.2-3b-instruct-v1`) and manually create a SQL statement that includes the citations, context, and retrieved table schema.

```
mysql> SELECT sys.ML_GENERATE(
@input,
JSON_OBJECT(
"task", "generation",
"model_id", "llama3.2-3b-instruct-v1",
```

```
"context", @final_ctx
)
) INTO @result;
```

10. Generate the output and SQL text.

```
mysql> SELECT JSON_UNQUOTE(JSON_EXTRACT(@result,'$.text')) AS generated_sql;
generated_sql
To train a model to predict the net worth of a singer, you can use the ML_TRAIN routine. First, prep
which in this case seems to be the 'singer' table in the 'mlcorpus' schema. Ensure that the table ha
such as 'Singer_ID', 'Name', 'Birth_Year', 'Net_Worth_Millions', and 'Citizenship'.

The 'Net_Worth_Millions' column will be your target column for prediction. You may need to preproces
for example, converting categorical variables like 'Name' and 'Citizenship' into numerical variables

Then, you can call the ML_TRAIN routine with the appropriate options. For a regression task like pre
you would specify the task as 'regression' in the JSON options. Here's a simplified example:

```sql
CALL sys.ML_TRAIN('mlcorpus.singer',
                  @model_handle,
                  'Net_Worth_Millions',
                  JSON_OBJECT('task', 'regression',
                              'algorithm', 'XGBRegressor'));
```

Replace '@model_handle' with your actual model handle variable. This will train a model to predict t
based on the other columns in your 'singer' table. After training, you can use the ML_PREDICT_ROW or
to generate predictions for new, unseen data.
```

View Chat History and Change Model

To view the chat history and current Generative AI model, query the `@nl2ml_options` session variable. Optionally, use `JSON_PRETTY` to view the output in an easily readable format.

```
mysql> SELECT JSON_PRETTY(@nl2ml_options);
```

Use one of the following commands to reset the chat history:

```
mysql> SET @nl2ml_options = NULL;

mysql> SET @nl2ml_options = JSON_REMOVE(@nl2ml_options, '$.chat_history');
```

You have the option to change the current Generative AI model. To review available options for `model_id`, see Pretrained Foundational Models in Generative AI.

To change the current Generative AI model and maintain the current chat history, use the following command:

```
mysql> SET @nl2ml_options = JSON_SET(@nl2ml_options, '$.mode_id', "model_id");
```

For example:

```
mysql> SET @nl2ml_options = JSON_SET(@nl2ml_options, '$.mode_id', "meta.llama-3.1-405b-instruct");
```

To change the current Generative AI model and reset the chat history, use the following command:

```
mysql> SET @nl2ml_options = JSON_OBJECT("model_id", "model_id");
```

For example:

```
mysql> SET @nl2ml_options = JSON_OBJECT("model_id", "meta.llama-3.1-405b-instruct");
```

NL2ML Use Cases

The following use case uses `NL2ML` to request AutoML queries to train, use, and evaluate a model for data related to a food delivery service. The examples show how you can use follow-up questions based on previous questions and responses.

The additional information for training queries is available as of MySQL 9.4.1.

Refer to the following questions and responses. The output for each question is similar to the following:

```
mysql> CALL sys.NL2ML("Give me a training query to predict how long it takes to deliver food", @output);
Query OK, 0 rows affected (6.8284 sec)

mysql> SELECT @output;
+-----------------------------------------------------------------------------------------------------------------------
| @output
+-----------------------------------------------------------------------------------------------------------------------
| {"text": "**Data Preparation: **\n\n To prepare the dataset for the machine learning task of regression,
|           \n\nFirst, let's integrate the tables into one full table using a `CREATE TABLE ... AS SELECT
|           \n\nFrom the provided dataset, we can see that the `orders` table contains information about or
|           \n\nWe can also join the `orders` table with the `delivery_logs` table based on the `order_id`
|           \n\n```sql\nCREATE TABLE integrated_table AS\nSELECT o.order_id, o.customer_id, o.restaurant_id
|           \n       r.x1 AS restaurant_name, r.x2 AS cuisine_type, r.average_prep_time, r.rating,\n
|           \nJOIN delivery_logs dl ON o.order_id = dl.order_id;\n```\n\nNext, we need to identify columns
|           \n\nFrom the integrated table, we can exclude the following columns:\n\n* `order_id` (unique id
|           \n* `rider_id` (unique identifier for each rider)\n* `log_time` (timestamp for each delivery lo
|           \n\n```json\nJSON_ARRAY('order_id', 'customer_id', 'restaurant_id', 'rider_id', 'log_time', 'st
|           \n\nHere's an example SQL query to split the data:\n\n```sql\nCALL sys.TRAIN_TEST_SPLIT('food_d
|           \n```\n\nThis will split the data into training and testing sets, with 80% of the data used for
|           \nJSON_OBJECT('task', 'regression', 'exclude_column_list', \nJSON_ARRAY('order_id', 'customer_i
+-----------------------------------------------------------------------------------------------------------------------
1 row in set (0.0003 sec)
mysql> CALL sys.NL2ML("how can I deploy this model?", @output);
Query OK, 0 rows affected (11.7221 sec)

mysql> SELECT @output;
+-----------------------------------------------------------------------------------------------------------------------
| @output
+-----------------------------------------------------------------------------------------------------------------------
| {"text": "To deploy a trained machine learning model for predicting food delivery duration, you need to l
|            \n\nFirst, you need to train the model using the ML_TRAIN routine and get the model handle.\n\
|            \n```\n\nThen, you can load the model into HeatWave AutoML using the ML_MODEL_LOAD routine.\n\
|            \n```\n\nAfter loading the model, you can use it to make predictions using the ML_PREDICT_TABL
|            \n\nFor example, to predict the delivery duration for a new food delivery order, you can use t
|            \n\n```mysql\nSELECT sys.ML_PREDICT_ROW(@new_order, @food_delivery_model);
|            \n```\n\nReplace `@new_order` with the actual data for the new order.\n\nNote: Make sure to re
|            \n\nAlso, ensure that the model is loaded into HeatWave AutoML before making predictions. You
|            \n\nIf you need to unload the model, you can use the ML_MODEL_UNLOAD routine.\n\n```mysql\nCAL
+-----------------------------------------------------------------------------------------------------------------------
1 row in set (0.0002 sec)
mysql> CALL sys.NL2ML("how can I evaluate this model?", @output);
Query OK, 0 rows affected (16.4552 sec)

mysql> SELECT @output;
+-----------------------------------------------------------------------------------------------------------------------
| @output
+-----------------------------------------------------------------------------------------------------------------------
| {"text": "To evaluate a trained machine learning model for predicting food delivery duration, you can use
|            \n\nHere's an example SQL query to evaluate the model:\n\n```sql\nCALL sys.ML_SCORE('mlcorpus.
|            \n```\n\nIn this query:\n\n- 'mlcorpus.food_delivery_validate' is the table used for validatio
|            \n- 'delivery_duration' is the target column containing the actual delivery durations.\n- @del
|            \n- 'neg_mean_absolute_error' is the metric used to evaluate the model.\n- @score is the user-
|            \n\nNote that you should replace '@delivery_duration_model' with the actual model handle or se
+-----------------------------------------------------------------------------------------------------------------------
1 row in set (0.0002 sec)
mysql> CALL sys.NL2ML("How can I call predictions for a new row??", @output);
Query OK, 0 rows affected (16.8089 sec)

mysql> SELECT @output;
+-----------------------------------------------------------------------------------------------------------------------
| @output
+-----------------------------------------------------------------------------------------------------------------------
| {"text": "To predict the delivery duration for a new food delivery order, you can use the ML_PREDICT_ROW
|            \n\nAssuming that your model handle is stored in a session variable @delivery_duration_model,
|            \n\n```sql\nSET @new_order = JSON_OBJECT(\n  \"order_id\", \"new_order_id\",\n  \"customer_id\
|            \n  \"food_item\", \"new_food_item\",\n  \"order_time\", \"new_order_time\",\n  \"delivery_tim
|            \n  \"delivery_distance\", \"new_delivery_distance\",\n  \"delivery_method\", \"new_delivery_m
|            \n  \"weather_conditions\", \"new_weather_conditions\",\n  \"order_priority\", \"new_order_pri
```

```
|           \n\nSELECT sys.ML_PREDICT_ROW(@new_order, @delivery_duration_model, NULL);\n```\n\nReplace
|           \n\nNote that the delivery_time is set to NULL, assuming that it is the target variable th
+-----------------------------------------------------------------------------------------------------
1 row in set (0.0003 sec)
mysql> CALL sys.NL2ML("How can I generate explanation for a new table?", @output);
Query OK, 0 rows affected (8.7374 sec)

mysql> SELECT @output;
+-----------------------------------------------------------------------------------------------------
| @output
+-----------------------------------------------------------------------------------------------------
| {"text": "```sql\nCALL sys.ML_EXPLAIN_TABLE('mlcorpus.new_food_delivery', @food_delivery_model, 'mlco
+-----------------------------------------------------------------------------------------------------
1 row in set (0.0003 sec)
```

The following use case has non-descriptive table columns and shows how NL2ML can scan column comments to retrieve the contextual information needed to prepare queries.

Refer to the following questions and responses. The output for each question is similar to the following:

```
mysql> SHOW TABLES;
+----------------------------+
| Tables_in_food_delivery_db |
+----------------------------+
| table_1                    |
| table_10                   |
| table_2                    |
| table_3                    |
| table_4                    |
| table_5                    |
| table_6                    |
| table_7                    |
| table_8                    |
| table_9                    |
+----------------------------+
10 rows in set (0.001 sec)

mysql> DESC table_3;
+-------+---------------------------------+------+-----+---------+----------------+
| Field | Type                            | Null | Key | Default | Extra          |
+-------+---------------------------------+------+-----+---------+----------------+
| x0    | int                             | NO   | PRI | NULL    | auto_increment |
| x1    | varchar(255)                    | NO   |     | NULL    |                |
| x2    | varchar(20)                     | YES  | UNI | NULL    |                |
| x3    | enum('bike','motorbike','car') | YES  |     | NULL    |                |
| x4    | decimal(9,6)                    | YES  |     | NULL    |                |
| x5    | decimal(9,6)                    | YES  |     | NULL    |                |
| x6    | decimal(2,1)                    | YES  |     | NULL    |                |
| x7    | tinyint(1)                      | YES  |     | 1       |                |
+-------+---------------------------------+------+-----+---------+----------------+
8 rows in set (0.001 sec)

mysql> SELECT COLUMN_COMMENT FROM INFORMATION_SCHEMA.COLUMNS WHERE TABLE_SCHEMA = 'food_delivery_db' AN
+---------------------------+
| COLUMN_COMMENT            |
+---------------------------+
| Rider id                  |
| Rider name                |
| Rider phone number        |
| Rider vehicle type        |
| Current latitude          |
| Current logitude          |
| rating                    |
| Whether rider is available |
+---------------------------+
8 rows in set (0.001 sec)

mysql> SET @input = 'Give me a training query to predict how long it takes to deliver food';
Query OK, 0 rows affected (0.000 sec)

mysql> CALL sys.NL2ML(@input, @output);
Query OK, 0 rows affected (1 min 35.274 sec)
```

```
mysql> SELECT @output;
+--------------------------------------------------------------------------------------------------------------------------------
| @output
+--------------------------------------------------------------------------------------------------------------------------------
| {"text": "**Data Preparation:**\n\n1. Integrate the tables into one full table using a SQL query.
|                              \n\n```sql\nCREATE TABLE integrated_table AS\nSELECT \n  t5.*,\n  t1.x1 AS r
|                              \n  t1.x3 AS restaurant_latitude,\n  t1.x4 AS restaurant_longitude,\n  t1.x5
|                              \n  t2.x1 AS customer_name,\n  t2.x2 AS customer_email,\n  t2.x3 AS customer
|                              \n  t2.x5 AS customer_longitude,\n  t3.x1 AS rider_name,\n  t3.x2 AS rider_p
|                              \n  t3.x4 AS rider_current_latitude,\n  t3.x5 AS rider_current_longitude,\n
|                              \n  t4.x1 AS food_item_name,\n  t4.x2 AS food_category,\n  t4.x3 AS food_ite
|                              \n  t6.x1 AS order_item_quantity,\n  t6.x2 AS order_item_price,\n  t7.x1 AS
|                              \n  t7.x5 AS log_longitude,\n  t8.x1 AS weather_temperature,\n  t8.x2 AS wea
|                              \n  t8.x4 AS weather_traffic_level,\n  t9.x1 AS origin_latitude,\n  t9.x2 AS
|                              \n  t9.x4 AS destination_longitude,\n  t9.x5 AS distance_in_kilometers,\n  t
|                              \n  t10.x2 AS shift_end_time,\n  t10.x3 AS total_orders_in_shift,\n  t10.x4
|                              \n  LEFT JOIN table_1 t1 ON t5.table_1_id = t1.x0\n  LEFT JOIN table_2 t2 ON
|                              \n  LEFT JOIN table_4 t4 ON t5.table_4_id = t4.x0\n  LEFT JOIN table_6 t6 ON
|                              \n  LEFT JOIN table_8 t8 ON t5.x0 = t8.x0\n  LEFT JOIN table_9 t9 ON t5.x0 =
|                              \n```\n\n2. Identify columns that should be excluded from training and provi
|                              \nJSON_ARRAY('x0', 'table_1_id', 'table_2_id', 'table_3_id', 'table_4_id', '
|                              \n```\n\n3. Use the available utility SQL stored procedure `sys.TRAIN_TEST_S
|                              \nCALL sys.TRAIN_TEST_SPLIT('food_delivery_db.integrated_table', NULL, JSON_
|                              \n\n**Training Query:**\n\nTo predict how long it takes to deliver food, we
|                              \n\n```sql\nCALL sys.ML_TRAIN('food_delivery_db.integrated_table_train', 'es
|                              \n             'exclude_column_list', JSON_ARRAY('x0', 'table_1_id', 'table
|                              \n             @food_delivery_model);\n```"}
+--------------------------------------------------------------------------------------------------------------------------------
1 row in set (0.000 sec)

mysql> SET @input = 'How can I evaluate this model?';
Query OK, 0 rows affected (0.000 sec)

mysql> CALL sys.NL2ML(@input, @output);
Query OK, 0 rows affected (15.196 sec)

mysql> SELECT @output;
+--------------------------------------------------------------------------------------------------------------------------------
| @output
+--------------------------------------------------------------------------------------------------------------------------------
| {"text": "To evaluate the regression model trained using the sys.ML_TRAIN procedure, you can use the sys.
|          \n\n```sql\nCALL sys.ML_SCORE('food_delivery_db.integrated_table_test', 'estimated_travel_durati
|          \n\nSELECT @score;\n```\n\nThis will return the mean squared error of the model on the test data
|          \n\nYou can replace 'neg_mean_squared_error' with other regression metrics available in the sys.
|          \n\nFor example, to evaluate the model using R-squared metric:\n\n```sql\nCALL sys.ML_SCORE('foo
|          \n\nSELECT @score;\n```\n\nThis will return the R-squared value of the model on the test data."}
+--------------------------------------------------------------------------------------------------------------------------------
1 row in set (0.000 sec)

mysql> SET @input = 'How can I call predictions for a new row?';
Query OK, 0 rows affected (0.000 sec)

mysql> CALL sys.NL2ML(@input, @output);
Query OK, 0 rows affected (15.651 sec)

mysql> SELECT @output;
+--------------------------------------------------------------------------------------------------------------------------------
| @output
+--------------------------------------------------------------------------------------------------------------------------------
| {"text": "To make predictions for a new row of data using the trained model, you can use the `sys.ML_PRED
|          \n\n```sql\nSET @new_row = JSON_OBJECT(\n  \"estimated_travel_duration\", NULL,  # This will be
|          \n  \"origin_latitude\", 40.7128,\n  \"origin_longitude\", -74.0060,\n  \"destination_latitude\"
|          \n  # Add more column values as needed\n);\n\nSELECT sys.ML_PREDICT_ROW(@new_row, @food_delivery
|          \n```\n\nReplace the `# Add your new row's column values here` comment with the actual column va
|          \n\nThis will return a JSON object containing the predicted value for `estimated_travel_duration
+--------------------------------------------------------------------------------------------------------------------------------
1 row in set (0.000 sec)

mysql> SET @input = 'How can I generate explanation for a new table?';
Query OK, 0 rows affected (0.000 sec)
```

```
mysql> CALL sys.NL2ML(@input, @output);
Query OK, 0 rows affected (22.206 sec)

mysql> SELECT @output;
+-----------------------------------------------------------------------------------------------------------------
| @output
+-----------------------------------------------------------------------------------------------------------------
| {"text": "To generate explanations for predictions made on a new table using the trained food deliver
|          \n\nFirst, make sure to load the model using the `sys.ML_MODEL_LOAD` procedure:\n\n```sql\nC
|          \n```\n\nThen, create a new table with the data you want to generate explanations for. This
|          \n\nLet's assume the new table is named `new_table` in the `food_delivery_db` schema. \n\nYo
|          \n\n```sql\nCALL sys.ML_EXPLAIN_TABLE('food_delivery_db.new_table', \n
|          \n                          JSON_OBJECT('prediction_explainer', 'permutation_importance'));\
|          \n\nYou can replace `'permutation_importance'` with `'shap'` to use the SHAP prediction expl
|          \n\n```sql\nSELECT * FROM food_delivery_db.new_table_explanations;\n```"} |
+-----------------------------------------------------------------------------------------------------------------
1 row in set (0.000 sec)
```

What's Next

- Review Machine Learning Use Cases.
- Review the syntax and examples for the NL2ML routine.

6.7 Machine Learning Use Cases

6.7.1 Classify Data

Classification models predict the discrete value of input data to specific predefined categories. Some examples of classification include loan approvals, churn prediction, and spam detection.

The following tasks use a dataset generated by OCI GenAI using Meta Llama Models. The classification use-case is to approve or reject loan applications for clients based on their personal and socioeconomic status, assets, liabilities, credit rating, and past loan details.

To generate your own datasets to create machine learning models in MySQL HeatWave, learn how to Generate Text-Based Content.

Note

Datasets were generated using Meta Llama models. Your use of this Llama model is subject to your Oracle agreements and this Llama license agreement: https://downloads.mysql.com/docs/LLAMA_31_8B_INSTRUCT-license.pdf.

6.7.1.1 Prepare Data for a Classification Model

This topic describes how to prepare the data to use for a classification machine learning model. It uses a data sample generated by OCI GenAI. The classification use-case is to approve or reject loan applications for clients based on their personal and socioeconomic status, assets, liabilities, credit rating, and past loan details. To prepare the data for this use case, you set up a training dataset and a testing dataset. The training dataset has 20 records, and the testing dataset has 10 records. In a real-life use case, you should prepare a larger amount of records for training and testing, and ensure the predictions are valid and reliable before testing on unlabeled data. To ensure reliable predictions, you should create an additional validation dataset. You can reserve 20% of the records in the training dataset to create the validation dataset.

You have the option to automatically Prepare Training and Testing Datasets with your own data by using the `TRAIN_TEST_SPLIT` routine.

Before You Begin

- Learn how to Prepare Data.

Prepare Data

To prepare the data for the classification model:

1. Connect to your MySQL HeatWave Database System.
2. Create and use the database to store the data.

```
mysql> CREATE DATABASE classification_data;
mysql> USE classification_data;
```

3. Create the table to insert the sample data into. This is the training dataset.

```
mysql> CREATE TABLE Loan_Training (
    ClientID INT PRIMARY KEY,
    ClientAge INT NOT NULL,
    Gender VARCHAR(10) NOT NULL,
    Education VARCHAR(50) NOT NULL,
    Occupation VARCHAR(50) NOT NULL,
    Income REAL NOT NULL,
    Debt REAL NOT NULL,
    CreditScore INT NOT NULL,
    Assets REAL NOT NULL,
    Liabilities REAL NOT NULL,
    LoanType VARCHAR(20) NOT NULL,
    LoanAmount REAL NOT NULL,
    Approved VARCHAR(10) NOT NULL
);
```

4. Insert the sample data into the table. Copy and paste the following commands.

```
INSERT INTO Loan_Training (ClientID, ClientAge, Gender, Education, Occupation, Income, Debt, CreditScor
(101, 30, 'Male', 'Bachelor''s', 'Engineer', 75000, 15000, 700, 300000, 80000, 'Home', 250000, 'Approve
(102, 25, 'Female', 'Master''s', 'Analyst', 60000, 10000, 680, 200000, 50000, 'Personal', 120000, 'Reje
(103, 40, 'Male', 'High School', 'Manager', 80000, 20000, 650, 450000, 120000, 'Business', 150000, 'App
(104, 35, 'Female', 'PhD', 'Doctor', 120000, 30000, 750, 600000, 250000, 'Car', 30000, 'Approved'),
(105, 28, 'Male', 'College', 'IT Specialist', 55000, 8000, 620, 280000, 90000, 'Education', 80000, 'Rej
(106, 45, 'Female', 'Bachelor''s', 'Teacher', 70000, 15000, 720, 500000, 180000, 'Home', 200000, 'Appro
(107, 32, 'Male', 'Associate', 'Sales', 65000, 12000, 670, 350000, 100000, 'Vacation', 18000, 'Rejected
(108, 22, 'Female', 'College', 'Student', 30000, 5000, 660, 150000, 40000, 'Education', 10000, 'Approve
(109, 50, 'Male', 'Master''s', 'Lawyer', 110000, 40000, 780, 700000, 350000, 'Investment', 500000, 'Rej
(110, 38, 'Female', 'High School', 'Nurse', 52000, 18000, 640, 220000, 120000, 'Medical', 35000, 'Appro
(111, 48, 'Male', 'Diploma', 'Plumber', 48000, 10000, 600, 180000, 70000, 'Home Improvement', 25000, 'R
(112, 55, 'Female', 'Bachelor''s', 'Writer', 90000, 25000, 760, 400000, 200000, 'Retirement', 150000, '
(113, 36, 'Male', 'Master''s', 'Accountant', 78000, 22000, 740, 380000, 150000, 'Refinance', 200000, 'Ap
(114, 24, 'Female', 'College', 'Designer', 45000, 7000, 610, 250000, 100000, 'Startup', 50000, 'Rejecte
(115, 42, 'Male', 'PhD', 'Scientist', 130000, 50000, 800, 550000, 300000, 'Research', 400000, 'Approved
(116, 52, 'Female', 'Master''s', 'Marketer', 85000, 35000, 770, 480000, 280000, 'Marketing', 120000, 'R
(117, 34, 'Male', 'Bachelor''s', 'Programmer', 68000, 16000, 690, 320000, 110000, 'Equipment', 85000, '
(118, 26, 'Female', 'Associate', 'Retail', 42000, 6000, 630, 200000, 70000, 'Wedding', 28000, 'Rejected
(119, 46, 'Male', 'College', 'Pilot', 100000, 45000, 710, 520000, 250000, 'Boat', 350000, 'Approved'),
(120, 58, 'Female', 'PhD', 'Professor', 140000, 60000, 820, 650000, 450000, 'Real Estate', 550000, 'Rej
```

5. Create the table to use for generating predictions and explanations. This is the test dataset. It has the same columns as the training dataset, but the target column, `Approved`, is not considered when generating predictions or explanations.

```
mysql> CREATE TABLE Loan_Testing (
    ClientID INT PRIMARY KEY,
    ClientAge INT NOT NULL,
    Gender VARCHAR(10) NOT NULL,
    Education VARCHAR(50) NOT NULL,
    Occupation VARCHAR(50) NOT NULL,
    Income REAL NOT NULL,
    Debt REAL NOT NULL,
    CreditScore INT NOT NULL,
    Assets REAL NOT NULL,
    Liabilities REAL NOT NULL,
    LoanType VARCHAR(20) NOT NULL,
    LoanAmount REAL NOT NULL,
    Approved VARCHAR(10) NOT NULL
```

```
);
```

6. Insert the sample data into the table. Copy and paste the following commands.

```
INSERT INTO Loan_Testing (ClientID, ClientAge, Gender, Education, Occupation, Income, Debt, CreditSc
(201, 38, 'Male', 'College', 'Architect', 62000, 18000, 660, 380000, 160000, 'Home', 280000, 'Approv
(202, 29, 'Female', 'Master''s', 'HR Manager', 58000, 12000, 690, 260000, 110000, 'Personal', 150000
(203, 44, 'Male', 'Bachelor''s', 'Chef', 72000, 25000, 730, 420000, 200000, 'Business', 180000, 'App
(204, 56, 'Female', 'PhD', 'Psychologist', 105000, 35000, 790, 580000, 320000, 'Car', 40000, 'Reject
(205, 31, 'Male', 'High School', 'Carpenter', 50000, 8000, 610, 240000, 85000, 'Education', 90000,
(206, 27, 'Female', 'College', 'Artist', 48000, 7000, 640, 220000, 95000, 'Art', 150000, 'Rejected'
(207, 49, 'Male', 'Associate', 'Electrician', 55000, 15000, 670, 300000, 120000, 'Home Improvement',
(208, 53, 'Female', 'Bachelor''s', 'Journalist', 88000, 30000, 750, 460000, 280000, 'Travel', 180000
(209, 37, 'Male', 'Master''s', 'Financial Advisor', 76000, 22000, 700, 360000, 150000, 'Investment',
(210, 23, 'Female', 'College', 'Intern', 35000, 5000, 600, 160000, 60000, 'Education', 20000, 'Rejec
```

What's Next

- Learn how to Train a Classification Model.

6.7.1.2 Train a Classification Model

After preparing the data for a classification model, you can train the model.

Before You Begin

- Review and complete all the tasks to Prepare Data for a Classification Model.

Train Model

Train the model with the `ML_TRAIN` routine and use the `training_data` table previously created. Before training the model, it is good practice to define the model handle instead of automatically creating one. See Define Model Handle.

1. Optionally, set the value of the session variable, which sets the model handle to this same value.

```
mysql> SET @variable = 'model_handle';
```

Replace *`@variable`* and *`model_handle`* with your own definitions. For example:

```
mysql> SET @model='classification_use_case';
```

The model handle is set to `classification_use_case`.

2. Run the `ML_TRAIN` routine.

```
mysql> CALL sys.ML_TRAIN('table_name', 'target_column_name', JSON_OBJECT('task', 'task_name'), model
```

Replace *`table_name`*, *`target_column_name`*, *`task_name`*, and *`model_handle`* with your own values.

The following example runs `ML_TRAIN` on the training dataset previously created.

```
mysql> CALL sys.ML_TRAIN('classification_data.Loan_Training', 'Approved', JSON_OBJECT('task', 'class
```

Where:

- `classification_data.Loan_Training` is the fully qualified name of the table that contains the training dataset (*`database_name.table_name`*).
- `Approved` is the name of the target column, which contains ground truth values.
- `JSON_OBJECT('task', 'classification')` specifies the machine learning task type.
- `@model` is the session variable previously set that defines the model handle to the name defined by the user: `classification_use_case`. If you do not define the model handle before training

the model, the model handle is automatically generated, and the session variable only stores the model handle for the duration of the connection. User variables are written as *@var_name*. Any valid name for a user-defined variable is permitted. See Work with Model Handles to learn more.

3. When the training operation finishes, the model handle is assigned to the `@model` session variable, and the model is stored in the model catalog. View the entry in the model catalog with the following query. Replace *user1* with your MySQL account name.

```
mysql> SELECT model_id, model_handle, train_table_name FROM ML_SCHEMA_user1.MODEL_CATALOG WHERE model_ha
+----------+--------------------------------------------+-------------------------------------+
| model_id | model_handle                               | train_table_name                    |
+----------+--------------------------------------------+-------------------------------------+
|        1 | classification_use_case                    | classification_data.Loan_Training   |
+----------+--------------------------------------------+-------------------------------------+
```

What's Next

- Learn how to Generate Predictions for a Classification Model.

6.7.1.3 Generate Predictions for a Classification Model

After training the model, you can generate predictions.

To generate predictions, use the sample data from the `testing_data` dataset. Even though the table has labels for the `Approved` target column, the column is not considered when generating predictions. This allows you to compare the predictions to the actual values in the dataset and determine if the predictions are reliable. Once you determine the trained model is reliable for generating predictions, you can start using unlabeled datasets for generating predictions.

Before You Begin

Complete the following tasks:

- Prepare Data for a Classification Model
- Train a Classification Model

Generate Predictions for a Table

1. If not already done, load the model. You can use the session variable for the model that is valid for the duration of the connection. Alternatively, you can use the model handle previously set. For the option to set the user name, you can set it to `NULL`.

   The following example uses the session variable.

   ```
   mysql> CALL sys.ML_MODEL_LOAD(@model, NULL);
   ```

   The following example uses the model handle.

   ```
   mysql> CALL sys.ML_MODEL_LOAD('classification_use_case', NULL);
   ```

2. Make predictions for the test dataset by using the `ML_PREDICT_TABLE` routine.

   ```
   mysql> CALL sys.ML_PREDICT_TABLE(table_name, model_handle, output_table_name), [options]);
   ```

   Replace *table_name*, *model_handle*, and *output_table_name* with your own values. Add *options* as needed.

   As of MySQL 9.4.1, you have the option to specify the input table and output table as the same table if specific conditions are met. See Input Tables and Output Tables to learn more.

   The following example runs `ML_PREDICT_TABLE` on the testing dataset previously created.

   ```
   mysql> CALL sys.ML_PREDICT_TABLE('classification_data.Loan_Testing', @model, 'classification_data.Loan_
   ```

Where:

- `classification_data.Loan_Testing` is the fully qualified name of the input table that contains the data to generate predictions for (*`database_name.table_name`*).
- `@model` is the session variable for the model handle.
- `classification_data.Loan_Testing_predictions` is the fully qualified name of the output table with predictions (*`database_name.table_name`*).
- `NULL` sets no options for the routine.

3. Query the `Approved`, `Prediction`, and `ml_results` columns from the output table. This allows you to compare the real value with the generated prediction. You can also review the probabilities for each prediction. If needed, you can also query all the columns from the table (`SELECT * FROM classification_predictions`) to review all the data at once.

```
mysql> SELECT Approved, Prediction, ml_results FROM Loan_Testing_predictions;
+----------+------------+-------------------------------------------------------------------------------
| Approved | Prediction | ml_results
+----------+------------+-------------------------------------------------------------------------------
| Approved | Approved   | {"predictions": {"Approved": "Approved"}, "probabilities": {"Approved": 0.
| Rejected | Rejected   | {"predictions": {"Approved": "Rejected"}, "probabilities": {"Approved": 0.
| Approved | Approved   | {"predictions": {"Approved": "Approved"}, "probabilities": {"Approved": 0.
| Rejected | Rejected   | {"predictions": {"Approved": "Rejected"}, "probabilities": {"Approved": 0.
| Approved | Rejected   | {"predictions": {"Approved": "Rejected"}, "probabilities": {"Approved": 0.
| Rejected | Rejected   | {"predictions": {"Approved": "Rejected"}, "probabilities": {"Approved": 0.
| Approved | Approved   | {"predictions": {"Approved": "Approved"}, "probabilities": {"Approved": 0.
| Rejected | Rejected   | {"predictions": {"Approved": "Rejected"}, "probabilities": {"Approved": 0.
| Approved | Approved   | {"predictions": {"Approved": "Approved"}, "probabilities": {"Approved": 0.
| Rejected | Approved   | {"predictions": {"Approved": "Approved"}, "probabilities": {"Approved": 0.
+----------+------------+-------------------------------------------------------------------------------
10 rows in set (0.0430 sec)
```

The results show that two predictions do not match up with the real values.

To learn more about generating predictions for one or more rows of data, see Generate Predictions for a Row of Data.

What's Next

- Learn now to Query Model Explanation and Generate Prediction Explanations for a Classification Model.

6.7.1.4 Query Model Explanation and Generate Prediction Explanations for a Classification Model

After training a classification model, you can query the default model explanation or query new model explanations. You can also generate prediction explanations. Explanations help you understand which features had the most influence on generating predictions.

Feature importance is presented as an attribution value. A positive value indicates that a feature contributed toward the prediction. A negative value can have different interpretations depending on the specific model explainer used for the model. For example, a negative value for the permutation importance explainer means that the feature is not important.

Before You Begin

Complete the following tasks:

- Prepare Data for a Classification Model
- Train a Classification Model

- Generate Predictions for a Classification Model

Generate the Model Explanation

After training a model, you can query the default model explanation with the Premutation Importance explainer.

To generate explanations for other model explainers, see Generate Model Explanations and ML_EXPLAIN.

Query the `model_explanation` column from the model catalog and define the model handle previously created. Update `user1` with your own user name. Use `JSON_PRETTY` to view the output in an easily readable format.

```
mysql> SELECT JSON_PRETTY(model_explanation) FROM ML_SCHEMA_user1.MODEL_CATALOG
        WHERE model_handle='classification_use_case';
+-----------------------------------------------------------------------------------------------------+
| JSON_PRETTY(model_explanation)                                                                      |
+-----------------------------------------------------------------------------------------------------+
| {
  "permutation_importance": {
    "Debt": 0.5014,
    "Assets": 0.0,
    "Gender": 0.0,
    "Income": 0.0,
    "ClientID": 0.0,
    "LoanType": 0.0,
    "ClientAge": 0.1231,
    "Education": 0.0,
    "LoanAmount": 0.0,
    "Occupation": 0.0,
    "CreditScore": 0.0,
    "Liabilities": 0.0525
  }
} |
+-----------------------------------------------------------------------------------------------------+
1 row in set (0.0382 sec)
```

Feature importance values display for each column.

Generate Prediction Explanations for a Table

After training a model, you can generate a table of prediction explanations on the `testing_data` dataset by using the default Permutation Importance prediction explainer.

To generate explanations for other model explainers, see Generate Prediction Explanations and ML_EXPLAIN_TABLE.

1. If not already done, load the model. You can use the session variable for the model that is valid for the duration of the connection. Alternatively, you can use the model handle previously set. For the option to set the user name, you can set it to `NULL`.

   The following example uses the session variable.

   ```
   mysql> CALL sys.ML_MODEL_LOAD(@model, NULL);
   ```

   The following example uses the model handle.

   ```
   mysql> CALL sys.ML_MODEL_LOAD('classification_use_case', NULL);
   ```

2. Use the `ML_EXPLAIN_TABLE` routine to generate explanations for predictions made in the test dataset.

   ```
   mysql> CALL sys.ML_EXPLAIN_TABLE(table_name, model_handle, output_table_name, [options]);
   ```

   Replace *`table_name`*, *`model_handle`*, and *`output_table_name`* with your own values. Add *`options`* as needed.

As of MySQL 9.4.1, you have the option to specify the input table and output table as the same table if specific conditions are met. See Input Tables and Output Tables to learn more.

The following example runs `ML_EXPLAIN_TABLE` on the testing dataset previously created.

```
mysql> CALL sys.ML_EXPLAIN_TABLE('classification_data.Loan_Testing', @model, 'classification_data.Lo
                                  JSON_OBJECT('prediction_explainer', 'permutation_importance'));
Query OK, 0 rows affected (12.2957 sec)
```

Where:

- `classification_data.Loan_Testing` is the fully qualified name of the test dataset.
- `@model` is the session variable for the model handle.
- `classification_data.Loan_Testing_explanations` is the fully qualified name of the output table with explanations.
- `permutation_importance` is the selected prediction explainer to use to generate explanations.

3. Query `Notes` and `ml_results` from the output table to review which column contributed the most against or had the largest impact towards the prediction. You can also review individual attribution values for each column. Use `\G` to view the output in an easily readable format.

```
mysql> SELECT Notes, ml_results FROM Loan_Testing_explanations\G
*************************** 1. row ***************************
     Notes: Debt (18000.0) had the largest impact towards predicting Approved
ml_results: {"attributions": {"Debt": 0.87, "Liabilities": -0.0, "ClientAge": 0.0, "LoanAmount": 0.0
             "predictions": {"Approved": "Approved"}, "notes": "Debt (18000.0) had the largest impac
*************************** 2. row ***************************
     Notes: ClientAge (29) had the largest impact towards predicting Rejected, whereas Debt (12000.0
ml_results: {"attributions": {"Debt": -0.01, "Liabilities": 0.02, "ClientAge": 0.17, "LoanAmount": 0
             "predictions": {"Approved": "Rejected"}, "notes": "ClientAge (29) had the largest impac
*************************** 3. row ***************************
     Notes: Debt (25000.0) had the largest impact towards predicting Approved
ml_results: {"attributions": {"Debt": 0.87, "Liabilities": -0.0, "ClientAge": 0.0, "LoanAmount": 0.0
             "predictions": {"Approved": "Approved"}, "notes": "Debt (25000.0) had the largest impac
*************************** 4. row ***************************
     Notes: ClientAge (56) had the largest impact towards predicting Rejected, whereas Debt (35000.0
ml_results: {"attributions": {"Debt": -0.07, "Liabilities": 0.52, "ClientAge": 0.75, "LoanAmount": 0
             "predictions": {"Approved": "Rejected"}, "notes": "ClientAge (56) had the largest impac
*************************** 5. row ***************************
     Notes: LoanAmount (90000.0) had the largest impact towards predicting Rejected
ml_results: {"attributions": {"Debt": 0.0, "Liabilities": 0.01, "ClientAge": 0.1, "LoanAmount": 0.14
             "predictions": {"Approved": "Rejected"}, "notes": "LoanAmount (90000.0) had the largest
*************************** 6. row ***************************
     Notes: ClientAge (27) had the largest impact towards predicting Rejected
ml_results: {"attributions": {"Debt": -0.0, "Liabilities": 0.01, "ClientAge": 0.16, "LoanAmount": 0.
             "predictions": {"Approved": "Rejected"}, "notes": "ClientAge (27) had the largest impac
*************************** 7. row ***************************
     Notes: Debt (15000.0) had the largest impact towards predicting Approved, whereas ClientAge (49
ml_results: {"attributions": {"Debt": 0.49, "Liabilities": -0.07, "ClientAge": -0.43, "LoanAmount":
             "predictions": {"Approved": "Approved"}, "notes": "Debt (15000.0) had the largest impac
*************************** 8. row ***************************
     Notes: ClientAge (53) had the largest impact towards predicting Rejected, whereas Debt (30000.0
ml_results: {"attributions": {"Debt": -0.13, "Liabilities": 0.56, "ClientAge": 0.68, "LoanAmount": -
             "predictions": {"Approved": "Rejected"}, "notes": "ClientAge (53) had the largest impac
*************************** 9. row ***************************
     Notes: Debt (22000.0) had the largest impact towards predicting Approved
ml_results: {"attributions": {"Debt": 0.87, "Liabilities": -0.0, "ClientAge": 0.0, "LoanAmount": 0.0
             "predictions": {"Approved": "Approved"}, "notes": "Debt (22000.0) had the largest impac
*************************** 10. row ***************************
     Notes: No features had a significant impact on model prediction
ml_results: {"attributions": {"Debt": 0.0, "Liabilities": 0.0, "ClientAge": 0.0, "LoanAmount": 0.0},
             "predictions": {"Approved": "Approved"}, "notes": "No features had a significant impact
10 rows in set (0.0461 sec)
```

To generate prediction explanations for one or more rows of data, see Generate Prediction Explanations for a Row of Data.

What's Next

- Learn how to Score a Classification Model.

6.7.1.5 Score a Classification Model

After generating predictions and explanations, you can score the model to assess its reliability. For a list of scoring metrics you can use with classification models, see Classification Metrics. For this use case, you use the test dataset for validation. In a real-world use case, you should use a separate validation dataset that has the target column and ground truth values for the scoring validation. You should also use a larger number of records for training and validation to get a valid score.

Before You Begin

Complete the following tasks:

- Prepare Data for a Classification Model
- Train a Classification Model
- Generate Predictions for a Classification Model
- Query Model Explanation and Generate Prediction Explanations for a Classification Model

Score Model

1. If not already done, load the model. You can use the session variable for the model that is valid for the duration of the connection. Alternatively, you can use the model handle previously set. For the option to set the user name, you can set it to `NULL`.

   The following example uses the session variable.

   ```
   mysql> CALL sys.ML_MODEL_LOAD(@model, NULL);
   ```

   The following example uses the model handle.

   ```
   mysql> CALL sys.ML_MODEL_LOAD('classification_use_case', NULL);
   ```

2. Score the model with the `ML_SCORE` routine and use the `accuracy` metric.

   ```
   mysql> CALL sys.ML_SCORE(table_name, target_column_name, model_handle, metric, score, [options]);
   ```

   Replace *`table_name`*, *`target_column_name`*, *`model_handle`*, *`metric`*, *`score`* with your own values.

   The following example runs `ML_SCORE` on the testing dataset previously created.

   ```
   mysql> CALL sys.ML_SCORE('classification_data.Loan_Testing', 'Approved', @model, 'accuracy', @classific
   ```

   Where:

   - `classification_data.Loan_Testing` is the fully qualified name of the validation dataset.
   - `Approved` is the target column name with ground truth values.
   - `@model` is the session variable for the model handle.
   - `accuracy` is the selected scoring metric.
   - `@classification_score` is the session variable name for the score value.

- `NULL` means that no other options are defined for the routine.

3. Retrieve the score by querying the @classification_score session variable.

```
mysql> SELECT @classification_score;
+----------------------+
| @classification_score |
+----------------------+
|     0.800000011920929 |
+----------------------+
1 row in set (0.0431 sec)
```

4. If done working with the model, unload it with the `ML_MODEL_UNLOAD` routine.

```
mysql> CALL sys.ML_MODEL_UNLOAD('classification_use_case');
```

To avoid consuming too much memory, it is good practice to unload a model when you are finished using it.

What's Next

- Review other Machine Learning Use Cases.

6.7.2 Perform Regression Analysis

Machine learning regression models generate predictions based on the relationship between a dependent variable and one or more independent variables. Some examples of regression analysis include predicting sales during different seasons, predicting purchasing behavior on a website based on the characteristics of website visitors, and predicting the sale price of residences based on their size.

The following tasks use a dataset generated by OCI GenAI using Meta Llama Models. The regression use-case is to predict house prices based on the size of the house, the address of the house, and the state the house is located in.

To generate your own datasets to create machine learning models in MySQL HeatWave, learn how to Generate Text-Based Content.

Note

Datasets were generated using Meta Llama models. Your use of this Llama model is subject to your Oracle agreements and this Llama license agreement: https://downloads.mysql.com/docs/LLAMA_31_8B_INSTRUCT-license.pdf.

6.7.2.1 Prepare Data for a Regression Model

This topic describes how to prepare the data to use for a regression machine learning model. It uses a data sample generated by OCI GenAI. The regression use-case is to predict house prices based on the size of the house, the address of the house, and the state the house is located in. To prepare the data for this use case, you set up a training dataset and a testing dataset. The training dataset has 20 records, and the testing dataset has 10 records. In a real-life use case, you should prepare a larger amount of records for training and testing, and ensure the predictions are valid and reliable before testing on unlabeled data. To ensure reliable predictions, you should create an additional validation dataset. You can reserve 20% of the records in the training dataset to create the validation dataset.

You have the option to automatically Prepare Training and Testing Datasets with your own data by using the `TRAIN_TEST_SPLIT` routine.

Before You Begin

- Learn how to Prepare Data.

Prepare Data

To prepare the data for the regression model:

1. Connect to your MySQL HeatWave Database System.
2. Create and use the database to store the data.

```
mysql> CREATE DATABASE regression_data;
mysql> USE regression_data;
```

3. Create the table to insert the sample data into. This is the training dataset.

```
mysql> CREATE TABLE house_price_training (
    id INT PRIMARY KEY,
    house_size INT,
    address TEXT,
    state TEXT,
    price INT
);
```

4. Insert the sample data into the table. Copy and paste the following commands.

```
INSERT INTO house_price_training (id, house_size, address, state, price)
VALUES
    (1, 1500, '123 Main St', 'California', 500000),
    (2, 2000, '456 Elm St', 'Texas', 650000),
    (3, 1800, '789 Oak Ave', 'New York', 700000),
    (4, 1200, '222 Pine Rd', 'Florida', 420000),
    (5, 1600, '555 Maple Lane', 'Washington', 550000),
    (6, 2500, '888 River Blvd', 'California', 800000),
    (7, 1300, '333 Creek St', 'Texas', 480000),
    (8, 1700, '666 Mountain Rd', 'Colorado', 520000),
    (9, 1400, '999 Valley View', 'New York', 580000),
    (10, 1900, '111 Ocean Blvd', 'Florida', 620000),
    (11, 1550, '2222 Lake Dr', 'Illinois', 540000),
    (12, 2100, '3333 Forest Ave', 'Texas', 750000),
    (13, 1650, '4444 Desert Rd', 'Arizona', 570000),
    (14, 1250, '5555 Riverbank St', 'Washington', 450000),
    (15, 1850, '6666 Sky Blvd', 'California', 720000),
    (16, 1350, '7777 Meadow Lane', 'Ohio', 490000),
    (17, 2050, '8888 Hill St', 'New York', 850000),
    (18, 1450, '9999 Creek Rd', 'Florida', 590000),
    (19, 1750, '10101 Ocean Ave', 'Texas', 680000),
    (20, 1580, '11111 Pine St', 'Illinois', 560000);
```

5. Create the table to use for generating predictions and explanations. This is the test dataset. It has the same columns as the training dataset, but the target column, `price`, is not considered when generating predictions or explanations.

```
mysql> CREATE TABLE house_price_testing (
    id INT PRIMARY KEY,
    house_size INT,
    address TEXT,
    state TEXT,
    price INT
);
```

6. Insert the sample data into the table. Copy and paste the following commands.

```
INSERT INTO house_price_testing (id, house_size, address, state, price)
VALUES
    (1, 1400, '500 Elm St', 'Nevada', 470000),
    (2, 1900, '200 River Rd', 'California', 630000),
    (3, 1600, '300 Mountain Ave', 'Colorado', 530000),
    (4, 2200, '400 Lake Blvd', 'New York', 780000),
    (5, 1300, '500 Creek Lane', 'Texas', 460000),
    (6, 1700, '600 Valley View Rd', 'Florida', 510000),
    (7, 1500, '700 Ocean St', 'Washington', 500000),
    (8, 1800, '800 Sky Blvd', 'Oregon', 600000),
```

```
    (9, 1200, '900 Meadow Ave', 'Illinois', 430000),
    (10, 2100, '1000 Hill Rd', 'New Jersey', 760000);
```

What's Next

- Learn how to Train a Regression Model.

6.7.2.2 Train a Model for Regression

After preparing the data for a regression model, you can train the model.

Before You Begin

- Review and complete all the tasks to Prepare Data for a Regression Model.

Train Model

Train the model with the `ML_TRAIN` routine and use the `house_price_training` table previously created. Before training the model, it is good practice to define the model handle instead of automatically creating one. See Define Model Handle.

1. Optionally, set the value of the session variable, which sets the model handle to this same value.

   ```
   mysql> SET @variable = 'model_handle';
   ```

   Replace *`@variable`* and *`model_handle`* with your own definitions. For example:

   ```
   mysql> SET @model='regression_use_case';
   ```

   The model handle is set to `regression_use_case`.

2. Run the `ML_TRAIN` routine.

   ```
   mysql> CALL sys.ML_TRAIN('table_name', 'target_column_name', JSON_OBJECT('task', 'task_name'), @vari
   ```

   Replace *`table_name`*, *`target_column_name`*, *`task_name`*, and *`variable`* with your own values.

   The following example runs `ML_TRAIN` on the training dataset previously created.

   ```
   mysql> CALL sys.ML_TRAIN('regression_data.house_price_training', 'price', JSON_OBJECT('task', 'regre
   ```

   Where:

   - `regression_data.house_price_training` is the fully qualified name of the table that contains the training dataset (*`database_name.table_name`*).
   - `price` is the name of the target column, which contains ground truth values.
   - `JSON_OBJECT('task', 'regression')` specifies the machine learning task type.
   - `@model` is the session variable previously set that defines the model handle to the name defined by the user: `regression_use_case`. If you do not define the model handle before training the model, the model handle is automatically generated, and the session variable only stores the model handle for the duration of the connection. User variables are written as *`@var_name`*. Any valid name for a user-defined variable is permitted. See Work with Model Handles to learn more.

3. When the training operation finishes, the model handle is assigned to the `@model` session variable, and the model is stored in the model catalog. View the entry in the model catalog with the following query. Replace *`user1`* with your MySQL account name.

   ```
   mysql> SELECT model_id, model_handle, train_table_name FROM ML_SCHEMA_user1.MODEL_CATALOG WHERE mode
   +----------+--------------------------------------------+--------------------------------------
   | model_id | model_handle                               | train_table_name                     
   +----------+--------------------------------------------+--------------------------------------
   ```

```
|        2 | regression_use_case                          | regression_data.house_price_training   |
+----------+----------------------------------------------+----------------------------------------+
```

What's Next

- Learn how to Generate Predictions for a Regression Model.

6.7.2.3 Generate Predictions for a Regression Model

After training the model, you can generate predictions.

To generate predictions, use the sample data from the `house_price_testing` dataset. Even though the table has labels for the `price` target column, the column is not considered when generating predictions. This allows you to compare the predictions to the actual values in the dataset and determine if the predictions are reliable. Once you determine the trained model is reliable for generating predictions, you can start using unlabeled datasets for generating predictions.

Before You Begin

Complete the following tasks:

- Prepare Data for a Regression Model
- Train a Regression Model

Generate Predictions for a Table

1. If not already done, load the model. You can use the session variable for the model that is valid for the duration of the connection. Alternatively, you can use the model handle previously set. For the option to set the user name, you can set it to `NULL`.

   The following example uses the session variable.

   ```
   mysql> CALL sys.ML_MODEL_LOAD(@model, NULL);
   ```

   The following example uses the model handle.

   ```
   mysql> CALL sys.ML_MODEL_LOAD('regression_use_case', NULL);
   ```

2. Make predictions for the test dataset by using the `ML_PREDICT_TABLE` routine.

   ```
   mysql> CALL sys.ML_PREDICT_TABLE(table_name, model_handle, output_table_name), [options]);
   ```

   Replace *`table_name`*, *`model_handle`*, and *`output_table_name`* with your own values. Add *`options`* as needed.

   As of MySQL 9.4.1, you have the option to specify the input table and output table as the same table if specific conditions are met. See Input Tables and Output Tables to learn more.

   The following example runs `ML_PREDICT_TABLE` on the testing dataset previously created.

   ```
   mysql> CALL sys.ML_PREDICT_TABLE('regression_data.house_price_testing', @model, 'regression_data.house_
   ```

   Where:

   - `regression_data.regression_testing` is the fully qualified name of the input table that contains the data to generate predictions for (*`database_name.table_name`*).
   - `@model` is the session variable for the model handle.
   - `regression_data.regression_predictions` is the fully qualified name of the output table with predictions (*`database_name.table_name`*).
   - `NULL` sets no options for the routine.

3. Query the `price`, `Prediction`, and `ml_results` columns from the output table. This allows you to compare the real value with the generated prediction. If needed, you can also query all the columns from the table (`SELECT * FROM regression_predictions`) to review all the data at once.

```
mysql> SELECT price, Prediction, ml_results FROM house_price_predictions;
+--------+------------+-------------------------------------+
| price  | Prediction | ml_results                          |
+--------+------------+-------------------------------------+
| 470000 |     490000 | {"predictions": {"price": 490000.0}} |
| 630000 |     660000 | {"predictions": {"price": 660000.0}} |
| 530000 |     552000 | {"predictions": {"price": 552000.0}} |
| 780000 |     810000 | {"predictions": {"price": 810000.0}} |
| 460000 |     478000 | {"predictions": {"price": 478000.0}} |
| 510000 |     594000 | {"predictions": {"price": 594000.0}} |
| 500000 |     518000 | {"predictions": {"price": 518000.0}} |
| 600000 |     700000 | {"predictions": {"price": 700000.0}} |
| 430000 |     426000 | {"predictions": {"price": 426000.0}} |
| 760000 |     790000 | {"predictions": {"price": 790000.0}} |
+--------+------------+-------------------------------------+
10 rows in set (0.0403 sec)
```

   Review the predictions and compare with the real prices.

To learn more about generating predictions for one or more rows of data, see Generate Predictions for a Row of Data.

What's Next

- Learn now to Query Model Explanation and Generate Prediction Explanations for a Regression Model.

6.7.2.4 Query Model Explanation and Generate Prediction Explanations for a Regression Model

After training a regression model, you can query the default model explanation or query new model explanations. You can also generate prediction explanations. Explanations help you understand which features had the most influence on generating predictions.

Feature importance is presented as an attribution value. A positive value indicates that a feature contributed toward the prediction. A negative value can have different interpretations depending on the specific model explainer used for the model. For example, a negative value for the permutation importance explainer means that the feature is not important.

Before You Begin

Complete the following tasks:

- Prepare Data for a Regression Model
- Train a Regression Model
- Generate Predictions for a Regression Model

Generate the Model Explanation

After training a model, you can query the default model explanation with the Premutation Importance explainer.

To generate explanations for other model explainers, see Generate Model Explanations and ML_EXPLAIN.

Query the `model_explanation` column from the model catalog and define the model handle previously created. Update `user1` with your own user name. Use `JSON_PRETTY` to view the output in an easily readable format.

```
mysql> SELECT JSON_PRETTY(model_explanation) FROM ML_SCHEMA_user1.MODEL_CATALOG
        WHERE model_handle='regression_use_case';
+-----------------------------------------------------------------------------------------------------------------------
| JSON_PRETTY(model_explanation)
+-----------------------------------------------------------------------------------------------------------------------
| {
  "permutation_importance": {
    "id": 0.0,
    "state": 0.0814,
    "address": 0.0,
    "house_size": 1.8123
  }
} |
+-----------------------------------------------------------------------------------------------------------------------
1 row in set (0.0530 sec)
```

Feature importance values display for each column.

Generate Prediction Explanations for a Table

After training a model, you can generate a table of prediction explanations on the `house_price_testing` dataset by using the default Permutation Importance prediction explainer.

To generate explanations for other model explainers, see Generate Prediction Explanations and ML_EXPLAIN_TABLE.

1. If not already done, load the model. You can use the session variable for the model that is valid for the duration of the connection. Alternatively, you can use the model handle previously set. For the option to set the user name, you can set it to `NULL`.

   The following example uses the session variable.

   ```
   mysql> CALL sys.ML_MODEL_LOAD(@model, NULL);
   ```

   The following example uses the model handle.

   ```
   mysql> CALL sys.ML_MODEL_LOAD('regression_use_case', NULL);
   ```

2. Use the `ML_EXPLAIN_TABLE` routine to generate explanations for predictions made in the test dataset.

   ```
   mysql> CALL sys.ML_EXPLAIN_TABLE(table_name, model_handle, output_table_name, [options]);
   ```

   Replace *`table_name`*, *`model_handle`*, and *`output_table_name`* with your own values. Add *`options`* as needed.

   As of MySQL 9.4.1, you have the option to specify the input table and output table as the same table if specific conditions are met. See Input Tables and Output Tables to learn more.

   The following example runs `ML_EXPLAIN_TABLE` on the testing dataset previously created.

   ```
   mysql> CALL sys.ML_EXPLAIN_TABLE('regression_data.house_price_testing', 'regression_use_case', 'regress
                                     JSON_OBJECT('prediction_explainer', 'permutation_importance'));
   Query OK, 0 rows affected (7.3776 sec)
   ```

   Where:

   - `regression_data.house_price_testing` is the fully qualified name of the test dataset.
   - `regression_use_case` is the model handle for the trained table.
   - `regression_data.regression_explanations` is the fully qualified name of the output table with explanations.
   - `permutation_importance` is the selected prediction explainer to use to generate explanations.

3. Query `Notes` and `ml_results` from the output table to review which column contributed the most against or had the largest impact towards the prediction. You can also review individual attribution values for each column. Use `\G` to view the output in an easily readable format.

```
mysql> SELECT Notes, ml_results FROM regression_data.regression_explanations\G
*************************** 1. row ***************************
     Notes: house_size (1400) increased the value the model predicted the most, whereas state (Nevad
ml_results: {"attributions": {"house_size": 64000.0, "state": -4053.33},
             "predictions": {"price": 490000.0}, "notes": "house_size (1400) increased the value the
*************************** 2. row ***************************
     Notes: house_size (1900) increased the value the model predicted the most, whereas state (Calif
ml_results: {"attributions": {"house_size": 234000.0, "state": -1760.0},
             "predictions": {"price": 660000.0}, "notes": "house_size (1900) increased the value the
*************************** 3. row ***************************
     Notes: house_size (1600) increased the value the model predicted the most, whereas state (Color
ml_results: {"attributions": {"house_size": 126000.0, "state": -1200.0},
             "predictions": {"price": 552000.0}, "notes": "house_size (1600) increased the value the
*************************** 4. row ***************************
     Notes: house_size (2200) increased the value the model predicted the most
ml_results: {"attributions": {"house_size": 320000.0, "state": 50000.0},
             "predictions": {"price": 810000.0}, "notes": "house_size (2200) increased the value the
*************************** 5. row ***************************
     Notes: house_size (1300) increased the value the model predicted the most, whereas state (Texas
ml_results: {"attributions": {"house_size": 52000.0, "state": -5826.67},
             "predictions": {"price": 478000.0}, "notes": "house_size (1300) increased the value the
*************************** 6. row ***************************
     Notes: house_size (1700) increased the value the model predicted the most, whereas state (Flori
ml_results: {"attributions": {"house_size": 168000.0, "state": -1813.33},
             "predictions": {"price": 594000.0}, "notes": "house_size (1700) increased the value the
*************************** 7. row ***************************
     Notes: house_size (1500) increased the value the model predicted the most, whereas state (Washi
ml_results: {"attributions": {"house_size": 92000.0, "state": -3626.67},
             "predictions": {"price": 518000.0}, "notes": "house_size (1500) increased the value the
*************************** 8. row ***************************
     Notes: house_size (1800) increased the value the model predicted the most, whereas state (Orego
ml_results: {"attributions": {"house_size": 274000.0, "state": -720.0},
             "predictions": {"price": 700000.0}, "notes": "house_size (1800) increased the value the
*************************** 9. row ***************************
     Notes: state (Illinois) reduced the value the model predicted the most
ml_results: {"attributions": {"house_size": 0.0, "state": -7680.0},
             "predictions": {"price": 426000.0}, "notes": "state (Illinois) reduced the value the mo
*************************** 10. row ***************************
     Notes: house_size (2100) increased the value the model predicted the most
ml_results: {"attributions": {"house_size": 300000.0, "state": 31200.0},
             "predictions": {"price": 790000.0}, "notes": "house_size (2100) increased the value the
10 rows in set (0.0427 sec)
```

To generate prediction explanations for one or more rows of data, see Generate Prediction Explanations for a Row of Data.

What's Next

- Learn how to Score a Regression Model.

6.7.2.5 Score a Regression Model

After generating predictions and explanations, you can score the model to assess its reliability. For a list of scoring metrics you can use with regression models, see Regression Metrics. For this use case, you use the test dataset for validation. In a real-world use case, you should use a separate validation dataset that has the target column and ground truth values for the scoring validation. You should also use a larger number of records for training and validation to get a valid score.

Before You Begin

Complete the following tasks:

- Prepare Data for a Regression Model

- Train a Regression Model
- Generate Predictions for a Regression Model
- Query Model Explanation and Generate Prediction Explanations for a Regression Model

Score Model

1. If not already done, load the model. You can use the session variable for the model that is valid for the duration of the connection. Alternatively, you can use the model handle previously set. For the option to set the user name, you can set it to `NULL`.

   The following example uses the session variable.

   ```
   mysql> CALL sys.ML_MODEL_LOAD(@model, NULL);
   ```

   The following example uses the model handle.

   ```
   mysql> CALL sys.ML_MODEL_LOAD('regression_use_case', NULL);
   ```

2. Score the model with the `ML_SCORE` routine and use the `r2` metric.

   ```
   mysql> CALL sys.ML_SCORE('regression_data.house_price_testing', 'price', 'regression_use_case', 'r2', @
   ```

   Where:

   - `regression_data.house_price_testing` is the fully qualified name of the validation dataset.
   - `price` is the target column name with ground truth values.
   - `'regression_use_case'` is the model handle for the trained model.
   - `r2` is the selected scoring metric.
   - `@regression_score` is the session variable name for the score value.
   - `NULL` means that no other options are defined for the routine.

3. Retrieve the score by querying the @regression_score session variable.

   ```
   mysql> SELECT @regression_score;
   +--------------------+
   | @regression_score  |
   +--------------------+
   | 0.8438237905502319 |
   +--------------------+
   1 row in set (0.0453 sec)
   ```

4. If done working with the model, unload it with the `ML_MODEL_UNLOAD` routine.

   ```
   mysql> CALL sys.ML_MODEL_UNLOAD('regression_use_case');
   ```

   To avoid consuming too much memory, it is good practice to unload a model when you are finished using it.

What's Next

- Review other Machine Learning Use Cases.

6.7.3 Generate Forecasts

Forecasting models generate predictions on timeseries data. Some examples of forecasting include predicting the closing price of a stock, predicting the number of units sold in a day, and predicting the average price of gasoline.

The following tasks use a dataset generated by OCI GenAI using Meta Llama Models. The forecasting use-case is a univariate forecasting model that captures the monthly demand for electricity in San Francisco, California.

To generate your own datasets to create machine learning models in MySQL HeatWave, learn how to Generate Text-Based Content.

Note

Datasets were generated using Meta Llama models. Your use of this Llama model is subject to your Oracle agreements and this Llama license agreement: https://downloads.mysql.com/docs/LLAMA_31_8B_INSTRUCT-license.pdf.

6.7.3.1 Forecasting Task Types

This topic describes the types of forecasting models supported by AutoML.

Before You Begin

- Review the list of supported Forecasting Models.

You can create the following types of forecasting models.

Univariate Models

In a univariate model, you define one numeric column as an endogenous variable, specified as a `JSON_ARRAY`. This is the target column that MySQL HeatWave AutoML forecasts. For example, you forecast the rainfall for the next month by using the past daily rainfall as an endogenous variable.

Multivariate Models

In a multivariate model, you define multiple columns as endogenous variables, specified as a `JSON_ARRAY`. You must define one of these columns as the target column (the column with ground truth values). For example, you forecast the rainfall for the next month by using the past rainfall, temperature highs and lows, atmospheric pressure, and humidity. The target column is rainfall.

Univariate and Multivariate Models with Exogenous Variables

You have the option to define exogenous variables for univariate and multivariate models. These columns have independent, non-forecast, predictive variables. For example, you forecast future sales and use weather conditions like rainfall and high and low daily temperature values as exogenous variables.

Selecting Forecasting Models

To specify which models that are considered for training, use the `model_list` option and enter the appropriate model names. If only one model is set for `model_list`, then only that model is considered. Review the list of supported Forecasting Models and which type of model they support, univariate endogenous models, univariate endogenous models with exogenous variables, and multivariate endogenous models with exogenous variables. .

If the `model_list` option is not set, then `ML_TRAIN` considers all supported models during the algorithm selection stage. If *`options`* includes `exogenous_variables`, all supported models are still considered, including models that do not support `exogenous_variables`.

For example, if *`options`* includes univariate `endogenous_variables` with `exogenous_variables`, then `ML_TRAIN` considers `NaiveForecaster`, `ThetaForecaster`, `ExpSmoothForecaster`, `ETSForecaster`, `STLwESForecaster`, `STLwARIMAForecaster`,

`SARIMAXForecaster`, and `OrbitForecaster`. `ML_TRAIN` ignores `exogenous_variables` if the model does not support them.

Similarly, if *`options`* includes multivariate `endogenous_variables` with `exogenous_variables`, then `ML_TRAIN` considers `VARMAXForecaster` and `DynFactorForecaster`.

If *`options`* also includes `include_column_list`, this forces `ML_TRAIN` to only consider those models that support `exogenous_variables`.

What's Next

- Learn more about Prediction Intervals.
- Learn how to Train a Forecasting Model.

6.7.3.2 Prediction Intervals

MySQL 8.4.0 introduces prediction intervals for forecasting models. This feature specifies upper and lower bounds on predictions for forecasting based on level of confidence. For example, for a prediction interval of 0.95 with a lower bound of 25 units and an upper bound of 65 units, you are 95% confident that product ABC will sell between 25 and 65 units on a randomly selected day.

The `prediction_interval` option is included for the `ML_PREDICT_TABLE` routine, which specifies a level of confidence. Predictions provide three outputs corresponding to each endogenous variable: the forecasted value, a lower bound, and an upper bound.

For the `prediction_interval` option:

- The default value is 0.95.
- The data type for this value must be FLOAT.
- The value must be greater than 0 and less than 1.0.

What's Next

- Learn how to Train a Forecasting Model.

6.7.3.3 Prepare Data for a Forecasting Model

This topic describes how to prepare the data to use for a forecasting machine learning model. It uses a data sample generated by OCI GenAI. To prepare the data for this use case, you set up a training dataset and a testing dataset. The training dataset has 37 records, and the testing dataset has 4 records. In a real-life use case, you should prepare a larger amount of records for training and testing, and ensure the predictions are valid and reliable before testing on unlabeled data. To ensure reliable predictions, you should create an additional validation dataset. You can reserve 20% of the records in the training dataset to create the validation dataset.

You have the option to automatically Prepare Training and Testing Datasets with your own data by using the `TRAIN_TEST_SPLIT` routine.

Before You Begin

- Learn how to Prepare Data.

Prepare Data

To prepare the data for the forecasting model:

1. Connect to your MySQL HeatWave Database System.
2. Create and use the database to store the data.

```
mysql> CREATE DATABASE forecasting_data;
mysql> USE forecasting_data;
```

3. Create the table that is the sample dataset.

```
mysql> CREATE TABLE electricity_demand (
    date DATE PRIMARY KEY,
    demand FLOAT NOT NULL,
    temperature FLOAT NOT NULL
);
```

4. Insert the sample data into the table. Copy and paste the following commands.

```
INSERT INTO electricity_demand (date, demand, temperature) VALUES
('2022-01-01', 929.00, 53.53),
('2022-01-31', 949.69, 60.80),
('2022-03-02', 1160.84, 69.28),
('2022-04-01', 1054.52, 74.48),
('2022-05-01', 1061.40, 71.06),
('2022-05-31', 1012.36, 58.05),
('2022-06-30', 1098.87, 51.90),
('2022-07-30', 964.31, 39.70),
('2022-08-29', 1026.06, 32.47),
('2022-09-28', 995.23, 30.82),
('2022-10-28', 1076.04, 32.97),
('2022-11-27', 1059.46, 42.91),
('2022-12-27', 1060.97, 51.52),
('2023-01-26', 1153.59, 60.24),
('2023-02-25', 1204.72, 68.21),
('2023-03-27', 1203.33, 70.67),
('2023-04-26', 1218.42, 70.31),
('2023-05-26', 1163.28, 59.59),
('2023-06-25', 1161.86, 50.63),
('2023-07-25', 1131.38, 38.29),
('2023-08-24', 1138.72, 27.57),
('2023-09-23', 1119.34, 31.31),
('2023-10-23', 1090.38, 34.41),
('2023-11-22', 1213.87, 38.52),
('2023-12-22', 1219.91, 54.54),
('2024-01-21', 1193.49, 57.09),
('2024-02-20', 1326.44, 67.41),
('2024-03-21', 1274.64, 69.63),
('2024-04-20', 1325.90, 70.39),
('2024-05-20', 1351.45, 62.94),
('2024-06-19', 1306.45, 50.31),
('2024-07-19', 1341.97, 40.76),
('2024-08-18', 1214.96, 30.90),
('2024-09-17', 1300.12, 26.04),
('2024-10-17', 1262.46, 31.98),
('2024-11-16', 1281.46, 40.31),
('2024-12-16', 1331.06, 52.46),
('2025-01-15', 1379.42, 62.40),
('2025-02-14', 1426.11, 66.55),
('2025-03-16', 1381.74, 69.40),
('2025-04-15', 1488.34, 65.22);
```

5. Create the table to use as the training dataset. It retrieves some of the data from the sample dataset.

```
mysql> CREATE TABLE electricity_demand_train AS SELECT * FROM electricity_demand WHERE date < '2025-
```

6. Create the table to use for generating predictions. This is the test dataset. It retrieves the data from the sample dataset not used for the training dataset. It has the same columns as the training dataset, but the target column, `demand`, is not considered when generating predictions.

```
mysql> CREATE TABLE electricity_demand_test AS SELECT * FROM electricity_demand WHERE date >= '2025-
```

What's Next

- Learn how to Train a Forecasting Model.

6.7.3.4 Train a Forecasting Model

After preparing the data for a forecasting model, you can train the model.

This topic has the following sections.

- Before You Begin
- Requirements for Forecasting Training
- Forecasting Options
- Unsupported Routines
- Train Model
- What's Next

Before You Begin

- Review and complete all the tasks to Prepare Data for a Forecasting Model.

Requirements for Forecasting Training

Define the following required parameters to train a forecasting model.

- Set the `task` parameter to `forecasting`.
- `datetime_index`: Define the column that has date and time data. The model uses this column as an index for the forecast variable. The following data types for this column are supported: `DATETIME`, `TIMESTAMP`, `DATE`, `TIME`, and `YEAR`, or an auto-incrementing index.

  The forecast models `SARIMAXForecaster`, `VARMAXForecaster`, and `DynFactorForecaster` cannot back test, that is forecast into training data, when using `exogenous_variables`. Therefore, the predict table must not overlap the `datetime_index` with the training table. The start date in the predict table must be a date immediately following the last date in the training table when `exogenous_variables` are used. For example, the predict table has to start with year 2024 if the training table with `YEAR` data type `datetime_index` ends with year 2023. The predict table cannot start with year, for example, 2025 or 2030, because that would miss out 1 and 6 years, respectively.

  When *`options`* do not include `exogenous_variables` , the predict table can overlap the `datetime_index` with the training table. This supports back testing, with the exception of the following models: SARIMAXForecaster, VARMAXForecaster, and DynFactorForecaster.

  The valid range of years for `datetime_index` dates must be between 1678 and 2261. An error generates if any part of the training table or predict table has dates outside this range. The last date in the training table plus the predict table length must still be inside the valid year range. For example, if the `datetime_index` in the training table has `YEAR` data type, and the last date is year 2023, the predict table length must be less than 238 rows: 2261 minus 2023 equals 238 rows.
- `endogenous_variables`: Define the column or columns to be forecast. One of these columns must also be specified as the `target_column_name`.

Forecasting Options

Based on the type of forecasting model you train, set the appropriate `JSON` options:

- `exogenous_variables`: Define the column or columns that have independent, non-forecast, predictive variables. These optional variables are not forecast, but help to predict the future values of the forecast variables. These variables affect a model without being affected by it. For example, for sales forecasting these variables might be advertising expenditure, occurrence of promotional events, weather, or holidays. Review Forecasting Models to see which models support exogenous variables.

- `model_list`: Set the type of forecasting model algorithm. See Forecasting Models to review supported algorithms.
- `include_column_list`: Define the columns of `exogenous_variables` that must be included for training and should not be dropped.

Unsupported Routines

You cannot run the following routines for a trained forecasting model:

- `ML_EXPLAIN`
- `ML_EXPLAIN_ROW`
- `ML_EXPLAIN_TABLE`
- `ML_PREDICT_ROW`

Train Model

After following the steps to Prepare Data for Forecasting Model, train the model with the `ML_TRAIN` routine and use the `electricity_demand_training` table previously created. Before training the model, it is good practice to define the model handle instead of automatically creating one. See Define Model Handle.

1. Optionally, set the value of the session variable, which sets the model handle to this same value.

```
mysql> SET @variable = 'model_handle';
```

Replace *`@variable`* and *`model_handle`* with your own definitions. For example:

```
mysql> SET @model='forecasting_use_case';
```

The model handle is set to `forecasting_use_case`.

2. Run the `ML_TRAIN` routine.

```
mysql> CALL sys.ML_TRAIN('table_name', 'target_column_name', JSON_OBJECT('task', 'task_name'), mode
```

Replace *`table_name`*, *`target_column_name`*, *`task_name`*, and *`model_handle`* with your own values.

The following example runs `ML_TRAIN` on the training dataset previously created.

```
mysql> CALL sys.ML_TRAIN('forecasting_data.electricity_demand_train', 'demand',
                         JSON_OBJECT('task', 'forecasting',
                                     'datetime_index', 'date',
                                     'endogenous_variables', JSON_ARRAY('demand')), @model);
```

Where:

- `forecasting_data.electricity_demand_train` is the fully qualified name of the table that contains the training dataset (*`database_name.table_name`*).
- `demand` is the name of the target column, which contains ground truth values.
- The `JSON_OBJECT` defines the following:
  - `'task', 'forecasting'` specifies the machine learning task type.
  - `'datetime_index', 'date'` defines the `date` column as the one with data and time data.
  - `'endogenous_variables', JSON_ARRAY('demand')` defines the endogenous variables in a `JSON_ARRAY`. Since it is a univariate model, the only endogenous variable is `demand`.

- `@model` is the session variable previously set that defines the model handle to the name defined by the user: `forecasting_use_case`. If you do not define the model handle before training the model, the model handle is automatically generated, and the session variable only stores the model handle for the duration of the connection. User variables are written as `@var_name`. Any valid name for a user-defined variable is permitted. See Work with Model Handles to learn more.

3. When the training operation finishes, the model handle is assigned to the `@model` session variable, and the model is stored in the model catalog. View the entry in the model catalog with the following query. Replace *`user1`* with your MySQL account name.

```
mysql> SELECT model_id, model_handle, train_table_name FROM ML_SCHEMA_user1.MODEL_CATALOG WHERE model_h
+----------+-------------------------------------------+-----------------------------------------
| model_id | model_handle                              | train_table_name
+----------+-------------------------------------------+-----------------------------------------
|        3 | forecasting_use_case                      | forecasting_data.electricity_demand_train
+----------+-------------------------------------------+-----------------------------------------
```

What's Next

- Learn how to Generate Predictions for a Forecasting Model.
- Review additional Syntax Examples for Forecast Training

6.7.3.5 Generate Predictions for a Forecasting Model

After training the model, you can generate predictions.

To generate predictions, use the sample data from the `electricity_demand_test` dataset. Even though the table has labels for the `demand` target column, the column is not considered when generating predictions. This allows you to compare the predictions to the actual values in the dataset and determine if the predictions are reliable. Once you determine the trained model is reliable for generating predictions, you can start using unlabeled datasets for generating predictions.

The `datetime_index` column must be included. If using `exogenous_variables`, they must also be included. Any extra columns, for example `endogenous_variables`, are ignored for the prediction, but included in the output table.

As of MySQL 8.4.0, prediction interval values are included in the prediction results. See Prediction Intervals to learn more.

You cannot run `ML_PREDICT_ROW` with forecasting models.

Before You Begin

Complete the following tasks:

- Prepare Data for a Forecasting Model.
- Review how to Train a Forecasting Model.

Generate Forecasts for a Table

1. If not already done, load the model. You can use the session variable for the model that is valid for the duration of the connection. Alternatively, you can use the model handle previously set. For the option to set the user name, you can set it to `NULL`.

   The following example uses the session variable.

```
mysql> CALL sys.ML_MODEL_LOAD(@model, NULL);
```

   The following example uses the model handle.

```
mysql> CALL sys.ML_MODEL_LOAD('forecasting_use_case', NULL);
```

2. Make predictions for the test dataset by using the `ML_PREDICT_TABLE` routine.

```
mysql> CALL sys.ML_PREDICT_TABLE(table_name, model_handle, output_table_name), [options]);
```

Replace *`table_name`*, *`model_handle`*, and *`output_table_name`* with your own values. Add *`options`* as needed.

As of MySQL 9.4.1, you have the option to specify the input table and output table as the same table if specific conditions are met. See Input Tables and Output Tables to learn more.

The following example runs `ML_PREDICT_TABLE` on the testing dataset previously created.

```
mysql> CALL sys.ML_PREDICT_TABLE('forecasting_data.electricity_demand_test', @model, 'forecasting_da
```

Where:

- `forecasting_data.electricity_demand_test` is the fully qualified name of the input table that contains the data to generate predictions for (*`database_name.table_name`*).
- `@model` is the session variable for the model handle.
- `forecasting_data.electricity_demand_predictions` is the fully qualified name of the output table with predictions (*`database_name.table_name`*).
- `NULL` sets no options for the routine.

3. Query the `demand`, and `ml_results` columns from the output table. This allows you to compare the real demand with the generated forecast. You can also review the lower bound and upper bound prediction interval values for each forecast. Since no prediction interval value is set when running `ML_PREDICT_TABLE`, the default value of 0.95 is used.

```
mysql> SELECT Approved, Prediction, ml_results FROM electricity_demand_predictions;
+---------+-----------------------------------------------------------------------------------------
| demand  | ml_results
+---------+-----------------------------------------------------------------------------------------
| 1379.42 | {"predictions": {"demand": 1316.5263873105694, "prediction_interval_demand": [1312.64875
| 1426.11 | {"predictions": {"demand": 1322.148597544633, "prediction_interval_demand": [1317.796601
| 1381.74 | {"predictions": {"demand": 1327.6276527841787, "prediction_interval_demand": [1322.84806
| 1488.34 | {"predictions": {"demand": 1332.9671980996688, "prediction_interval_demand": [1327.79518
+---------+-----------------------------------------------------------------------------------------
4 rows in set (0.0455 sec)
```

What's Next

- Learn how to Score a Forecasting Model

6.7.3.6 Score a Forecasting Model

After generating predictions, you can score the model to assess its reliability. For a list of scoring metrics you can use with forecasting models, see Forecasting Metrics. For this use case, you use the test dataset for validation. In a real-world use case, you should use a separate validation dataset that has the target column and ground truth values for the scoring validation. You should also use a larger number of records for training and validation to get a valid score.

The `ML_SCORE` routine does not require a `target_column_name` for forecasting, so you can set it to `NULL`. However, the target column needs to be in the table to generate a valid score value.

Before You Begin

Complete the following tasks:

- Prepare Data for a Forecasting Model

- Train a Forecasting Model
- Generate Predictions for a Forecasting Model

Score Model

1. If not already done, load the model. You can use the session variable for the model that is valid for the duration of the connection. Alternatively, you can use the model handle previously set. For the option to set the user name, you can set it to `NULL`.

   The following example uses the session variable.

   ```
   mysql> CALL sys.ML_MODEL_LOAD(@model, NULL);
   ```

   The following example uses the model handle.

   ```
   mysql> CALL sys.ML_MODEL_LOAD('forecasting_use_case', NULL);
   ```

2. Score the model with the `ML_SCORE` routine and use the `neg_sym_mean_abs_percent_error` metric.

   ```
   mysql> CALL sys.ML_SCORE(table_name, target_column_name, model_handle, metric, score, [options]);
   ```

   Replace *`table_name`*, *`target_column_name`*, *`model_handle`*, *`metric`*, *`score`* with your own values.

   The following example runs `ML_SCORE` on the testing dataset previously created.

   ```
   mysql> CALL sys.ML_SCORE('forecasting_data.electricity_demand_test', 'demand', @model, 'neg_sym_mean_ab
   ```

   Where:

   - `forecasting_data.electricity_demand_test` is the fully qualified name of the validation dataset.
   - `demand` is the target column name with ground truth values.
   - `@model` is the session variable for the model handle.
   - `neg_sym_mean_abs_percent_error` is the selected scoring metric.
   - `@forecasting_score` is the session variable name for the score value.
   - `NULL` means that no other options are defined for the routine.

3. Retrieve the score by querying the @forecasting_score session variable.

   ```
   mysql> SELECT @forecasting_score;
   +----------------------+
   | @forecasting_score   |
   +----------------------+
   | -0.06810028851032257 |
   +----------------------+
   1 row in set (0.0429 sec)
   ```

4. If done working with the model, unload it with the `ML_MODEL_UNLOAD` routine.

   ```
   mysql> CALL sys.ML_MODEL_UNLOAD('forecasting_use_case');
   ```

   To avoid consuming too much memory, it is good practice to unload a model when you are finished using it.

What's Next

- Review other Machine Learning Use Cases.

6.7.4 Detect Anomalies

Anomaly detection, which is also known as outlier detection, is the machine learning task that finds unusual patterns in data.

MySQL HeatWave AutoML supports unsupervised and semi-supervised anomaly detection. See Anomaly Detection Learning Types to learn more.

The following tasks use datasets generated by OCI GenAI using Meta Llama Models. The anomaly detection use-cases are to find unusual patterns of purchasing behavior for credit card transactions, and to find anomalies in log data.

To generate your own datasets to create machine learning models in MySQL HeatWave, learn how to Generate Text-Based Content.

Note

Datasets were generated using Meta Llama models. Your use of this Llama model is subject to your Oracle agreements and this Llama license agreement: https://downloads.mysql.com/docs/LLAMA_31_8B_INSTRUCT-license.pdf.

6.7.4.1 Anomaly Detection Model Types

You can use the following anomaly detection model types:

- GkNN (Generalized kth Nearest Neighbors)
- PCA (Principal Component Analysis) (As of MySQL 8.4.0)
- GLOF (Generalized Local Outlier Factor) (As of MySQL 8.4.0)

GkNN Model

Generalized kth Nearest Neighbors (GkNN) is an algorithm model developed at Oracle. It is a single ensemble algorithm that outperforms state-of-the-art models on public benchmarks. It can identify common anomaly types, such as local, global, and clustered anomalies, and can achieve an AUC score that is similar to, or better than, when identifying the following:

- Global anomalies compared to KNN, with an optimal k hyperparameter value.
- Local anomalies compared to LOF, with an optimal k hyperparameter value.
- Clustered anomalies.

Optimal k hyperparameter values would be extremely difficult to set without labels and knowledge of the use-case.

Other algorithms would require training and comparing scores from at least three algorithms to address global and local anomalies, ignoring clustered anomalies: LOF for local, KNN for global, and another generic method to establish a 2/3 voting mechanism.

What's Next

- Learn more about the following:
  - Anomaly Detection Learning Types
  - Anomaly Detection for Logs

6.7.4.2 Anomaly Detection Learning Types

In MySQL HeatWave, there are two types of learning for anomaly detection models: unsupervised and semi-supervised.

Unsupervised Anomaly Detection

When running an unsupervised anomaly detection model, the machine learning algorithm requires no labeled data. When training the model, the `target_column_name` parameter must be set to `NULL`.

Semi-supervised Anomaly Detection

MySQL 9.0.1-u1 introduces support for semi-supervised learning. This type of machine learning algorithm uses a specific set of labeled data along with unlabeled data to detect anomalies. To enable this, use the `experimental` and `semisupervised` options. The `target_column_name` parameter must specify a column whose only allowed values are 0 (normal), 1 (anomalous), and NULL (unlabeled). All rows are used to train the unsupervised component, while the rows with a value different than NULL are used to train the supervised component.

What's Next

- Learn more about the following:
  - Anomaly Detection Algorithm Model Types
  - Anomaly Detection for Logs
- Learn how to Prepare Data for an Anomaly Detection Model.

6.7.4.3 Anomaly Detection for Logs

MySQL 9.2.2 introduces the ability to detect anomalies in log data. To perform anomaly detection on logs, log data is cleaned, segemented, and encoded before running anomaly detection. This feature leverages the log template miner Drain3.

Consider the following when running anomaly detection on logs.

- The input table can only have the following columns:
  - The column containing the logs.
  - If including logs from different sources, a column containing the source of each log. The values in this column contain the names of the sources that each log belongs to. These values are used to group each host's logs together. If this column is not present, it is assumed that all logs originate from the same source.
  - If including labeled data, a column identifying the labeled log lines. See Semi-supervised Anomaly Detection to learn more.
  - As of MySQL 9.4.1, at least one column must act as the primary key to establish the temporal order of logs. If the primary key column (or columns) is not one of the previous required columns (log data, source of log, or label), then you must use the `exclude_column_list` option when running `ML_TRAIN` to exclude all primary key columns that don't include required data. See Syntax Examples for Anomaly Detection Training to review relevant examples.
- If the input table has additional columns to the ones permitted, you must use the `exclude_column_list` option when running `ML_TRAIN` to exclude irrelevant columns.
- The data collected for anomaly detection can be unsupervised or semi-supervised. To run semi-supervised anomaly detection, you can provide a separate column in the input table with labels for the labeled log lines. This column labels identified anomalous logs with a value of 1, non-anomalous logs with 0, and unlabeled logs with NULL. See Semi-supervised Anomaly Detection to learn more.
- In addition to the anomaly scores included in the output table, you have the option to leverage MySQL HeatWave GenAI to provide textual log summaries.

- By default the following parameters are masked in the input data (training or test data): IP, DATETIME, TIME, HEX, IPPORT, and OCID. You have the option to mask additional regex patterns with the `additional_masking_regex` option.
- For versions before MySQL 9.5.0, textual log data is transformed into numerical vectors using TF-IDF (Term Frequency-Inverse Document Frequency), which is a statistical feature extractor that assigns numerical weights to words according to their frequency across documents. As of MySQL 9.5.0, MySQL HeatWave uses a combination of a keyword feature extractor and an embedding model to train models. This allows trained models to capture semantic meanings in log data. You have the option to select the keyword model and any embedding model supported by MySQL HeatWave with the training options `keyword_model` and `embedding_model`. The available keyword model options are `tf-idf` and `NULL`. To review supported embedding models, run the following query: `SELECT sys.ML_LIST_LLMS();` and see models that have `capabilities` with `TEXT_EMBEDDINGS`. The default keyword feature extractor is `tf-idf`, and the default embedding model is `multilingual-e5-small`. Using an embedding model causes higher memory usage. You can set either `embedding_model` or `keyword_model` to `NULL`, but you cannot set both to `NULL`.

What's Next

- Learn more about the following:
  - Anomaly Detection Algorithm Model Types
  - Anomaly Detection Learning Types
- Learn how to Prepare Data for an Anomaly Detection Model.

6.7.4.4 Prepare Data for an Anomaly Detection Model

This topic describes how to prepare the data to use for two anomaly detection machine learning models: a semi-supervised anomaly detection model, and an unsupervised anomaly detection model for logs. It uses data samples generated by OCI GenAI. To prepare the data for this use case, you set up a training dataset and a testing dataset. In a real-life use case, you should prepare a larger amount of records than these data samples for training and testing, and ensure the predictions are valid and reliable before testing on unlabeled data. To ensure reliable predictions, you should create an additional validation dataset. You can reserve 20% of the records in the training dataset to create the validation dataset.

You have the option to automatically Prepare Training and Testing Datasets with your own data by using the `TRAIN_TEST_SPLIT` routine.

This topic has the following sections.

- Before You Begin
- Prepare Data for a Semi-Supervised Anomaly Detection Model
- Prepare Data for an Unsupervised Anomaly Detection Model for Logs
- What's Next

Before You Begin

- Learn how to Prepare Data.

Prepare Data for a Semi-Supervised Anomaly Detection Model

The semi-supervised anomaly detection model (available as of MySQL 9.0.1-u1) looks for unusual patterns in credit card transactions. The data has a column, `target`, that has three possible values: 0 for normal, 1 for anomalous, and NULL for unlabeled.

To prepare the data for the semi-supervised anomaly detection model:

1. Connect to your MySQL HeatWave Database System.

2. Create and use the database to store the data.

```
mysql> CREATE DATABASE anomaly_data;
mysql> USE anomaly_data;
```

3. Create the table to insert the sample data into. This is the training dataset.

```
mysql> CREATE TABLE credit_card_train (
    transaction_id INT AUTO_INCREMENT PRIMARY KEY,
    home_address VARCHAR(100),
    purchase_location VARCHAR(100),
    purchase_amount DECIMAL(10, 2),
    purchase_time DATETIME,
    target INT
);
```

4. Insert the sample data to train into the table. Copy and paste the following commands.

```
INSERT INTO credit_card_train (home_address, purchase_location, purchase_amount, purchase_time, target)
VALUES
    ('123 Main St, City A', 'Store X, City A', 50.75, '2023-08-01 14:30:00', 0),
    ('456 Elm St, City B', 'Cafe B, City B', 15.20, '2023-08-02 09:45:00', 1),
    ('789 Oak Ave, City C', 'Online Shop', 250.00, '2023-08-03 18:10:00', 0),
    ('222 Maple Lane, City A', 'Grocery Store A', 35.50, '2023-08-04 11:00:00', NULL),
    ('555 River Rd, City D', 'Electronics Store, City D', 800.50, '2023-08-05 16:20:00', 1),
    ('1010 Mountain View, City E', 'Boutique, City E', 120.30, '2023-08-06 10:35:00', 0),
    ('333 Ocean Blvd, City F', 'Convenience Store, City F', 20.15, '2023-08-07 19:50:00', NULL),
    ('666 Sky St, City G', 'Luxury Store, City G', 1500.00, '2023-08-08 12:00:00', 1),
    ('999 Green Valley, City H', 'Hardware Store, City H', 75.90, '2023-08-09 08:40:00', 0),
    ('111 Sunset Ave, City A', 'Store X, City A', 60.40, '2023-08-10 15:10:00', NULL),
    ('2222 Country Road, City B', 'Cafe B, City B', 28.75, '2023-08-11 07:30:00', 0),
    ('3333 Lakeside, City C', 'Online Shop', 180.25, '2023-08-12 13:20:00', 1),
    ('4444 Forest Glade, City D', 'Grocery Store, City D', 45.60, '2023-08-13 09:50:00', 0),
    ('5555 Meadow Lane, City E', 'Electronics Store, City E', 300.75, '2023-08-14 17:40:00', NULL),
    ('6666 Creekside, City F', 'Boutique, City F', 95.50, '2023-08-15 11:30:00', 1),
    ('7777 Hillcrest, City G', 'Convenience Store, City G', 12.80, '2023-08-16 18:50:00', 0),
    ('8888 Riverbank, City H', 'Luxury Store, City H', 2200.00, '2023-08-17 14:10:00', NULL),
    ('9999 Sunrise Blvd, City A', 'Hardware Store, City A', 55.25, '2023-08-18 09:30:00', 0),
    ('101010 Ocean View, City B', 'Store X, City B', 70.50, '2023-08-19 16:40:00', 1),
    ('111111 Mountain Rd, City C', 'Cafe C, City C', 32.90, '2023-08-20 11:20:00', NULL),
    ('121212 Downtown, City D', 'Online Shop', 450.00, '2023-08-21 17:50:00', 0),
    ('131313 Lakeside Ave, City E', 'Grocery Store, City E', 28.50, '2023-08-22 10:10:00', 1),
    ('141414 Green Park, City F', 'Electronics Store, City F', 650.75, '2023-08-23 15:30:00', 0),
    ('151515 Skyway, City G', 'Boutique, City G', 180.40, '2023-08-24 08:50:00', NULL),
    ('161616 Meadow View, City H', 'Convenience Store, City H', 35.10, '2023-08-25 13:40:00', 0),
    ('171717 River Rd, City A', 'Luxury Store, City A', 1300.50, '2023-08-26 19:20:00', 1),
    ('181818 Sunset Blvd, City B', 'Hardware Store, City B', 85.60, '2023-08-27 12:30:00', NULL),
    ('191919 Country Lane, City C', 'Store Y, City C', 150.20, '2023-08-28 07:40:00', 0),
    ('202020 Forest Edge, City D', 'Cafe D, City D', 42.75, '2023-08-29 14:50:00', 1),
    ('212121 Lakeside View, City E', 'Online Shop', 220.50, '2023-08-30 09:20:00', 0),
    ('222222 Creekside Ave, City F', 'Grocery Store, City F', 55.90, '2023-08-31 16:10:00', NULL);
```

5. Create the table to use for generating predictions. This is the test dataset. It has the same columns as the training dataset. The target column, `target`, is used for the semi-supervised component of the training.

```
mysql> CREATE TABLE credit_card_test (
    transaction_id INT AUTO_INCREMENT PRIMARY KEY,
    home_address VARCHAR(100),
    purchase_location VARCHAR(100),
    purchase_amount DECIMAL(10, 2),
    purchase_time DATETIME,
    target INT
);
```

6. Insert the sample data to test into the table. Copy and paste the following commands.

```
INSERT INTO credit_card_test (home_address, purchase_location, purchase_amount, purchase_time, target)
VALUES
```

```
    ('3030 Riverbank Dr, City I', 'Grocery Store, City I', 52.30, '2023-09-01 10:30:00', 0),
    ('3131 Mountain Rd, City J', 'Electronics Store, City J', 120.50, '2023-09-02 16:45:00', 0),
    ('3232 Ocean Ave, City K', 'Boutique, City K', 85.20, '2023-09-03 11:20:00', 1),
    ('3333 Green Valley, City L', 'Convenience Store, City L', 25.60, '2023-09-04 18:50:00', 0),
    ('3434 Sunset Blvd, City I', 'Luxury Store, City I', 1600.00, '2023-09-05 14:10:00', 1),
    ('3535 Country Lane, City J', 'Hardware Store, City J', 68.40, '2023-09-06 09:30:00', 0),
    ('3636 Lakeside View, City K', 'Store Z, City K', 135.75, '2023-09-07 17:20:00', 0),
    ('3737 Forest Glade, City L', 'Cafe E, City L', 38.50, '2023-09-08 12:40:00', 1),
    ('3838 Meadow Lane, City I', 'Online Shop', 280.50, '2023-09-09 08:50:00', 0),
    ('3939 Creekside Ave, City J', 'Grocery Store, City J', 48.75, '2023-09-10 15:30:00', 0),
    ('4040 River Rd, City K', 'Electronics Store, City K', 720.25, '2023-09-11 11:10:00', 1),
    ('4141 Skyway Blvd, City L', 'Boutique, City L', 165.90, '2023-09-12 17:40:00', 0),
    ('4242 Hillcrest Rd, City I', 'Convenience Store, City I', 22.50, '2023-09-13 10:20:00', 0),
    ('4343 Riverbank View, City J', 'Luxury Store, City J', 2100.75, '2023-09-14 16:50:00', 1),
    ('4444 Country Club, City K', 'Hardware Store, City K', 92.30, '2023-09-15 12:30:00', 0),
    ('4545 Lakeside Ave, City L', 'Store Alpha, City L', 145.60, '2023-09-16 08:40:00', 0),
    ('4646 Forest Edge, City I', 'Cafe F, City I', 55.80, '2023-09-17 15:20:00', 1),
    ('4747 Creekside View, City J', 'Online Shop', 320.40, '2023-09-18 11:50:00', 0),
    ('4848 Meadow Park, City K', 'Grocery Store, City K', 62.50, '2023-09-19 18:30:00', 0),
    ('4949 River Walk, City L', 'Electronics Store, City L', 550.30, '2023-09-20 14:10:00', 1);
```

Prepare Data for an Unsupervised Anomaly Detection Model for Logs

The anomaly detection model for logs (available as of MySQL 9.2.2) looks for unusual patterns in log data. The model uses unsupervised learning, so the `target` column is excluded for training and predicting anomalies.

To prepare the data for the anomaly detection model for logs:

1. Connect to your MySQL HeatWave Database System.
2. If not already done, create and use the database to store the data.

```
mysql> CREATE DATABASE anomaly_log_data;
mysql> USE anomaly_log_data;
```

3. Create the table to insert the sample data into. This is the training dataset.

```
mysql> CREATE TABLE training_data (
    log_id INT AUTO_INCREMENT PRIMARY KEY,
    log_message TEXT,
    timestamp DATETIME,
    target TINYINT
);
```

4. Insert the sample data to be trained into the table. Copy and paste the following commands.

```
INSERT INTO training_data (log_message, timestamp, target) VALUES
    ("User login successful: admin", "2023-08-07 09:00:00", 0),
    ("Database connection established", "2023-08-07 09:05:23", 0),
    ("Failed login attempt from IP: 192.168.1.20", "2023-08-07 09:12:15", 1),
    ("Server load is high: 85%", "2023-08-07 09:20:30", 1),
    ("Normal system behavior", "2023-08-07 09:35:00", 0),
    ("Anomalous CPU usage spike", "2023-08-07 10:10:45", 1),
    ("New user registered", "2023-08-07 10:25:00", 0),
    ("Error: File not found", "2023-08-07 11:02:10", 1),
    ("System startup completed", "2023-08-07 11:30:00", 0),
    ("Network packet loss detected", "2023-08-07 12:15:35", 1),
    ("User activity: John accessed dashboard", "2023-08-07 13:00:20", 0),
    ("Security alert: Brute force attack detected", "2023-08-07 13:45:55", 1),
    ("Log rotation completed", "2023-08-07 14:20:00", 0),
    ("Anomalous memory usage pattern", "2023-08-07 15:05:30", 1),
    ("User feedback submitted", "2023-08-07 15:40:10", 0),
    ("System error: Out of memory", "2023-08-07 16:15:25", 1),
    ("Network connectivity restored", "2023-08-07 16:50:00", 0),
    ("Unlabeled log entry", NULL, NULL),
    ("Potential intrusion detected", "2023-08-07 17:35:40", 1),
    ("User logout: Jane", "2023-08-07 18:10:00", 0);
```

5. Create the table to use for generating predictions. This is the test dataset. It has the same columns as the training dataset, but the target column, `target`, must be excluded when generating predictions.

```
mysql> CREATE TABLE testing_data (
    log_id INT AUTO_INCREMENT PRIMARY KEY,
    log_message TEXT,
    timestamp DATETIME,
    target TINYINT
);
```

6. Insert the sample data to test into the table. Copy and paste the following commands.

```
INSERT INTO testing_data (log_message, timestamp, target) VALUES
    ("User login failed: Invalid credentials", "2023-08-08 10:30:00", 1),
    ("Server response time increased", "2023-08-08 11:15:45", 1),
    ("Normal database query", "2023-08-08 12:00:20", 0),
    ("Unusual network traffic from IP: 10.0.0.5", "2023-08-08 12:45:30", 1),
    ("System update completed successfully", "2023-08-08 13:30:00", 0),
    ("Error log: Stack trace included", "2023-08-08 14:10:50", 1),
    ("User activity: Admin accessed settings", "2023-08-08 15:00:10", 0),
    ("Unlabeled log: Further investigation needed", NULL, NULL),
    ("Security alert: Potential malware detected", "2023-08-08 16:25:35", 1),
    ("System shutdown initiated", "2023-08-08 17:10:00", 0);
```

What's Next

- Learn how to Train an Anomaly Detection Model.

6.7.4.5 Train an Anomaly Detection Model

After preparing the data for an anomaly detection model, you can train the model.

This topic has the following sections.

- Before You Begin
- Requirements for Anomaly Detection Training
- Anomaly Detection Options
- Semi-supervised Learning Options
- Log Anomaly Detection Options
- Unsupported Anomaly Detection Options
- Unsupported Routines
- Train a Semi-Supervised Anomaly Detection Model
- Train an Unsupervised Anomaly Detection Model for Logs
- What's Next

Before You Begin

- Review and complete all the tasks to Prepare Data for an Anomaly Detection Model.

Requirements for Anomaly Detection Training

Consider the following based on the type of anomaly detection you are running:

- Set the `task` parameter to `anomaly_detection` for running anomaly detection on table data, or `log_anomaly_detection` for running anomaly detection on log data (MySQL 9.2.2 and later).

- If running an unsupervised model, the `target_column_name` parameter must be set to `NULL`.
- If running a semi-supervised model:
  - The `target_column_name` parameter must specify a column whose only allowed values are 0 (normal), 1 (anomalous), and NULL (unlabeled). All rows are used to train the unsupervised component, while the rows with a value different than NULL are used to train the supervised component.
  - The `experimental` option must be set to `semisupervised`.
- If running anomaly detection on log data (MySQL 9.2.2 and later), the input table can only have the following columns:
  - The column containing the logs.
  - If including logs from different sources, a column containing the source of each log. Identify this column with the `log_source_column` option.
  - If including labeled data, a column identifying the labeled log lines. See Semi-supervised Anomaly Detection to learn more.
  - As of MySQL 9.4.1, at least one column must act as the primary key to establish the temporal order of logs. If the primary key column (or columns) is not one of the previous required columns (log data, source of log, or label), then you must use the `exclude_column_list` option when running `ML_TRAIN` to exclude all primary key columns that don't include required data. See Syntax Examples for Anomaly Detection Training to review relevant examples.
  - If the input table has additional columns to the ones permitted, you must use the `exclude_column_list` option to exclude irrelevant columns.

Anomaly Detection Options

Use the following `JSON` *`options`*:

- `contamination`: Represents an estimate of the percentage of outliers in the training table.
  - The contamination factor is calculated as: estimated number of rows with anomalies/total number of rows in the training table.
  - The contamination value must be greater than 0 and less than 0.5. The default value is 0.01.
- As of MySQL 8.4.0, `model_list` is supported to allow the selection of the Principal Component Analysis (PCA) model and Generalized Local Outlier Factor (GLOF) model. If no option is specified, the default model is Generalized kth Nearest Neighbors (GkNN). Selecting more than one model or an unsupported model produces an error.

Semi-supervised Learning Options

MySQL 9.0.1-u1 introduces the following options to train a semi-supervised anomaly detection model:

- `supervised_submodel_options`: Allows you to set optional override parameters for the supervised model component. The only model supported is `DistanceWeightedKNNClassifier`. The following parameters are supported:
  - `n_neighbors`: Sets the desired k value that checks the k closest neighbors for each unclassified point. The default value is 5 and the value must be an integer greater than 0.
  - `min_labels`: Sets the minimum number of labeled data points required to train the supervised component. If fewer labeled data points are provided during training of the model, `ML_TRAIN` fails. The default value is 20 and the value must be an integer greater than 0.

- `ensemble_score`: This option specifies the metric to use to score the ensemble of unsupervised and supervised components. It identifies the optimal weight between the two components based on the metric. The supported metrics are `accuracy`, `precision`, `recall`, and `f1`. The default metric is `f1`.

Log Anomaly Detection Options

MySQL 9.2.2 introduces anomaly detection for log data. The following options are available as a separate `JSON_OBJECT` named `logad_options`:

- `additional_masking_regex`: Allows you to mask log data by using regular expression in a `JSON_ARRAY`. By default, the following parameters are automatically masked during training and when generating anomaly scores.
  - IP
  - DATETIME
  - TIME
  - HEX
  - IPPORT
  - OCID
- `window_size`: Specifies the maximum number of log lines to be grouped for anomaly detection. The default value is 10.
- `window_stride`: Specifies the stride value to use for segmenting log lines. For example, there is log A, B, C, D, and E. The `window_size` is 3, and the `window_stride` is 2. The first row has log A, B, and C. The second row has log C, D, and E. If this value is equal to `window_size`, there is no overlapping of log segments. The default value is 3.
- `log_source_column`: Specifies the column name that contains the source identifier of the respective log lines. Log lines are grouped according to their respective source (for example, logs from multiple MySQL databases that are in the same table). By default, all log lines are assumed to be from the same source.
- `embedding_model`: Available as of MySQL 9.5.0. The embedding model used to extract semantic features from log data. To review supported embedding models in MySQL HeatWave, run the following query: `SELECT sys.ML_LIST_LLMS();` and see models that have `capabilities` with `TEXT_EMBEDDINGS`. The default value is `multilingual-e5-small`. Using an embedding model causes higher memory usage. If you set this to `NULL`, then you cannot also set `keyword_model` to `NULL`.
- `keyword_model`: Available as of MySQL 9.5.0. The keyword feature extractor used to extract keyword features from log data. The available options are `tf-idf` and `NULL`. The default value is `tf-idf`. If you set this to `NULL`, then you cannot also set `embedding_model` to `NULL`.

Unsupported Anomaly Detection Options

The following options are not supported for anomaly detection:

- `exclude_model_list`
- `optimization_metric`
- Before MySQL 8.4.0, `model_list` is not supported because the only supported algorithm model is Generalized kth Nearest Neighbors (GkNN). As of MySQL 8.4.0, `model_list` is supported to allow the selection of the Principal Component Analysis (PCA) model and Generalized Local Outlier Factor (GLOF) model.

Unsupported Routines

You cannot run the following routines for a trained anomaly detection model:

- `ML_EXPLAIN`
- `ML_EXPLAIN_ROW`
- `ML_EXPLAIN_TABLE`
- `ML_PREDICT_ROW` (only for anomaly detection for logs)

Train a Semi-Supervised Anomaly Detection Model

Train the model with the `ML_TRAIN` routine and use the `credit_card_train` table previously created. Before training the model, it is good practice to define the model handle instead of automatically creating one. See Define Model Handle.

1. Optionally, set the value of the session variable, which sets the model handle to this same value.

   ```
   mysql> SET @variable = 'model_handle';
   ```

   Replace *`@variable`* and *`model_handle`* with your own definitions. For example:

   ```
   mysql> SET @semi_supervised_model='anomaly_detection_semi_supervised_use_case';
   ```

   The model handle is set to `anomaly_detection_semi_supervised_use_case`.

2. Run the `ML_TRAIN` routine.

   ```
   mysql> CALL sys.ML_TRAIN('table_name', 'target_column_name', JSON_OBJECT('task', 'task_name'), model
   ```

   Replace *`table_name`*, *`target_column_name`*, *`task_name`*, and *`model_handle`* with your own values.

   The following example runs `ML_TRAIN` on the training dataset previously created.

   ```
   mysql> CALL sys.ML_TRAIN('anomaly_data.credit_card_train', "target",
                            CAST('{"task": "anomaly_detection", "experimental": {"semisupervised": {}}
                            @semi_supervised_model);
   ```

   Where:

   - `anomaly_data.credit_card_train` is the fully qualified name of the table that contains the training dataset (*`database_name.table_name`*).
   - `target` is the name of the target column, which contains ground truth values to use for semi-supervised learning.
   - `CAST('{"task": "anomaly_detection", "experimental": {"semisupervised": {}}}' as JSON)` specifies the machine learning task type. The `experimental` parameter is required to use a semi-supervised learning model. All default values are used for semi-supervised learning.
   - `@semi_supervised_model` is the session variable previously set that defines the model handle to the name defined by the user: `anomaly_detection_semi_supervised_use_case`. If you do not define the model handle before training the model, the model handle is automatically generated, and the session variable only stores the model handle for the duration of the connection. User variables are written as *`@var_name`*. Any valid name for a user-defined variable is permitted. See Work with Model Handles to learn more.

3. When the training operation finishes, the model handle is assigned to the `@semi_supervised_model` session variable, and the model is stored in the model catalog. View

the entry in the model catalog with the following query. Replace *user1* with your MySQL account name.

```
mysql> SELECT model_id, model_handle, train_table_name FROM ML_SCHEMA_user1.MODEL_CATALOG
       WHERE model_handle = 'anomaly_detection_semi_supervised_use_case';
+----------+-----------------------------------------------+-----------------------------------+
| model_id | model_handle                                  | train_table_name                  |
+----------+-----------------------------------------------+-----------------------------------+
|        3 | anomaly_detection_semi_supervised_use_case    | anomaly_data.credit_card_train    |
+----------+-----------------------------------------------+-----------------------------------+
```

Train an Unsupervised Anomaly Detection Model for Logs

Train the model with the `ML_TRAIN` routine and use the `training_data` table previously created. Before training the model, it is good practice to define the model handle instead of automatically creating one. See Define Model Handle.

1. As of MySQL 9.5.0, you have the option to select the keyword feature extractor and embedding model for training the model. See Anomaly Detection for Logs to learn more. Run the following query to confirm available models that have `capabilities` with `TEXT_EMBEDDINGS`.

   ```
   mysql> SELECT sys.ML_LIST_LLMS();
   +-----------------------------------------------------------------------------------------------------------
   | sys.ML_LIST_LLMS()
   +-----------------------------------------------------------------------------------------------------------
   | [{"model_id": "llama3.1-8b-instruct-v1", "provider": "HeatWave", "capabilities": ["GENERATION"], "defa
   | {"model_id": "llama3.2-1b-instruct-v1", "provider": "HeatWave", "capabilities": ["GENERATION"], "defau
   | {"model_id": "llama3.2-3b-instruct-v1", "provider": "HeatWave", "capabilities": ["GENERATION"], "defau
   | {"model_id": "mistral-7b-instruct-v3", "provider": "HeatWave", "capabilities": ["GENERATION"], "defaul
   | {"model_id": "all_minilm_l12_v2", "provider": "HeatWave", "capabilities": ["TEXT_EMBEDDINGS"], "defaul
   | {"model_id": "multilingual-e5-small", "provider": "HeatWave", "capabilities": ["TEXT_EMBEDDINGS"], "de
   +-----------------------------------------------------------------------------------------------------------
   ```

   The output displays two compatible models: `all_minilm_l12_v2` and `multilingual-e5-small`.

2. Optionally, set the value of the session variable, which sets the model handle to this same value.

   ```
   mysql> SET @variable = 'model_handle';
   ```

   Replace *@variable* and *model_handle* with your own definitions. For example:

   ```
   mysql> SET @unsupervised_log_model='anomaly_detection_log_use_case';
   ```

   The model handle is set to `anomaly_detection_log_use_case`.

3. Run the `ML_TRAIN` routine.

   ```
   mysql> CALL sys.ML_TRAIN('table_name', 'target_column_name', JSON_OBJECT('task', 'task_name'), model_ha
   ```

   Replace *table_name*, *target_column_name*, *task_name*, and *model_handle* with your own values.

   The following example runs `ML_TRAIN` on the training dataset previously created.

   ```
   mysql> CALL sys.ML_TRAIN('anomaly_log_data.training_data', NULL,
                            JSON_OBJECT('task', 'log_anomaly_detection',
                                        'exclude_column_list', JSON_ARRAY('log_id', 'timestamp', 'target'
                                        'logad_options',
                                        JSON_OBJECT('embedding_model', 'all_minilm_l12_v2', 'keyword_mode
   ```

   Where:

   - `anomaly_log_data.training_data` is the fully qualified name of the table that contains the training dataset (*database_name.table_name*).

- `NULL` is set for the target column because it is an unsupervised learning model, so no labeled data is used to train the model.
- `JSON_OBJECT('task', 'log_anomaly_detection'` specifies the machine learning task type.
- `'exclude_column_list', JSON_ARRAY('log_id', 'timestamp', 'target')` sets the required options to run the model for anomaly detection on logs. The columns `log_id` and `timestamp` are excluded because they are not any of the required columns for training. See Requirements for Anomaly Detection Training to learn more. The `target` column is excluded because it is an unsupervised learning model.
- `'embedding_model', 'all_minilm_l12_v2` sets the embedding model to one of the models previously confirmed as available for training the model.
- `'keyword_model', 'tf-idf'` sets the keyword feature extractor for training the model.
- `@unsupervised_log_model` is the session variable previously set that defines the model handle to the name defined by the user: `anomaly_detection_log_use_case`. If you do not define the model handle before training the model, the model handle is automatically generated, and the session variable only stores the model handle for the duration of the connection. User variables are written as *`@var_name`*. Any valid name for a user-defined variable is permitted. See Work with Model Handles to learn more.

4. When the training operation finishes, the model handle is assigned to the `@unsupervised_log_model` session variable, and the model is stored in the model catalog. View the entry in the model catalog with the following query. Replace *`user1`* with your MySQL account name.

```
mysql> SELECT model_id, model_handle, train_table_name FROM ML_SCHEMA_user1.MODEL_CATALOG
       WHERE model_handle = 'anomaly_detection_log_use_case';
+----------+-----------------------------------------+-----------------------------------+
| model_id | model_handle                            | train_table_name                  |
+----------+-----------------------------------------+-----------------------------------+
|        4 | anomaly_detection_log_use_case          | anomaly_log_data.training_data    |
+----------+-----------------------------------------+-----------------------------------+
```

What's Next

- Learn how to Generate Predictions for an Anomaly Detection Model
- Review additional Syntax Examples for Anomaly Detection Training

6.7.4.6 Generate Predictions for an Anomaly Detection Model

After training the model, you can generate predictions.

To generate predictions, use the sample data from the two anomaly detection datasets: `credit_card_train` and `training_data`. Both datasets have labeled and unlabeled rows, but only the dataset for semi-supervised learning uses this for training. The other dataset for log data is trained using unsupervised learning. Having labels for both datasets allows you to compare the predictions to the actual values and determine if the predictions are reliable. Once you determine the trained model is reliable for generating predictions, you can start using it on unseen data.

Anomaly detection models produce anomaly scores, which represent the degree to which a data point deviates from the expected normal behavior. Higher scores indicate a greater degree of abnormality, potentially signaling an anomaly that warrants further investigation. In the results, `is_anomaly` generates a value of 1 for an anomaly, or 0 for normal. The `normal` value represents the degree to which the row of data or log segment exhibits normal behavior. The `anomaly` value represents the degree to which the row of data or log segment exhibits anomalous behavior.

To detect anomalies, run the `ML_PREDICT` routines on data with the same columns as the training model.

- To detect anomalies in row data, you can run the `ML_PREDICT_ROW` or `ML_PREDICT_TABLE` routines.
- To detect anomalies in log data (MySQL 9.2.2 and later), you can only run the `ML_PREDICT_TABLE` routine.

This topic has the following sections.

- Before You Begin
- Requirements for Generating Predictions
- Anomaly Detection Model Options
- Options for Anomaly Detection on Log Data
- Generate Predictions for a Semi-Supervised Anomaly Detection Model
- Generate Predictions for an Unsupervised Anomaly Detection Model on Log Data
- What's Next

Before You Begin

Complete the following tasks:

- Prepare Data for an Anomaly Detection Model
- Train an Anomaly Detection Model.

Requirements for Generating Predictions

As of MySQL 9.4.1, if you run `ML_PREDICT_TABLE` with the `log_anomaly_detection` task, at least one column must act as the primary key to establish the temporal order of logs.

Anomaly Detection Model Options

The threshold you set on anomaly detection models determines which rows in the output table are labeled as anomalies. The value for the threshold is the degree to which a row of data or log segment is considered for anomaly detection. Any sample with an anomaly score above the threshold is classified an anomaly.

There are two ways to set threshold values for anomaly detection models.

Set the Contamination Value

You can set the `contamination` option for the `ML_TRAIN` routine. This option uses the following calculation to set the threshold: (1 - `contamination`)-th percentile of all the anomaly scores.

The default `contamination` value is 0.01. The default `threshold` value based on the default `contamination` value is the 0.99-th percentile of all the anomaly scores.

Set the Threshold Value

You can set the `threshold` option for the `ML_PREDICT_TABLE`, `ML_PREDICT_ROW`, and `ML_SCORE` routines. The value must be greater than 0 and less than 1.

If no value is set for the `threshold` option, the value set for the `contamination` option in the `ML_TRAIN` routine determines the threshold.

The following additional options are available:

- An alternative to `threshold` is `topk`. The results include the top K rows with the highest anomaly scores. The `ML_PREDICT_TABLE` and `ML_SCORE` routines include the `topk` option, which is an integer between 1 and the table length.
- `ML_SCORE` includes an options parameter in `JSON` format. The options are `threshold` and `topk`.
- When running a semi-supervised model, the `ML_PREDICT_ROW`, `ML_PREDICT_TABLE`, and `ML_SCORE` routines have the `supervised_submodel_weight` option. It allows you to override the `ensemble_score` weighting estimated during `ML_TRAIN` with a new value. The value must be greater than 0 and less than 1.

Options for Anomaly Detection on Log Data

When you run anomaly detection on log data (MySQL 9.2.2 and later), you have the option to leverage MySQL HeatWave GenAI for textual summaries of the results. To create summaries, use the following options:

- `summarize_logs`: Enable summaries by setting this to `TRUE`. If enabled, summaries are generated for log segments that are labeled as an anomaly or exceed the value set for the `summary_threshold`.
- `summary_threshold`: Determines the rows in the output table that are summarized. This does not affect how the `contamination` and `threshold` options determine anomalies. You can set a value greater than 0 and less than 1. The default value is `NULL`.

  Summaries are generated for the following:

  - All rows labeled as anomalies.
  - If a value is set for `summary_threshold`, any non-anomaly rows that exceed the value of the `summary_threshold`.

  If the default `NULL` value is used for `summary_threshold`, then only rows labeled as anomalies are summarized.

Note

Enabling the `summary_threshold` option and setting a very low threshold value can potentially lead to a high number of summaries being generated, which may substantially increase the time required to generate output tables.

Generate Predictions for a Semi-Supervised Anomaly Detection Model

1. If not already done, load the model. You can use the session variable for the model that is valid for the duration of the connection. Alternatively, you can use the model handle previously set. For the option to set the user name, you can set it to `NULL`.

   The following example uses the session variable.

   ```
   mysql> CALL sys.ML_MODEL_LOAD(@semi_supervised_model, NULL);
   ```

   The following example uses the model handle.

   ```
   mysql> CALL sys.ML_MODEL_LOAD('anomaly_detection_semi_supervised_use_case', NULL);
   ```

2. Make predictions for the test dataset by using the `ML_PREDICT_TABLE` routine.

   ```
   mysql> CALL sys.ML_PREDICT_TABLE(table_name, model_handle, output_table_name), [options]);
   ```

   Replace *`table_name`*, *`model_handle`*, and *`output_table_name`* with your own values. Add *`options`* as needed.

As of MySQL 9.4.1, you have the option to specify the input table and output table as the same table if specific conditions are met. See Input Tables and Output Tables to learn more.

The following example runs `ML_PREDICT_TABLE` on the testing dataset previously created.

```
mysql> CALL sys.ML_PREDICT_TABLE('anomaly_data.credit_card_train', @semi_supervised_model,
                                 'anomaly_data.credit_card_predictions_semi', JSON_OBJECT('threshold',
```

Where:

- `anomaly_data.credit_card_train` is the fully qualified name of the input table that contains the data to generate predictions for (*`database_name.table_name`*).
- `@model` is the session variable for the model handle.
- `anomaly_data.credit_card_predictions_semi` is the fully qualified name of the output table with predictions (*`database_name.table_name`*).
- `JSON_OBJECT('threshold', 0.55)` sets a threshold value of 55%, which means any row that generates an anomaly score of over 55% is labeled as an anomaly.

3. Query the `target` and `ml_results` columns from the output table. This allows you to compare the real value with the generated anomaly prediction. Review `is_anomaly` to see if the row is labeled as an anomaly (1) or normal (0). Review the anomaly score for each prediction next to `normal` and `anomaly`. If needed, you can also query all the columns from the table (`SELECT * FROM credit_card_predictions_semi`) to review all the data at once.

```
mysql> SELECT target, ml_results FROM credit_card_predictions_semi;
+--------+-------------------------------------------------------------------------------------------+
| target | ml_results                                                                                |
+--------+-------------------------------------------------------------------------------------------+
|      0 | {"predictions": {"is_anomaly": 1}, "probabilities": {"normal": 0.43, "anomaly": 0.57}}    |
|      1 | {"predictions": {"is_anomaly": 1}, "probabilities": {"normal": 0.4377, "anomaly": 0.5623}} |
|      0 | {"predictions": {"is_anomaly": 0}, "probabilities": {"normal": 0.8677, "anomaly": 0.1323}} |
|   NULL | {"predictions": {"is_anomaly": 0}, "probabilities": {"normal": 0.8652, "anomaly": 0.1348}} |
|      1 | {"predictions": {"is_anomaly": 0}, "probabilities": {"normal": 0.4921, "anomaly": 0.5079}} |
|      0 | {"predictions": {"is_anomaly": 0}, "probabilities": {"normal": 0.8487, "anomaly": 0.1513}} |
|   NULL | {"predictions": {"is_anomaly": 0}, "probabilities": {"normal": 0.7622, "anomaly": 0.2378}} |
|      1 | {"predictions": {"is_anomaly": 0}, "probabilities": {"normal": 0.57, "anomaly": 0.43}}    |
|      0 | {"predictions": {"is_anomaly": 0}, "probabilities": {"normal": 0.8317, "anomaly": 0.1683}} |
|   NULL | {"predictions": {"is_anomaly": 0}, "probabilities": {"normal": 0.8539, "anomaly": 0.1461}} |
|      0 | {"predictions": {"is_anomaly": 0}, "probabilities": {"normal": 0.9264, "anomaly": 0.0736}} |
|      1 | {"predictions": {"is_anomaly": 1}, "probabilities": {"normal": 0.4079, "anomaly": 0.5921}} |
|      0 | {"predictions": {"is_anomaly": 0}, "probabilities": {"normal": 0.8379, "anomaly": 0.1621}} |
|   NULL | {"predictions": {"is_anomaly": 0}, "probabilities": {"normal": 0.7971, "anomaly": 0.2029}} |
|      1 | {"predictions": {"is_anomaly": 0}, "probabilities": {"normal": 0.4623, "anomaly": 0.5377}} |
|      0 | {"predictions": {"is_anomaly": 0}, "probabilities": {"normal": 0.8816, "anomaly": 0.1184}} |
|   NULL | {"predictions": {"is_anomaly": 0}, "probabilities": {"normal": 0.8267, "anomaly": 0.1733}} |
|      0 | {"predictions": {"is_anomaly": 0}, "probabilities": {"normal": 0.8816, "anomaly": 0.1184}} |
|      1 | {"predictions": {"is_anomaly": 0}, "probabilities": {"normal": 0.4661, "anomaly": 0.5339}} |
|   NULL | {"predictions": {"is_anomaly": 0}, "probabilities": {"normal": 0.8202, "anomaly": 0.1798}} |
|      0 | {"predictions": {"is_anomaly": 0}, "probabilities": {"normal": 0.9113, "anomaly": 0.0887}} |
|      1 | {"predictions": {"is_anomaly": 0}, "probabilities": {"normal": 0.5078, "anomaly": 0.4922}} |
|      0 | {"predictions": {"is_anomaly": 0}, "probabilities": {"normal": 0.9378, "anomaly": 0.0622}} |
|   NULL | {"predictions": {"is_anomaly": 0}, "probabilities": {"normal": 0.8963, "anomaly": 0.1037}} |
|      0 | {"predictions": {"is_anomaly": 0}, "probabilities": {"normal": 0.5262, "anomaly": 0.4738}} |
|      1 | {"predictions": {"is_anomaly": 0}, "probabilities": {"normal": 0.5002, "anomaly": 0.4998}} |
|   NULL | {"predictions": {"is_anomaly": 0}, "probabilities": {"normal": 0.8767, "anomaly": 0.1233}} |
|      0 | {"predictions": {"is_anomaly": 0}, "probabilities": {"normal": 0.8878, "anomaly": 0.1122}} |
|      1 | {"predictions": {"is_anomaly": 0}, "probabilities": {"normal": 0.4661, "anomaly": 0.5339}} |
|      0 | {"predictions": {"is_anomaly": 0}, "probabilities": {"normal": 0.9037, "anomaly": 0.0963}} |
|   NULL | {"predictions": {"is_anomaly": 1}, "probabilities": {"normal": 0.4171, "anomaly": 0.5829}} |
+--------+-------------------------------------------------------------------------------------------+
31 rows in set (0.0479 sec)
```

To learn more about generating predictions for one or more rows of data, see Generate Predictions for a Row of Data.

Generate Predictions for an Unsupervised Anomaly Detection Model on Log Data

1. If not already done, load the model. You can use the session variable for the model that is valid for the duration of the connection. Alternatively, you can use the model handle previously set. For the option to set the user name, you can set it to `NULL`.

   The following example uses the session variable.

   ```
   mysql> CALL sys.ML_MODEL_LOAD(@unsupervised_log_model, NULL);
   ```

   The following example uses the model handle.

   ```
   mysql> CALL sys.ML_MODEL_LOAD('anomaly_detection_log_use_case', NULL);
   ```

2. Make predictions for the test dataset by using the `ML_PREDICT_TABLE` routine.

   ```
   mysql> CALL sys.ML_PREDICT_TABLE(table_name, model_handle, output_table_name), [options]);
   ```

   Replace *`table_name`*, *`model_handle`*, and *`output_table_name`* with your own values. Add *`options`* as needed.

   The following example runs `ML_PREDICT_TABLE` on the testing dataset previously created.

   ```
   mysql> CALL sys.ML_PREDICT_TABLE('anomaly_log_data.testing_data', @unsupervised_log_model, 'anomaly_
                                    JSON_OBJECT('logad_options', JSON_OBJECT('summarize_logs', TRUE)))
   ```

   Where:

   - `anomaly_log_data.testing_data` is the fully qualified name of the input table that contains the data to generate predictions for (*`database_name.table_name`*).
   - `@model` is the session variable for the model handle.
   - `anomaly_log_data.log_predictions_unsupervised` is the fully qualified name of the output table with predictions (*`database_name.table_name`*).
   - `JSON_OBJECT('logad_options', JSON_OBJECT('summarize_logs', TRUE))` enables the textual summaries generated by MySQL HeatWave GenAI. No threshold is set for the summaries, so the default value of any labeled anomaly generates a summary.

3. Query the output table and compare the real value with the generated anomaly prediction. Use `\G` to view the output in an easily readable format.

   ```
   mysql> SELECT * FROM log_predictions_unsupervised\G
   *************************** 1. row ***************************
                   id: 1
   parsed_log_segment: User login failed: Invalid credentials Server response time increased Normal dat
                                          Error log: Stack trace included User activity: Admin accessed
           ml_results: {"summary": "\nHere is a concise summary of the text:\n\nThe system encountered
                                     A potential malware detection was also triggered, prompting a sec
                                     "index_map": [1, 2, 3, 4, 5, 6, 7, 8, 9, 10], "predictions": {"is
   *************************** 2. row ***************************
                   id: 2
   parsed_log_segment: Unusual network traffic from IP: 10.0.0.5 System update completed successfully E
           ml_results: {"summary": "\nHere is a concise summary:\n\nA system update was completed succe
                                    "index_map": [4, 5, 6, 7, 8, 9, 10], "predictions": {"is_anomaly": 1
   *************************** 3. row ***************************
                   id: 3
   parsed_log_segment: User activity: Admin accessed settings Unlabeled log: Further investigation need
           ml_results: {"summary": "\nAn administrator has accessed the system settings, triggered a se
                                    "index_map": [7, 8, 9, 10], "predictions": {"is_anomaly": 1}, "proba
   *************************** 4. row ***************************
                   id: 4
   parsed_log_segment: System shutdown initiated
   ```

```
        ml_results: {"summary": "\nThe system is shutting down.", "index_map": [10], "predictions": {"i
```

The size of the output table is based on the `window_size` and `window_stride` parameters when the model is trained. Since this use case does not set these parameters, the default values of 10 for `window_size` and 3 for `window_stride` is used. See Log Anomaly Detection Options to learn more.

Review the following in the output table:

- `is_anomaly` to see if the row is labeled as an anomaly (1) or normal (0).
- `normal` and `anomaly` to see the anomaly score for each.
- `index_map` to see which rows in the input table are included in the prediction based on the `window_size` and `window_stride`.
- `summary` to see the generated text summary describing the anomaly.

What's Next

- Learn how to Score an Anomaly Detection Model.

6.7.4.7 Score an Anomaly Detection Model

After generating predictions, you can score the model to assess its reliability. For a list of scoring metrics you can use with anomaly detection models, see Anomaly Detection Metrics. For this use case, you use the test dataset for validation. In a real-world use case, you should use a separate validation dataset that has the target column and ground truth values for the scoring validation. You should also use a larger number of records for training and validation to get a valid score.

To generate a score, the `target_column_name` column must only contain the anomaly scores as an integer: `1` for an anomaly, or `0` for normal.

Before You Begin

Complete the following tasks:

- Prepare Data for an Anomaly Detection Model
- Train an Anomaly Detection Model
- Generate Predictions for an Anomaly Detection Model

Requirements for Scoring Models

As of MySQL 9.4.1, if you run `ML_SCORE` with the `log_anomaly_detection` task, at least one column must act as the primary key to establish the temporal order of logs.

Score a Semi-Supervised Anomaly Detection Model

1. If not already done, load the model. You can use the session variable for the model that is valid for the duration of the connection. Alternatively, you can use the model handle previously set. For the option to set the user name, you can set it to `NULL`.

   The following example uses the session variable.

   ```
   mysql> CALL sys.ML_MODEL_LOAD(@model, NULL);
   ```

   The following example uses the model handle.

   ```
   mysql> CALL sys.ML_MODEL_LOAD('anomaly_detection_semi_supervised_use_case', NULL);
   ```

2. Score the model with the `ML_SCORE` routine and use the `accuracy` metric.

   ```
   mysql> CALL sys.ML_SCORE(table_name, target_column_name, model_handle, metric, score, [options]);
   ```

Replace *table_name*, *target_column_name*, *model_handle*, *metric*, *score* with your own values.

The following example runs ML_SCORE on the testing dataset previously created.

```
mysql> CALL sys.ML_SCORE('anomaly_data.credit_card_test', 'target', 'anomaly_detection_semi_supervis
```

Where:

- anomaly_data.credit_card_test is the fully qualified name of the validation dataset.
- target is the target column name with ground truth values.
- 'anomaly_detection_semi_supervised_use_case' is the model handle for the trained model.
- accuracy is the selected scoring metric.
- @anomaly_score is the session variable name for the score value.
- NULL means that no other options are defined for the routine.

3. Retrieve the score by querying the @score session variable.

```
mysql> SELECT @anomaly_score;
+--------------------+
| @anomaly_score     |
+--------------------+
| 0.6499999761581421 |
+--------------------+
1 row in set (0.0481 sec)
```

4. If done working with the model, unload it with the ML_MODEL_UNLOAD routine.

```
mysql> CALL sys.ML_MODEL_UNLOAD('anomaly_detection_semi_supervised_use_case');
```

To avoid consuming too much memory, it is good practice to unload a model when you are finished using it.

Score an Unsupervised Anomaly Detection Model for Log Data

Even though you score an unsupervised model, you must provide a labeled dataset for generating a score.

1. If not already done, load the model. You can use the session variable for the model that is valid for the duration of the connection. Alternatively, you can use the model handle previously set. For the option to set the user name, you can set it to NULL.

The following example uses the session variable.

```
mysql> CALL sys.ML_MODEL_LOAD(@model, NULL);
```

The following example uses the model handle.

```
mysql> CALL sys.ML_MODEL_LOAD('anomaly_detection_log_use_case', NULL);
```

2. Score the model with the ML_SCORE routine and use the accuracy metric.

```
mysql> CALL sys.ML_SCORE(table_name, target_column_name, model_handle, metric, score, [options]);
```

Replace *table_name*, *target_column_name*, *model_handle*, *metric*, *score* with your own values.

The following example runs ML_SCORE on the testing dataset previously created.

```
mysql> CALL sys.ML_SCORE('anomaly_log_data.testing_data', 'target', 'anomaly_detection_log_use_case', '
```

Where:

- `anomaly_log_data.testing_data` is the fully qualified name of the validation dataset.
- `target` is the target column name with ground truth values.
- `'anomaly_detection_log_use_case'` is the model handle for the trained model.
- `f1` is the selected scoring metric.
- `@anomaly_log_score` is the session variable name for the score value.
- `NULL` means that no other options are defined for the routine.

3. Retrieve the score by querying the @score session variable.

```
mysql> SELECT @anomaly_log_score;
+--------------------+
| @anomaly_log_score |
+--------------------+
| 0.8571428656578064 |
+--------------------+
1 row in set (0.0452 sec)
```

4. If done working with the model, unload it with the `ML_MODEL_UNLOAD` routine.

```
mysql> CALL sys.ML_MODEL_UNLOAD('anomaly_detection_log_use_case');
```

To avoid consuming too much memory, it is good practice to unload a model when you are finished using it.

What's Next

- Review other Machine Learning Use Cases.

6.7.5 Generate Recommendations

Recommendation models find patterns in user behavior to recommend products and users based on prior behavior and preferences. Common examples include a streaming service recommending movies and shows based on past viewing history, or an online shopping site recommending products based on prior purchases.

The main goal of recommendation models is to recommend either items that a user will like, or recommend users who may like a specific item. MySQL HeatWave AutoML includes recommendation models that can recommend the following:

- The rating that a user will give to an item.
- Users who will like an item.
- Items that a user will like.
- Identify similar items.
- Identify similar users.

The following tasks use a dataset generated by OCI GenAI using Meta Llama Models. The recommendation use-case is to create a machine learning model based on users giving a rating of 1 to 10 for different items.

To generate your own datasets to create machine learning models in MySQL HeatWave, learn how to Generate Text-Based Content.

Note

Datasets were generated using Meta Llama models. Your use of this Llama model is subject to your Oracle agreements and this Llama license agreement: https://downloads.mysql.com/docs/LLAMA_31_8B_INSTRUCT-license.pdf.

6.7.5.1 Recommendation Task Types

You can create recommendation models based on either explicit or implicit feedback. See Recommendation Models to review models that support either implicit or explicit feedback.

Recommendation Models with Explicit Feedback

Recommendation models that use explicit feedback collect data on users that directly provide ratings on items. The user ratings can be positive or negative. The recommendation models then use the feedback to generate predicted ratings for users and items. The ratings are specific values, and the higher the value, the better the rating.

Recommendation Models with Implicit Feedback

Recommendation models that use implicit feedback collect data on users' behavior, such as past purchases, clicks, and view times. Users do not have to explicitly express their taste about an item. When a user interacts with an item, the implication is that they prefer it to an item that they do not interact with. Therefore, only positive observations are available. The non-observed user-item interactions are a blend of negative feedback (the user doesn't like the item) or missing values (the user might be interested in the item). The recommendation model generates rankings for users and items. Rankings are a comparative measure, and the lower the value, the better the ranking. Because A is better than B, the ranking for A has a lower value than the ranking for B. MySQL HeatWave AutoML derives rankings based on ratings from implicit feedback for all ratings that are at or above the feedback threshold.

Implicit feedback data can be in the following formats:

- Unary data: Only records if an interaction occurred or not. This type of data often uses a value of 1 to represent an interaction, such as a click or view. Non-interactions can be represented by a value of 0 or missing values.
- Binary data: Explicitly categorizes interactions as positive or negative, such as users expressing likes or dislikes.
- Numerical data: Provides more granular information about the interaction, such as how long a user watched a video or how many times a user listened to a song. If numerical data is used for implicit feedback, it is important to set the `feedback_threshold` option during training to distinguish what constitutes positive feedback. This threshold determines what value is equivalent to a positive interaction. For example, if users are tracked by how many times they have interacted with an item, you might set the `feedback_threshold` with a value of 3, which means that positive feedback is represented by users that interact with the item more than three times.

Content-Based Recommendation Models

Content-based recommendation models allow you to include item and user descriptions in the input of the recommendation model. This helps the model provide more accurate representations of items and users. When training a content-based recommendation model, you can use the following models:

- Collaborative Topic Regression (CTR): This model combines the ideas of matrix factorization models and topic modeling using Latent Dirichlet Allocation (LDA).
- TwoTower: As of MySQL 9.5.0, the default training model. See Recommendation Training Models to learn more.

What's Next

- Learn how to Prepare Data for a Recommendation Model.

6.7.5.2 Prepare Data for a Recommendation Model

This topic describes how to prepare the data to use for a recommendation machine learning model using explicit feedback. It uses a data sample generated by OCI GenAI. To prepare the data for this use case, you set up a training dataset and a testing dataset. The training dataset has 86 records, and the testing dataset has 40 records. In a real-life use case, you should prepare a larger amount of records for training and testing, and ensure the predictions are valid and reliable before testing on unlabeled data. To ensure reliable predictions, you should create an additional validation dataset. You can reserve 20% of the records in the training dataset to create the validation dataset.

You have the option to automatically Prepare Training and Testing Datasets with your own data by using the `TRAIN_TEST_SPLIT` routine.

Before You Begin

- Learn how to Prepare Data.

Prepare Data

To prepare the data for the recommendation model:

1. Connect to your MySQL HeatWave Database System.
2. Create and use the database to store the data.

```
mysql> CREATE DATABASE recommendation_data;
mysql> USE recommendation_data;
```

3. Create the table to insert the sample data into. This is the training dataset. The columns for users and items (`user_id` and `item_id`), must be in string data type.

```
mysql> CREATE TABLE training_dataset (
    user_id VARCHAR(3),
    item_id VARCHAR(3),
    rating DECIMAL(3, 1),
    PRIMARY KEY (user_id, item_id)
);
```

4. Insert the sample data to train into the table. Copy and paste the following commands.

```
INSERT INTO training_dataset (user_id, item_id, rating) VALUES
    (1, 1, 5.0),
    (1, 3, 8.0),
    (1, 5, 2.5),
    (1, 7, 6.5),
    (1, 9, 4.0),
    (1, 11, 7.5),
    (1, 13, 3.0),
    (1, 15, 9.0),
    (1, 17, 1.5),
    (1, 19, 5.5),
    (2, 2, 4.5),
    (2, 4, 7.5),
    (2, 6, 2.0),
    (2, 8, 5.5),
    (2, 10, 9.0),
    (2, 12, 3.5),
    (2, 14, 6.0),
    (2, 16, 1.0),
    (2, 18, 4.5),
    (2, 20, 8.5),
    (3, 1, 3.5),
    (3, 4, 6.5),
    (3, 7, 2.5),
    (3, 9, 5.0),
    (3, 11, 8.5),
    (3, 13, 1.0),
```

```
    (3, 15, 4.0),
    (3, 17, 7.0),
    (3, 19, 2.5),
    (4, 2, 5.5),
    (4, 5, 8.5),
    (4, 8, 3.0),
    (4, 10, 6.5),
    (4, 12, 9.5),
    (4, 14, 2.0),
    (4, 16, 4.5),
    (4, 18, 7.5),
    (5, 3, 7.0),
    (5, 6, 1.5),
    (5, 8, 4.0),
    (5, 11, 6.0),
    (5, 13, 8.0),
    (5, 15, 2.5),
    (5, 17, 5.5),
    (5, 19, 9.0),
    (6, 1, 4.5),
    (6, 4, 7.5),
    (6, 6, 3.0),
    (6, 9, 5.5),
    (6, 12, 8.0),
    (6, 14, 1.5),
    (6, 16, 4.0),
    (6, 18, 6.5),
    (7, 2, 6.0),
    (7, 5, 3.5),
    (7, 7, 5.0),
    (7, 10, 7.5),
    (7, 12, 2.0),
    (7, 14, 4.5),
    (7, 16, 7.0),
    (7, 18, 9.5),
    (8, 3, 8.5),
    (8, 6, 2.5),
    (8, 8, 5.0),
    (8, 11, 3.5),
    (8, 13, 6.5),
    (8, 15, 1.0),
    (8, 17, 4.5),
    (8, 19, 7.0),
    (9, 2, 5.0),
    (9, 5, 8.0),
    (9, 7, 1.5),
    (9, 10, 4.0),
    (9, 12, 6.5),
    (9, 14, 9.0),
    (9, 16, 2.5),
    (9, 18, 5.5),
    (10, 1, 6.5),
    (10, 4, 3.0),
    (10, 6, 5.5),
    (10, 8, 8.0),
    (10, 11, 2.0),
    (10, 13, 4.5),
    (10, 15, 7.0),
    (10, 17, 9.5),
    (10, 19, 1.5);
```

5. Create the table to use for generating predictions. This is the test dataset. It has the same columns as the training dataset.

```
mysql> CREATE TABLE testing_dataset (
    user_id VARCHAR(3),
    item_id VARCHAR(3),
    rating DECIMAL(3, 1),
    PRIMARY KEY (user_id, item_id)
);
```

6. Insert the sample data to test into the table. Copy and paste the following commands.

```
INSERT INTO testing_dataset (user_id, item_id, rating) VALUES
    (1, 2, 4.0),
    (1, 4, 7.0),
    (1, 6, 1.5),
    (1, 8, 3.5),
    (2, 1, 5.0),
    (2, 3, 8.0),
    (2, 5, 2.5),
    (2, 7, 6.5),
    (3, 2, 3.5),
    (3, 5, 6.5),
    (3, 8, 2.5),
    (3, 18, 7.0),
    (4, 1, 5.5),
    (4, 3, 8.5),
    (4, 6, 2.0),
    (4, 7, 5.5),
    (5, 2, 7.0),
    (5, 4, 1.5),
    (5, 6, 4.0),
    (5, 12, 5.0),
    (6, 3, 6.0),
    (6, 5, 1.5),
    (6, 7, 4.5),
    (6, 8, 7.0),
    (7, 1, 6.5),
    (7, 4, 3.0),
    (7, 5, 5.5),
    (7, 9, 8.0),
    (8, 2, 8.5),
    (8, 4, 2.5),
    (8, 6, 5.0),
    (8, 9, 3.5),
    (9, 1, 5.0),
    (9, 3, 8.0),
    (9, 7, 2.5),
    (9, 8, 5.5),
    (10, 2, 6.5),
    (10, 5, 3.0),
    (10, 6, 5.5),
    (10, 18, 1.5);
```

What's Next

- Learn how to Train a Recommendation Model.

6.7.5.3 Train a Recommendation Model

After preparing the data for a recommendation model, you can train the model.

This topic has the following sections.

- Before You Begin
- Requirements for Recommendation Training
- Options for All Recommendation Model Types
- Recommendation Training Models
- Options for Recommendation Models with Explicit Feedback
- Options for Recommendation Models with Implicit Feedback
- Options for Content-Based Recommendation Models
- Unsupported Routines
- Train Model

- What's Next

Before You Begin

- Review and complete all the tasks to Prepare Data for a Recommendation Model.

Requirements for Recommendation Training

Define the following as required to train a recommendation model.

- Set the `task` parameter to `recommendation` to train a recommendation model.
- `users`: Specifies the column name corresponding to the user IDs. Values in this column must be in a `STRING` data type, otherwise an error is returned during training.
- `items`: Specifies the column name corresponding to the item IDs. Values in this column must be in a `STRING` data type, otherwise an error is returned during training.

If the `users` or `items` column contains `NULL` values, the corresponding rows are dropped and are not be considered during training.

Options for All Recommendation Model Types

See Common ML_TRAIN Options to view available options for training recommendation models.

Recommendation Training Models

As of MySQL 9.5.0:

- The default recommendation training model is the `TwoTower` model with Pytorch. In earlier versions, MySQL HeatWave AutoML selects the best model for the task based on the available models. This is affected by the `model_list` and `exclude_model_list` training options. Review the list of available Recommendation Models.
- You cannot add the `TwoTower` model with the `model_list` option. Adding the model generates an error. The `TwoTower` model is already set as the default recommendation model if `model_list` is not specified.

The `TwoTower` model is a deep learning recommender system pipeline that provides enhanced quality and faster speed than other recommender training models. It uses user-item interactions and user-item features to train embedding vectors for users and items, which allows for quicker predictions. To enable faster predictions, the model generates tables with embeddings for users and items. The tables are used as a representation for each user and item. The model also generates a table of interactions that stores each interaction between a user and item. You can view these tables in the `ML_SCHEMA_`*`MySQL_username`* database. For example, `ML_SCHEMA_user1.abc123_users`, `ML_SCHEMA_user1.abc123_items`, and `ML_SCHEMA_user1.abc123_interactions`. Providing `item_metadata` and `user_metadata` is optional for the `TwoTower` model.

To review limitations related to the `TwoTower` model, see Routine and Query Limitations and Other Limitations.

Options for Recommendation Models with Explicit Feedback

Define the following `JSON` options to train a recommendation model with explicit feedback. To learn more about recommendation models, see Recommendation Model Types.

- `feedback`: Set to `explicit`. If not set, the default value is `explicit`.

Options for Recommendation Models with Implicit Feedback

Define the following `JSON` options to train a recommendation model with implicit feedback. To learn more about recommendation models, see Recommendation Model Types.

- `feedback`: Set to `implicit`.

- `feedback_threshold`: The feedback threshold for a recommendation model that uses implicit feedback. It represents the threshold required to be considered positive feedback. For example, if numerical data records the number of times users interact with an item, you might set a threshold with a value of 3. This means users would need to interact with an item more than three times to be considered positive feedback.

Options for Content-Based Recommendation Models

Define the following `JSON` options to train a content-based recommendation model. To learn more about recommendation models, see Recommendation Model Types.

- `item_metadata`: As of MySQL 9.5.0, it defines the table that has item descriptions. It is a JSON object that has the `table_name` option as a key, which specifies the table that has item descriptions. One column must be the same as the `item_id` in the input table.
- `user_metadata`: As of MySQL 9.5.0, it defines the table that has user descriptions. It is a JSON object that has the `table_name` option as a key, which specifies the table that has user descriptions. One column must be the same as the `user_id` in the input table.
  - `table_name`: To be used with the `item_metadata` and `user_metadata` options. It specifies the table name that has item or user descriptions. It must be a string in a fully qualified format (schema_name.table_name) that specifies the table name.

Unsupported Routines

You cannot run the following routines for a trained recommendation model:

- `ML_EXPLAIN`
- `ML_EXPLAIN_ROW`
- `ML_EXPLAIN_TABLE`

Train Model

Train the model with the `ML_TRAIN` routine and use the `training_data` table previously created. Before training the model, it is good practice to define the model handle instead of automatically creating one. See Define Model Handle.

1. Optionally, set the value of the session variable, which sets the model handle to this same value.

   ```
   mysql> SET @variable = 'model_handle';
   ```

   Replace *`@variable`* and *`model_handle`* with your own definitions. For example:

   ```
   mysql> SET @model='recommendation_use_case';
   ```

   The model handle is set to `recommendation_use_case`.

2. Run the `ML_TRAIN` routine.

   ```
   mysql> CALL sys.ML_TRAIN('table_name', 'target_column_name', JSON_OBJECT('task', 'task_name'), model_ha
   ```

   Replace *`table_name`*, *`target_column_name`*, *`task_name`*, and *`model_handle`* with your own values.

   The following example runs `ML_TRAIN` on the training dataset previously created.

   ```
   mysql> CALL sys.ML_TRAIN('recommendation_data.training_dataset', 'rating',
                            JSON_OBJECT('task', 'recommendation', 'users', 'user_id', 'items', 'item_id')
   ```

   Where:

   - `recommendation_data.training_dataset` is the fully qualified name of the table that contains the training dataset (*`database_name.table_name`*).

- `rating` is the name of the target column, which contains ground truth values (item ratings).
- `JSON_OBJECT('task', 'recommendation', 'users', 'user_id', 'items', 'item_id')` specifies the machine learning task type and defines the `users` and `items` columns. Since no model type is defined, the default value of a recommendation model using explicit feedback is trained.
- `@model` is the session variable previously set that defines the model handle to the name defined by the user: `recommendation_use_case`. If you do not define the model handle before training the model, the model handle is automatically generated, and the session variable only stores the model handle for the duration of the connection. User variables are written as *`@var_name`*. Any valid name for a user-defined variable is permitted. See Work with Model Handles to learn more.

3. When the training operation finishes, the model handle is assigned to the `@model` session variable, and the model is stored in the model catalog. View the entry in the model catalog with the following query. Replace *`user1`* with your MySQL account name.

```
mysql> SELECT model_id, model_handle, train_table_name FROM ML_SCHEMA_user1.MODEL_CATALOG  WHERE mod
+----------+--------------------------------------------+--------------------------------------
| model_id | model_handle                               | train_table_name
+----------+--------------------------------------------+--------------------------------------
|        6 | recommendation_use_case                    | recommendation_data.training_dataset
+----------+--------------------------------------------+--------------------------------------
```

4. As of MySQL 9.5.0, if the model is trained with the `TwoTower` recommendation model, you can view the tables with embeddings and the table of interactions in the `ML_SCHEMA_`*`MySQL_username`* database. To view the names of the generated tables, run the following query. Replace *`user1`* with your MySQL account name.

```
mysql> SHOW TABLES FROM ML_SCHEMA_user1;
+---------------------------+
| Tables_in_ML_SCHEMA_admin |
+---------------------------+
| 3e094aa4ba_interactions   |
| 3e094aa4ba_items          |
| 3e094aa4ba_users          |
| MODEL_CATALOG             |
| MODEL_CATALOG_BACKUP_v3   |
| catalog_version_v1        |
| catalog_version_v2        |
| catalog_version_v3        |
| model_object_catalog      |
+---------------------------+
```

5. Run the following queries to view a sample of each generated table.

```
mysql> SELECT * FROM ML_SCHEMA_user1.3e094aa4ba_users LIMIT 5;
+---------+-----------------------------------------------------------------------------------------
| user_id | embedding_vector
+---------+-----------------------------------------------------------------------------------------
| 1       | 0x543B13BDDD328D3D6B3F6CBEB56B39BEA75052BEAA25C5BDAE894A3D721013BE3324B23D59779D3DD35B80
| 10      | 0xC3551FBEDA3F34BE2FD6583EEC0BC13D650262BE860F1C3D09DBEC3CAFBD803D006CD5BD3C26A5BD5FB9F0
| 2       | 0x73A719BECAD5E33C9C71973E164E203D7C4635BEB7ECB53ADCA1803EF48F553E927B34BEEEFF8DBD32F474
| 3       | 0x520298BEA7516EBE6A7C3C3CAC9CD6BD2296C63DF76ECC3DDF1C01BE79B0193D75F381BED1D4B0BD501863
| 4       | 0x992518BE74C5FDBD4AB0C73DB422B33D45F74EBE81F25FBE653D5A3E7F4134BDDAD13B3ED95E7E3DDA11B3
+---------+-----------------------------------------------------------------------------------------
5 rows in set (0.0409 sec)
mysql> SELECT * FROM ML_SCHEMA_user1.3e094aa4ba_items LIMIT 5;
+---------+-----------------------------------------------------------------------------------------
| item_id | embedding_vector
+---------+-----------------------------------------------------------------------------------------
| 1       | 0xED5A45BEF6D032BE667454BE9DB66FBE62B6F13D392199BE9FE6D63D8718353E1E26CCBC9B6F133EC0F0CE
| 10      | 0x80C883BC1DCE29BE126039BE1817133D70C63D3E3E795CBE62C33F3DDE1D5F3C84264B3D66672A3E427B39
| 11      | 0x105B08BE9084D63D126CD3BC6C0117BCCCBC683D04328B3CEE562A3E18F8FEBCE03F423D507486BD2D2244
| 12      | 0x757894BC9644373D7A1DA63DA3E5ECBD3B978E3D9B3A0B3E4E35343EF921CD3B2CEA753D68D0C3BD65C2F5
| 13      | 0x37DEEFBC545030BC2EF980BD3712923DCC0142BE9C5E143E129309BD4A3E02BE85C001BEA1140E3ED73323
+---------+-----------------------------------------------------------------------------------------
```

```
5 rows in set (0.0419 sec)
mysql> SELECT * FROM ML_SCHEMA_user1.3e094aa4ba_interactions LIMIT 5;
+-------------------+---------+---------+
| _4aad19ca6e_pk_id | user_id | item_id |
+-------------------+---------+---------+
|                 1 | 1       | 1       |
|                 2 | 1       | 11      |
|                 3 | 1       | 13      |
|                 4 | 1       | 15      |
|                 5 | 1       | 17      |
+-------------------+---------+---------+
5 rows in set (0.0454 sec)
```

What's Next

- Learn how to Generate Predictions for a Recommendation Model.
- Review additional Syntax Examples for Recommendation Training.

6.7.5.4 Generate Predictions for a Recommendation Model

After training the model, you can generate predictions. To generate predictions, use the sample data from the `testing_dataset` dataset. `NULL` values for any row in the `users` or `items` columns generates an error.

Before You Begin

Complete the following tasks:

- Prepare Data for a Recommendation Model
- Train a Recommendation Model

Options for Generating Predictions

The *`options`* for `ML_PREDICT_ROW` and `ML_PREDICT_TABLE` include the following:

- `threshold`: The optional threshold that defines positive feedback, and a relevant sample. Only use with ranking metrics. It can be used for either explicit or implicit feedback.
- `topk`: The number of recommendations to provide. The default is `3`.
- `recommend`: Specifies what to recommend. Permitted values are:
  - `ratings`: Predicts ratings that users will give. This is the default value.
  - `items`: Recommends items for users.
  - `users`: Recommends users for items.
  - `users_to_items`: This is the same as `items`.
  - `items_to_users`: This is the same as `users`.
  - `items_to_items`: Recommends similar items for items.
  - `users_to_users`: Recommends similar users for users.
- `remove_seen`: If `true`, the model does not repeat existing interactions from the training table. It only applies to the recommendations `items`, `users`, `users_to_items`, and `items_to_users`.
- `item_metadata`: As of MySQL 9.5.0, it defines the table that has item descriptions. It is a JSON object that has the `table_name` option as a key, which specifies the table that has item descriptions. One column must be the same as the `item_id` in the input table.

- `user_metadata`: As of MySQL 9.5.0, it defines the table that has user descriptions. It is a JSON object that has the `table_name` option as a key, which specifies the table that has user descriptions. One column must be the same as the `user_id` in the input table.
  - `table_name`: To be used with the `item_metadata` and `user_metadata` options. It specifies the table name that has item or user descriptions. It must be a string in a fully qualified format (schema_name.table_name) that specifies the table name.

Generating Predictions with the TwoTower Recommendation Model

If the model is trained with the `TwoTower` recommendation model, keep in mind the following:

- You have the option to specify additional user and item desciptions by using the `item_metadata` and `user_metadata` options.
- If there are missing descriptions for users and items, these missing descriptions are inferred when generating predictions.
- If user and items descriptions are provided for training, they are ignored when generating predictions. Instead, the generated embeddings for the users and items are used to generate predictions.
- The `ML_PREDICT_ROW` routine is not supported.

What's Next

- Learn about the different ways to generate specific recommendations with a recommendation model:
  - Generate Predictions for Ratings and Rankings
  - Generate Item Recommendations for Users
  - Generate User Recommendations for Items
  - Generate Recommendations for Similar Items
  - Generate Recommendations for Similar Users

6.7.5.5 Generate Predictions for Ratings and Rankings

This topic describes how to generate recommendations for either ratings (recommendation model with explicit feedback) or rankings (recommendation model with implicit feedback). If generating a rating, the output predicts the rating the user will give to an item. If generating a ranking, the output is a ranking of the user compared to other users.

- For known users and known items, the output includes the predicted rating or ranking for an item for a given pair of `user_id` and `item_id`.
- For a known user with a new item, the prediction is the global average rating or ranking. The routines can add a user bias if the model includes it.
- For a new user with a known item, the prediction is the global average rating or ranking. The routines can add an item bias if the model includes it.
- For a new user with a new item, the prediction is the global average rating or ranking.

Before You Begin

Review and complete the following tasks:

- Prepare Data for a Recommendation Model
- Train a Recommendation Model
- Generate Predictions for a Recommendation Model

Generate Rating Recommendations

Since the model you previously trained used explicit feedback, you generate ratings that the user is predicted to give an item. A higher rating means a a better rating. If you train a recommendation model using implicit feedback, you generate rankings. A lower ranking means a better ranking. The steps below are the same for both types of recommendation models. See Recommendation Task Types to learn more.

As of MySQL 9.5.0, you have the option to include item and user metadata when generating predictions. These steps include that metadata in the command to generate predictions.

1. If not already done, load the model. You can use the session variable for the model that is valid for the duration of the connection. Alternatively, you can use the model handle previously set. For the option to set the user name, you can set it to `NULL`.

   The following example uses the session variable.

   ```
   mysql> CALL sys.ML_MODEL_LOAD(@model, NULL);
   ```

   The following example uses the model handle.

   ```
   mysql> CALL sys.ML_MODEL_LOAD('recommendation_use_case', NULL);
   ```

2. Make predictions for the test dataset by using the `ML_PREDICT_TABLE` routine.

   ```
   mysql> CALL sys.ML_PREDICT_TABLE(table_name, model_handle, output_table_name), [options]);
   ```

   Replace *`table_name`*, *`model_handle`*, and *`output_table_name`* with your own values. Add *`options`* as needed.

   As of MySQL 9.4.1, you have the option to specify the input table and output table as the same table if specific conditions are met. See Input Tables and Output Tables to learn more.

   The following example runs `ML_PREDICT_TABLE` on the testing dataset previously created.

   ```
   mysql> CALL sys.ML_PREDICT_TABLE('recommendation_data.testing_dataset', @model, 'recommendation_data.re
                                    JSON_OBJECT ('user_metadata', JSON_OBJECT('table_name', 'recommendati
                                                 'item_metadata', JSON_OBJECT('table_name', 'recommendati
   ```

   Where:

   - `recommendation_data.testing_dataset` is the fully qualified name of the input table that contains the data to generate predictions for (*`database_name.table_name`*).
   - `@model` is the session variable for the model handle.
   - `recommendation_data.recommendations` is the fully qualified name of the output table with predictions (*`database_name.table_name`*).
   - `'user_metadata', JSON_OBJECT('table_name', 'recommendation_data.users')` specifies the table that has user metadata to use when generating predictions.
   - `'item_metadata', JSON_OBJECT('table_name', 'recommendation_data.items')` specifies the table that has item metadata to use when generating predictions.

3. Query the output table to review the predicted ratings that users give for each user-item pair.

   ```
   mysql> SELECT * from recommendations;
   +---------+---------+--------+-----------------------------------+
   | user_id | item_id | rating | ml_results                        |
   +---------+---------+--------+-----------------------------------+
   | 1       | 2       |    4.0 | {"predictions": {"rating": 2.71}} |
   | 1       | 4       |    7.0 | {"predictions": {"rating": 3.43}} |
   | 1       | 6       |    1.5 | {"predictions": {"rating": 1.6}}  |
   | 1       | 8       |    3.5 | {"predictions": {"rating": 2.71}} |
   ```

```
| 10       | 18       |    1.5 | {"predictions": {"rating": 3.63}} |
| 10       | 2        |    6.5 | {"predictions": {"rating": 2.82}} |
| 10       | 5        |    3.0 | {"predictions": {"rating": 3.09}} |
| 10       | 6        |    5.5 | {"predictions": {"rating": 1.67}} |
| 2        | 1        |    5.0 | {"predictions": {"rating": 2.88}} |
| 2        | 3        |    8.0 | {"predictions": {"rating": 4.65}} |
| 2        | 5        |    2.5 | {"predictions": {"rating": 3.09}} |
| 2        | 7        |    6.5 | {"predictions": {"rating": 2.23}} |
| 3        | 18       |    7.0 | {"predictions": {"rating": 3.25}} |
| 3        | 2        |    3.5 | {"predictions": {"rating": 2.53}} |
| 3        | 5        |    6.5 | {"predictions": {"rating": 2.77}} |
| 3        | 8        |    2.5 | {"predictions": {"rating": 2.53}} |
| 4        | 1        |    5.5 | {"predictions": {"rating": 3.36}} |
| 4        | 3        |    8.5 | {"predictions": {"rating": 5.42}} |
| 4        | 6        |    2.0 | {"predictions": {"rating": 1.94}} |
| 4        | 7        |    5.5 | {"predictions": {"rating": 2.61}} |
| 5        | 12       |    5.0 | {"predictions": {"rating": 3.29}} |
| 5        | 2        |    7.0 | {"predictions": {"rating": 2.9}}  |
| 5        | 4        |    1.5 | {"predictions": {"rating": 3.68}} |
| 5        | 6        |    4.0 | {"predictions": {"rating": 1.72}} |
| 6        | 3        |    6.0 | {"predictions": {"rating": 4.98}} |
| 6        | 5        |    1.5 | {"predictions": {"rating": 3.31}} |
| 6        | 7        |    4.5 | {"predictions": {"rating": 2.4}}  |
| 6        | 8        |    7.0 | {"predictions": {"rating": 3.03}} |
| 7        | 1        |    6.5 | {"predictions": {"rating": 3.18}} |
| 7        | 4        |    3.0 | {"predictions": {"rating": 3.95}} |
| 7        | 5        |    5.5 | {"predictions": {"rating": 3.41}} |
| 7        | 9        |    8.0 | {"predictions": {"rating": 3.17}} |
| 8        | 2        |    8.5 | {"predictions": {"rating": 2.6}}  |
| 8        | 4        |    2.5 | {"predictions": {"rating": 3.3}}  |
| 8        | 6        |    5.0 | {"predictions": {"rating": 1.54}} |
| 8        | 9        |    3.5 | {"predictions": {"rating": 2.65}} |
| 9        | 1        |    5.0 | {"predictions": {"rating": 2.99}} |
| 9        | 3        |    8.0 | {"predictions": {"rating": 4.83}} |
| 9        | 7        |    2.5 | {"predictions": {"rating": 2.32}} |
| 9        | 8        |    5.5 | {"predictions": {"rating": 2.93}} |
+----------+----------+--------+-----------------------------------+
40 rows in set (0.0459 sec)
```

Review each `user_id` and `item_id` pair and the respective `rating` value in the `ml_results` column. For example, in the first row, user 1 is expected to give item 2 a rating of 2.71.

The values in the `rating` column refer to the past rating the `user_id` gave to the `item_id`. They are not relevant to the values in `ml_results`.

What's Next

- Learn how to generate different types of recommendations:
  - Generate Item Recommendations for Users
  - Generate User Recommendations for Items
  - Generate Recommendations for Similar Items
  - Generate Recommendations for Similar Users
- Learn how to Score a Recommendation Model.

6.7.5.6 Generate Item Recommendations for Users

This topic describes how to generate recommended items for users.

- For known users and known items, the output includes a list of items that the user will most likely give a high rating and the predicted rating or ranking.
- For a new user, and an explicit feedback model, the prediction is the global top K items that received the average highest ratings.

- For a new user, and an implicit feedback model, the prediction is the global top K items with the highest number of interactions.
- For a user who has tried all known items, the prediction is an empty list because it is not possible to recommend any other items. Set `remove_seen` to `false` to repeat existing interactions from the training table.

Before You Begin

Review and complete the following tasks:

- Prepare Data for a Recommendation Model
- Train a Recommendation Model
- Generate Predictions for a Recommendation Model

Recommend Items to Users

When you run `ML_PREDICT_TABLE` to generate item recommendations, a default value of three items are recommended. To change this value, set the `topk` parameter.

As of MySQL 9.5.0, you have the option to include item and user metadata when generating predictions. These steps include that metadata in the command to generate predictions.

1. If not already done, load the model. You can use the session variable for the model that is valid for the duration of the connection. Alternatively, you can use the model handle previously set. For the option to set the user name, you can set it to `NULL`.

   The following example uses the session variable.

   ```
   mysql> CALL sys.ML_MODEL_LOAD(@model, NULL);
   ```

   The following example uses the model handle.

   ```
   mysql> CALL sys.ML_MODEL_LOAD('recommendation_use_case', NULL);
   ```

2. Make predictions for the test dataset by using the `ML_PREDICT_TABLE` routine.

   ```
   mysql> CALL sys.ML_PREDICT_TABLE(table_name, model_handle, output_table_name), [options]);
   ```

   Replace *`table_name`*, *`model_handle`*, and *`output_table_name`* with your own values. Add *`options`* as needed.

   As of MySQL 9.4.1, you have the option to specify the input table and output table as the same table if specific conditions are met. See Input Tables and Output Tables to learn more.

   The following example runs `ML_PREDICT_TABLE` on the testing dataset previously created and sets the `topk` parameter to 2, so only two items are recommended.

   ```
   mysql> CALL sys.ML_PREDICT_TABLE('recommendation_data.testing_dataset', @model, 'recommendation_data.it
                                    JSON_OBJECT('recommend', 'items',
                                                'topk', 2,
                                                'user_metadata', JSON_OBJECT('table_name', 'recommendatio
                                                'item_metadata', JSON_OBJECT('table_name', 'recommendatio
   ```

   Where:

   - `recommendation_data.testing_dataset` is the fully qualified name of the input table that contains the data to generate predictions for (*`database_name.table_name`*).
   - `@model` is the session variable for the model handle.
   - `recommendation_data.item_recommendations` is the fully qualified name of the output table with recommendations (*`database_name.table_name`*).

- `JSON_OBJECT('recommend', 'items', 'topk', 2)` sets the recommendation task to recommend items to users. A maximum of two items to recommend is set.
- `'user_metadata', JSON_OBJECT('table_name', 'recommendation_data.users')` specifies the table that has user metadata to use when generating predictions.
- `'item_metadata', JSON_OBJECT('table_name', 'recommendation_data.items')` specifies the table that has item metadata to use when generating predictions.

3. Query the output table to review the recommended top two items for each user in the output table.

```
mysql> SELECT * from item_recommendations;
+---------+---------+--------+-------------------------------------------------------------------------+
| user_id | item_id | rating | ml_results                                                              |
+---------+---------+--------+-------------------------------------------------------------------------+
| 1       | 2       |    4.0 | {"predictions": {"item_id": ["20", "18"], "rating": [4.7, 3.48]}}       |
| 1       | 4       |    7.0 | {"predictions": {"item_id": ["20", "18"], "rating": [4.7, 3.48]}}       |
| 1       | 6       |    1.5 | {"predictions": {"item_id": ["20", "18"], "rating": [4.7, 3.48]}}       |
| 1       | 8       |    3.5 | {"predictions": {"item_id": ["20", "18"], "rating": [4.7, 3.48]}}       |
| 10      | 18      |    1.5 | {"predictions": {"item_id": ["20", "3"], "rating": [4.9, 4.65]}}        |
| 10      | 2       |    6.5 | {"predictions": {"item_id": ["20", "3"], "rating": [4.9, 4.65]}}        |
| 10      | 5       |    3.0 | {"predictions": {"item_id": ["20", "3"], "rating": [4.9, 4.65]}}        |
| 10      | 6       |    5.5 | {"predictions": {"item_id": ["20", "3"], "rating": [4.9, 4.65]}}        |
| 2       | 1       |    5.0 | {"predictions": {"item_id": ["3", "17"], "rating": [4.65, 3.38]}}       |
| 2       | 3       |    8.0 | {"predictions": {"item_id": ["3", "17"], "rating": [4.65, 3.38]}}       |
| 2       | 5       |    2.5 | {"predictions": {"item_id": ["3", "17"], "rating": [4.65, 3.38]}}       |
| 2       | 7       |    6.5 | {"predictions": {"item_id": ["3", "17"], "rating": [4.65, 3.38]}}       |
| 3       | 18      |    7.0 | {"predictions": {"item_id": ["20", "3"], "rating": [4.39, 4.17]}}       |
| 3       | 2       |    3.5 | {"predictions": {"item_id": ["20", "3"], "rating": [4.39, 4.17]}}       |
| 3       | 5       |    6.5 | {"predictions": {"item_id": ["20", "3"], "rating": [4.39, 4.17]}}       |
| 3       | 8       |    2.5 | {"predictions": {"item_id": ["20", "3"], "rating": [4.39, 4.17]}}       |
| 4       | 1       |    5.5 | {"predictions": {"item_id": ["20", "3"], "rating": [5.71, 5.42]}}       |
| 4       | 3       |    8.5 | {"predictions": {"item_id": ["20", "3"], "rating": [5.71, 5.42]}}       |
| 4       | 6       |    2.0 | {"predictions": {"item_id": ["20", "3"], "rating": [5.71, 5.42]}}       |
| 4       | 7       |    5.5 | {"predictions": {"item_id": ["20", "3"], "rating": [5.71, 5.42]}}       |
| 5       | 12      |    5.0 | {"predictions": {"item_id": ["20", "18"], "rating": [5.05, 3.74]}}      |
| 5       | 2       |    7.0 | {"predictions": {"item_id": ["20", "18"], "rating": [5.05, 3.74]}}      |
| 5       | 4       |    1.5 | {"predictions": {"item_id": ["20", "18"], "rating": [5.05, 3.74]}}      |
| 5       | 6       |    4.0 | {"predictions": {"item_id": ["20", "18"], "rating": [5.05, 3.74]}}      |
| 6       | 3       |    6.0 | {"predictions": {"item_id": ["20", "3"], "rating": [5.25, 4.98]}}       |
| 6       | 5       |    1.5 | {"predictions": {"item_id": ["20", "3"], "rating": [5.25, 4.98]}}       |
| 6       | 7       |    4.5 | {"predictions": {"item_id": ["20", "3"], "rating": [5.25, 4.98]}}       |
| 6       | 8       |    7.0 | {"predictions": {"item_id": ["20", "3"], "rating": [5.25, 4.98]}}       |
| 7       | 1       |    6.5 | {"predictions": {"item_id": ["20", "3"], "rating": [5.41, 5.13]}}       |
| 7       | 4       |    3.0 | {"predictions": {"item_id": ["20", "3"], "rating": [5.41, 5.13]}}       |
| 7       | 5       |    5.5 | {"predictions": {"item_id": ["20", "3"], "rating": [5.41, 5.13]}}       |
| 7       | 9       |    8.0 | {"predictions": {"item_id": ["20", "3"], "rating": [5.41, 5.13]}}       |
| 8       | 2       |    8.5 | {"predictions": {"item_id": ["20", "18"], "rating": [4.53, 3.35]}}      |
| 8       | 4       |    2.5 | {"predictions": {"item_id": ["20", "18"], "rating": [4.53, 3.35]}}      |
| 8       | 6       |    5.0 | {"predictions": {"item_id": ["20", "18"], "rating": [4.53, 3.35]}}      |
| 8       | 9       |    3.5 | {"predictions": {"item_id": ["20", "18"], "rating": [4.53, 3.35]}}      |
| 9       | 1       |    5.0 | {"predictions": {"item_id": ["20", "3"], "rating": [5.09, 4.83]}}       |
| 9       | 3       |    8.0 | {"predictions": {"item_id": ["20", "3"], "rating": [5.09, 4.83]}}       |
| 9       | 7       |    2.5 | {"predictions": {"item_id": ["20", "3"], "rating": [5.09, 4.83]}}       |
| 9       | 8       |    5.5 | {"predictions": {"item_id": ["20", "3"], "rating": [5.09, 4.83]}}       |
+---------+---------+--------+-------------------------------------------------------------------------+
40 rows in set (0.0387 sec)
```

Review the recommended items in the `ml_results` column next to `item_id`. For example, user 1 is predicted to like items 20 and 18. Review the ratings in the `ml_results` column to review the expected ratings for each recommended item. For example, user 1 is expected to rate item 20 with a value of 4.7, and item 18 with a value of 3.48.

What's Next

- Learn how to generate different types of recommendations:
  - Generate Predictions for Ratings and Rankings

- Generate User Recommendations for Items
- Generate Recommendations for Similar Items
- Generate Recommendations for Similar Users

- Learn how to Score a Recommendation Model.

6.7.5.7 Generate User Recommendations for Items

This topic describes how to generate recommended users for items.

- For known users and known items, the output includes a list of users that will most likely give a high rating to an item and will also predict the ratings or rankings.
- For a new item, and an explicit feedback model, the prediction is the global top K users who have provided the average highest ratings.
- For a new item, and an implicit feedback model, the prediction is the global top K users with the highest number of interactions.
- For an item that has been tried by all known users, the prediction is an empty list because it is not possible to recommend any other users. Set `remove_seen` to `false` to repeat existing interactions from the training table.

Before You Begin

Review and complete the following tasks:

- Prepare Data for a Recommendation Model
- Train a Recommendation Model
- Generate Predictions for a Recommendation Model

Recommend Users to Items

When you run `ML_PREDICT_TABLE` to generate user recommendations, a default value of three users are recommended. To change this value, set the `topk` parameter.

As of MySQL 9.5.0, you have the option to include item and user metadata when generating predictions. These steps include that metadata in the command to generate predictions.

1. If not already done, load the model. You can use the session variable for the model that is valid for the duration of the connection. Alternatively, you can use the model handle previously set. For the option to set the user name, you can set it to `NULL`.

   The following example uses the session variable.

   ```
   mysql> CALL sys.ML_MODEL_LOAD(@model, NULL);
   ```

   The following example uses the model handle.

   ```
   mysql> CALL sys.ML_MODEL_LOAD('recommendation_use_case', NULL);
   ```

2. Make predictions for the test dataset by using the `ML_PREDICT_TABLE` routine.

   ```
   mysql> CALL sys.ML_PREDICT_TABLE(table_name, model_handle, output_table_name), [options]);
   ```

   Replace *`table_name`*, *`model_handle`*, and *`output_table_name`* with your own values. Add *`options`* as needed.

   As of MySQL 9.4.1, you have the option to specify the input table and output table as the same table if specific conditions are met. See Input Tables and Output Tables to learn more.

The following example runs `ML_PREDICT_TABLE` on the testing dataset previously created and sets the `topk` parameter to 2, so only two users are recommended.

```
mysql> CALL sys.ML_PREDICT_TABLE('recommendation_data.testing_dataset', @model, 'recommendation_data
                                 JSON_OBJECT('recommend', 'users',
                                             'topk', 2,
                                             'user_metadata', JSON_OBJECT('table_name', 'recommenda
                                             'item_metadata', JSON_OBJECT('table_name', 'recommenda
```

Where:

- `recommendation_data.testing_dataset` is the fully qualified name of the input table that contains the data to generate predictions for (*`database_name.table_name`*).
- `@model` is the session variable for the model handle.
- `recommendation_data.user_recommendations` is the fully qualified name of the output table with recommendations (*`database_name.table_name`*).
- `JSON_OBJECT('recommend', 'users', 'topk', 2)` sets the recommendation task to recommend users to items. A maximum of two users to recommend is set.
- `'user_metadata', JSON_OBJECT('table_name', 'recommendation_data.users')` specifies the table that has user metadata to use when generating predictions.
- `'item_metadata', JSON_OBJECT('table_name', 'recommendation_data.items')` specifies the table that has item metadata to use when generating predictions.

3. Query the output table to review the recommended top two users for each item in the output table.

```
mysql> SELECT * from user_recommendations;
+---------+---------+--------+-----------------------------------------------------------------------+
| user_id | item_id | rating | ml_results                                                            |
+---------+---------+--------+-----------------------------------------------------------------------+
| 1       | 2       |    4.0 | {"predictions": {"user_id": ["6", "5"], "rating": [3.02, 2.9]}}       |
| 1       | 4       |    7.0 | {"predictions": {"user_id": ["4", "7"], "rating": [4.16, 3.95]}}      |
| 1       | 6       |    1.5 | {"predictions": {"user_id": ["4", "7"], "rating": [1.94, 1.84]}}      |
| 1       | 8       |    3.5 | {"predictions": {"user_id": ["7", "6"], "rating": [3.12, 3.03]}}      |
| 10      | 18      |    1.5 | {"predictions": {"user_id": ["5", "10"], "rating": [3.74, 3.63]}}     |
| 10      | 2       |    6.5 | {"predictions": {"user_id": ["6", "5"], "rating": [3.02, 2.9]}}       |
| 10      | 5       |    3.0 | {"predictions": {"user_id": ["6", "5"], "rating": [3.31, 3.19]}}      |
| 10      | 6       |    5.5 | {"predictions": {"user_id": ["4", "7"], "rating": [1.94, 1.84]}}      |
| 2       | 1       |    5.0 | {"predictions": {"user_id": ["4", "7"], "rating": [3.36, 3.18]}}      |
| 2       | 3       |    8.0 | {"predictions": {"user_id": ["4", "7"], "rating": [5.42, 5.13]}}      |
| 2       | 5       |    2.5 | {"predictions": {"user_id": ["6", "5"], "rating": [3.31, 3.19]}}      |
| 2       | 7       |    6.5 | {"predictions": {"user_id": ["4", "6"], "rating": [2.61, 2.4]}}       |
| 3       | 18      |    7.0 | {"predictions": {"user_id": ["5", "10"], "rating": [3.74, 3.63]}}     |
| 3       | 2       |    3.5 | {"predictions": {"user_id": ["6", "5"], "rating": [3.02, 2.9]}}       |
| 3       | 5       |    6.5 | {"predictions": {"user_id": ["6", "5"], "rating": [3.31, 3.19]}}      |
| 3       | 8       |    2.5 | {"predictions": {"user_id": ["7", "6"], "rating": [3.12, 3.03]}}      |
| 4       | 1       |    5.5 | {"predictions": {"user_id": ["4", "7"], "rating": [3.36, 3.18]}}      |
| 4       | 3       |    8.5 | {"predictions": {"user_id": ["4", "7"], "rating": [5.42, 5.13]}}      |
| 4       | 6       |    2.0 | {"predictions": {"user_id": ["4", "7"], "rating": [1.94, 1.84]}}      |
| 4       | 7       |    5.5 | {"predictions": {"user_id": ["4", "6"], "rating": [2.61, 2.4]}}       |
| 5       | 12      |    5.0 | {"predictions": {"user_id": ["5", "10"], "rating": [3.29, 3.2]}}      |
| 5       | 2       |    7.0 | {"predictions": {"user_id": ["6", "5"], "rating": [3.02, 2.9]}}       |
| 5       | 4       |    1.5 | {"predictions": {"user_id": ["4", "7"], "rating": [4.16, 3.95]}}      |
| 5       | 6       |    4.0 | {"predictions": {"user_id": ["4", "7"], "rating": [1.94, 1.84]}}      |
| 6       | 3       |    6.0 | {"predictions": {"user_id": ["4", "7"], "rating": [5.42, 5.13]}}      |
| 6       | 5       |    1.5 | {"predictions": {"user_id": ["6", "5"], "rating": [3.31, 3.19]}}      |
| 6       | 7       |    4.5 | {"predictions": {"user_id": ["4", "6"], "rating": [2.61, 2.4]}}       |
| 6       | 8       |    7.0 | {"predictions": {"user_id": ["7", "6"], "rating": [3.12, 3.03]}}      |
| 7       | 1       |    6.5 | {"predictions": {"user_id": ["4", "7"], "rating": [3.36, 3.18]}}      |
| 7       | 4       |    3.0 | {"predictions": {"user_id": ["4", "7"], "rating": [4.16, 3.95]}}      |
| 7       | 5       |    5.5 | {"predictions": {"user_id": ["6", "5"], "rating": [3.31, 3.19]}}      |
| 7       | 9       |    8.0 | {"predictions": {"user_id": ["4", "7"], "rating": [3.34, 3.17]}}      |
| 8       | 2       |    8.5 | {"predictions": {"user_id": ["6", "5"], "rating": [3.02, 2.9]}}       |
```

```
| 8       | 4       |    2.5 | {"predictions": {"user_id": ["4", "7"], "rating": [4.16, 3.95]}}  |
| 8       | 6       |    5.0 | {"predictions": {"user_id": ["4", "7"], "rating": [1.94, 1.84]}}  |
| 8       | 9       |    3.5 | {"predictions": {"user_id": ["4", "7"], "rating": [3.34, 3.17]}}  |
| 9       | 1       |    5.0 | {"predictions": {"user_id": ["4", "7"], "rating": [3.36, 3.18]}}  |
| 9       | 3       |    8.0 | {"predictions": {"user_id": ["4", "7"], "rating": [5.42, 5.13]}}  |
| 9       | 7       |    2.5 | {"predictions": {"user_id": ["4", "6"], "rating": [2.61, 2.4]}}   |
| 9       | 8       |    5.5 | {"predictions": {"user_id": ["7", "6"], "rating": [3.12, 3.03]}}  |
+---------+---------+--------+-------------------------------------------------------------------+
40 rows in set (0.0476 sec)
```

Review the recommended users in the `ml_results` column next to `user_id`. For example, for item 2, users 6 and 5 are the top users predicted to like it. Review the ratings in the `ml_results` column to review the expected ratings for each recommended item. For example, user 6 is expected to rate item 2 with a value of 3.02, and user 5 with a value of 2.9.

What's Next

- Learn how to generate different types of recommendations:
  - Generate Predictions for Ratings and Rankings
  - Generate Item Recommendations for Users
  - Generate Recommendations for Similar Items
  - Generate Recommendations for Similar Users
- Learn how to Score a Recommendation Model.

6.7.5.8 Generate Recommendations for Similar Items

This topic describes how to generate recommendations for similar items.

- For known items, the output includes a list of predicted items that have similar ratings and are appreciated by similar users.
- The predictions are expressed in cosine similarity, and range from 0, very dissimilar, to 1, very similar.
- For a new item, there is no information to provide a prediction. This generates an error.

Before You Begin

Review and complete the following tasks:

- Prepare Data for a Recommendation Model
- Train a Recommendation Model
- Generate Predictions for a Recommendation Model

Generate Similar Items

When you run `ML_PREDICT_TABLE` to generate similar item recommendations, a default value of three similar items are recommended. To change this value, set the `topk` parameter.

As of MySQL 9.5.0, you have the option to include item and user metadata when generating predictions. These steps include that metadata in the command to generate predictions.

1. If not already done, load the model. You can use the session variable for the model that is valid for the duration of the connection. Alternatively, you can use the model handle previously set. For the option to set the user name, you can set it to `NULL`.

   The following example uses the session variable.

```
mysql> CALL sys.ML_MODEL_LOAD(@model, NULL);
```

The following example uses the model handle.

```
mysql> CALL sys.ML_MODEL_LOAD('recommendation_use_case', NULL);
```

2. Make predictions for the test dataset by using the `ML_PREDICT_TABLE` routine.

```
mysql> CALL sys.ML_PREDICT_TABLE(table_name, model_handle, output_table_name), [options]);
```

Replace *`table_name`*, *`model_handle`*, and *`output_table_name`* with your own values. Add *`options`* as needed.

As of MySQL 9.4.1, you have the option to specify the input table and output table as the same table if specific conditions are met. See Input Tables and Output Tables to learn more.

The following example runs `ML_PREDICT_TABLE` on the testing dataset previously created and sets the `topk` parameter to 2, so only two similar items are generated.

```
mysql> CALL sys.ML_PREDICT_TABLE('recommendation_data.testing_dataset', @model, 'recommendation_data
                                 JSON_OBJECT('recommend', 'items_to_items',
                                             'topk', 2,
                                             'user_metadata', JSON_OBJECT('table_name', 'recommenda
                                             'item_metadata', JSON_OBJECT('table_name', 'recommenda
```

Where:

- `recommendation_data.testing_dataset` is the fully qualified name of the input table that contains the data to generate predictions for (*`database_name.table_name`*).
- `@model` is the session variable for the model handle.
- `recommendation_data.similar_item_recommendations` is the fully qualified name of the output table with recommendations (*`database_name.table_name`*).
- `JSON_OBJECT('recommend', 'items_to_items', 'topk', 2)` sets the recommendation task to recommend similar items. A maximum of two similar items is set.
- `'user_metadata', JSON_OBJECT('table_name', 'recommendation_data.users')` specifies the table that has user metadata to use when generating predictions.
- `'item_metadata', JSON_OBJECT('table_name', 'recommendation_data.items')` specifies the table that has item metadata to use when generating predictions.

3. Query the output table to review the top two similar items for each item in the output table.

```
mysql> SELECT * from similar_item_recommendations;
+---------+---------+--------+-----------------------------------------------------------------------
| user_id | item_id | rating | ml_results
+---------+---------+--------+-----------------------------------------------------------------------
| 1       | 2       |    4.0 | {"predictions": {"item_id": ["14", "10"], "similarity": [0.9831, 0.96
| 1       | 4       |    7.0 | {"predictions": {"item_id": ["9", "6"], "similarity": [0.6838, 0.6444
| 1       | 6       |    1.5 | {"predictions": {"item_id": ["8", "17"], "similarity": [0.8991, 0.841
| 1       | 8       |    3.5 | {"predictions": {"item_id": ["6", "17"], "similarity": [0.8991, 0.794
| 10      | 18      |    1.5 | {"predictions": {"item_id": ["16", "12"], "similarity": [0.9869, 0.94
| 10      | 2       |    6.5 | {"predictions": {"item_id": ["14", "10"], "similarity": [0.9831, 0.96
| 10      | 5       |    3.0 | {"predictions": {"item_id": ["16", "2"], "similarity": [0.9036, 0.858
| 10      | 6       |    5.5 | {"predictions": {"item_id": ["8", "17"], "similarity": [0.8991, 0.841
| 2       | 1       |    5.0 | {"predictions": {"item_id": ["15", "17"], "similarity": [0.8462, 0.79
| 2       | 3       |    8.0 | {"predictions": {"item_id": ["19", "13"], "similarity": [0.9826, 0.88
| 2       | 5       |    2.5 | {"predictions": {"item_id": ["16", "2"], "similarity": [0.9036, 0.858
| 2       | 7       |    6.5 | {"predictions": {"item_id": ["11", "15"], "similarity": [0.6959, 0.67
| 3       | 18      |    7.0 | {"predictions": {"item_id": ["16", "12"], "similarity": [0.9869, 0.94
| 3       | 2       |    3.5 | {"predictions": {"item_id": ["14", "10"], "similarity": [0.9831, 0.96
| 3       | 5       |    6.5 | {"predictions": {"item_id": ["16", "2"], "similarity": [0.9036, 0.858
| 3       | 8       |    2.5 | {"predictions": {"item_id": ["6", "17"], "similarity": [0.8991, 0.794
```

```
| 4       | 1       |    5.5 | {"predictions": {"item_id": ["15", "17"], "similarity": [0.8462, 0.7966]}}
| 4       | 3       |    8.5 | {"predictions": {"item_id": ["19", "13"], "similarity": [0.9826, 0.8851]}}
| 4       | 6       |    2.0 | {"predictions": {"item_id": ["8", "17"], "similarity": [0.8991, 0.8412]}}
| 4       | 7       |    5.5 | {"predictions": {"item_id": ["11", "15"], "similarity": [0.6959, 0.6724]}}
| 5       | 12      |    5.0 | {"predictions": {"item_id": ["18", "16"], "similarity": [0.9464, 0.9454]}}
| 5       | 2       |    7.0 | {"predictions": {"item_id": ["14", "10"], "similarity": [0.9831, 0.965]}}
| 5       | 4       |    1.5 | {"predictions": {"item_id": ["9", "6"], "similarity": [0.6838, 0.6444]}}
| 5       | 6       |    4.0 | {"predictions": {"item_id": ["8", "17"], "similarity": [0.8991, 0.8412]}}
| 6       | 3       |    6.0 | {"predictions": {"item_id": ["19", "13"], "similarity": [0.9826, 0.8851]}}
| 6       | 5       |    1.5 | {"predictions": {"item_id": ["16", "2"], "similarity": [0.9036, 0.8586]}}
| 6       | 7       |    4.5 | {"predictions": {"item_id": ["11", "15"], "similarity": [0.6959, 0.6724]}}
| 6       | 8       |    7.0 | {"predictions": {"item_id": ["6", "17"], "similarity": [0.8991, 0.7942]}}
| 7       | 1       |    6.5 | {"predictions": {"item_id": ["15", "17"], "similarity": [0.8462, 0.7966]}}
| 7       | 4       |    3.0 | {"predictions": {"item_id": ["9", "6"], "similarity": [0.6838, 0.6444]}}
| 7       | 5       |    5.5 | {"predictions": {"item_id": ["16", "2"], "similarity": [0.9036, 0.8586]}}
| 7       | 9       |    8.0 | {"predictions": {"item_id": ["1", "4"], "similarity": [0.7721, 0.6838]}}
| 8       | 2       |    8.5 | {"predictions": {"item_id": ["14", "10"], "similarity": [0.9831, 0.965]}}
| 8       | 4       |    2.5 | {"predictions": {"item_id": ["9", "6"], "similarity": [0.6838, 0.6444]}}
| 8       | 6       |    5.0 | {"predictions": {"item_id": ["8", "17"], "similarity": [0.8991, 0.8412]}}
| 8       | 9       |    3.5 | {"predictions": {"item_id": ["1", "4"], "similarity": [0.7721, 0.6838]}}
| 9       | 1       |    5.0 | {"predictions": {"item_id": ["15", "17"], "similarity": [0.8462, 0.7966]}}
| 9       | 3       |    8.0 | {"predictions": {"item_id": ["19", "13"], "similarity": [0.9826, 0.8851]}}
| 9       | 7       |    2.5 | {"predictions": {"item_id": ["11", "15"], "similarity": [0.6959, 0.6724]}}
| 9       | 8       |    5.5 | {"predictions": {"item_id": ["6", "17"], "similarity": [0.8991, 0.7942]}}
+---------+---------+--------+-------------------------------------------------------------------------
40 rows in set (0.0401 sec)
```

Review the recommended similar items in the `ml_results` column next to `item_id`. For example, for item 2, items 14 and 10 are the top items predicted to be most similar. Review the similarity values in the `ml_results` column next to `similarity` to review the how similar each item is. For example, item 14 has a similarity value of 0.9831 to item 2, and item 10 has a similarity value of 0.965.

What's Next

- Learn how to generate different types of recommendations:
  - Generate Predictions for Ratings and Rankings
  - Generate Item Recommendations for Users
  - Generate User Recommendations for Items
  - Generate Recommendations for Similar Users
- Learn how to Score a Recommendation Model.

6.7.5.9 Generate Recommendations for Similar Users

This topic describes how to generate recommendations for similar users.

- For known users, the output includes a list of predicted users that have similar behavior and taste.
- The predictions are expressed in cosine similarity, and range from 0, very dissimilar, to 1, very similar.
- For a new user, there is no information to provide a prediction. This generates an error.

Before You Begin

Review and complete the following tasks:

- Prepare Data for a Recommendation Model
- Train a Recommendation Model
- Generate Predictions for a Recommendation Model

Generate Similar Users

When you run `ML_PREDICT_TABLE` to generate similar user recommendations, a default value of three similar users are recommended. To change this value, set the `topk` parameter.

As of MySQL 9.5.0, you have the option to include item and user metadata when generating predictions. These steps include that metadata in the command to generate predictions.

1. If not already done, load the model. You can use the session variable for the model that is valid for the duration of the connection. Alternatively, you can use the model handle previously set. For the option to set the user name, you can set it to `NULL`.

   The following example uses the session variable.

   ```
   mysql> CALL sys.ML_MODEL_LOAD(@model, NULL);
   ```

   The following example uses the model handle.

   ```
   mysql> CALL sys.ML_MODEL_LOAD('recommendation_use_case', NULL);
   ```

2. Make predictions for the test dataset by using the `ML_PREDICT_TABLE` routine.

   ```
   mysql> CALL sys.ML_PREDICT_TABLE(table_name, model_handle, output_table_name), [options]);
   ```

   Replace *`table_name`*, *`model_handle`*, and *`output_table_name`* with your own values. Add *`options`* as needed.

   As of MySQL 9.4.1, you have the option to specify the input table and output table as the same table if specific conditions are met. See Input Tables and Output Tables to learn more.

   The following example runs `ML_PREDICT_TABLE` on the testing dataset previously created and sets the `topk` parameter to 2, so only two similar users are generated.

   ```
   mysql> CALL sys.ML_PREDICT_TABLE('recommendation_data.testing_dataset', @model, 'recommendation_data
                                    JSON_OBJECT('recommend', 'users_to_users',
                                                'topk', 2,
                                                'user_metadata', JSON_OBJECT('table_name', 'recommenda
                                                'item_metadata', JSON_OBJECT('table_name', 'recommenda
   ```

   Where:

   - `recommendation_data.testing_dataset` is the fully qualified name of the input table that contains the data to generate predictions for (*`database_name.table_name`*).
   - `@model` is the session variable for the model handle.
   - `recommendation_data.similar_user_recommendations` is the fully qualified name of the output table with recommendations (*`database_name.table_name`*).
   - `JSON_OBJECT('recommend', 'users_to_users', 'topk', 2)` sets the recommendation task to recommend similar users. A maximum of two similar users is set.
   - `'user_metadata', JSON_OBJECT('table_name', 'recommendation_data.users')` specifies the table that has user metadata to use when generating predictions.
   - `'item_metadata', JSON_OBJECT('table_name', 'recommendation_data.items')` specifies the table that has item metadata to use when generating predictions.

3. Query the output table to review the top two similar users generated for each user in the output table.

   ```
   mysql> SELECT * from similar_user_recommendations;
   +---------+---------+--------+-------------------------------------------------------------------------
   | user_id | item_id | rating | ml_results
   +---------+---------+--------+-------------------------------------------------------------------------
   ```

```
| 1       | 2       |    4.0 | {"predictions": {"user_id": ["3", "5"], "similarity": [0.7922, 0.7238]}}
| 1       | 4       |    7.0 | {"predictions": {"user_id": ["3", "5"], "similarity": [0.7922, 0.7238]}}
| 1       | 6       |    1.5 | {"predictions": {"user_id": ["3", "5"], "similarity": [0.7922, 0.7238]}}
| 1       | 8       |    3.5 | {"predictions": {"user_id": ["3", "5"], "similarity": [0.7922, 0.7238]}}
| 10      | 18      |    1.5 | {"predictions": {"user_id": ["3", "5"], "similarity": [0.6827, 0.5943]}}
| 10      | 2       |    6.5 | {"predictions": {"user_id": ["3", "5"], "similarity": [0.6827, 0.5943]}}
| 10      | 5       |    3.0 | {"predictions": {"user_id": ["3", "5"], "similarity": [0.6827, 0.5943]}}
| 10      | 6       |    5.5 | {"predictions": {"user_id": ["3", "5"], "similarity": [0.6827, 0.5943]}}
| 2       | 1       |    5.0 | {"predictions": {"user_id": ["7", "9"], "similarity": [0.6473, 0.5746]}}
| 2       | 3       |    8.0 | {"predictions": {"user_id": ["7", "9"], "similarity": [0.6473, 0.5746]}}
| 2       | 5       |    2.5 | {"predictions": {"user_id": ["7", "9"], "similarity": [0.6473, 0.5746]}}
| 2       | 7       |    6.5 | {"predictions": {"user_id": ["7", "9"], "similarity": [0.6473, 0.5746]}}
| 3       | 18      |    7.0 | {"predictions": {"user_id": ["1", "10"], "similarity": [0.7922, 0.6827]}}
| 3       | 2       |    3.5 | {"predictions": {"user_id": ["1", "10"], "similarity": [0.7922, 0.6827]}}
| 3       | 5       |    6.5 | {"predictions": {"user_id": ["1", "10"], "similarity": [0.7922, 0.6827]}}
| 3       | 8       |    2.5 | {"predictions": {"user_id": ["1", "10"], "similarity": [0.7922, 0.6827]}}
| 4       | 1       |    5.5 | {"predictions": {"user_id": ["9", "7"], "similarity": [0.9764, 0.9087]}}
| 4       | 3       |    8.5 | {"predictions": {"user_id": ["9", "7"], "similarity": [0.9764, 0.9087]}}
| 4       | 6       |    2.0 | {"predictions": {"user_id": ["9", "7"], "similarity": [0.9764, 0.9087]}}
| 4       | 7       |    5.5 | {"predictions": {"user_id": ["9", "7"], "similarity": [0.9764, 0.9087]}}
| 5       | 12      |    5.0 | {"predictions": {"user_id": ["8", "1"], "similarity": [0.992, 0.7238]}}
| 5       | 2       |    7.0 | {"predictions": {"user_id": ["8", "1"], "similarity": [0.992, 0.7238]}}
| 5       | 4       |    1.5 | {"predictions": {"user_id": ["8", "1"], "similarity": [0.992, 0.7238]}}
| 5       | 6       |    4.0 | {"predictions": {"user_id": ["8", "1"], "similarity": [0.992, 0.7238]}}
| 6       | 3       |    6.0 | {"predictions": {"user_id": ["4", "9"], "similarity": [0.5695, 0.4862]}}
| 6       | 5       |    1.5 | {"predictions": {"user_id": ["4", "9"], "similarity": [0.5695, 0.4862]}}
| 6       | 7       |    4.5 | {"predictions": {"user_id": ["4", "9"], "similarity": [0.5695, 0.4862]}}
| 6       | 8       |    7.0 | {"predictions": {"user_id": ["4", "9"], "similarity": [0.5695, 0.4862]}}
| 7       | 1       |    6.5 | {"predictions": {"user_id": ["9", "4"], "similarity": [0.9738, 0.9087]}}
| 7       | 4       |    3.0 | {"predictions": {"user_id": ["9", "4"], "similarity": [0.9738, 0.9087]}}
| 7       | 5       |    5.5 | {"predictions": {"user_id": ["9", "4"], "similarity": [0.9738, 0.9087]}}
| 7       | 9       |    8.0 | {"predictions": {"user_id": ["9", "4"], "similarity": [0.9738, 0.9087]}}
| 8       | 2       |    8.5 | {"predictions": {"user_id": ["5", "1"], "similarity": [0.992, 0.6356]}}
| 8       | 4       |    2.5 | {"predictions": {"user_id": ["5", "1"], "similarity": [0.992, 0.6356]}}
| 8       | 6       |    5.0 | {"predictions": {"user_id": ["5", "1"], "similarity": [0.992, 0.6356]}}
| 8       | 9       |    3.5 | {"predictions": {"user_id": ["5", "1"], "similarity": [0.992, 0.6356]}}
| 9       | 1       |    5.0 | {"predictions": {"user_id": ["4", "7"], "similarity": [0.9764, 0.9738]}}
| 9       | 3       |    8.0 | {"predictions": {"user_id": ["4", "7"], "similarity": [0.9764, 0.9738]}}
| 9       | 7       |    2.5 | {"predictions": {"user_id": ["4", "7"], "similarity": [0.9764, 0.9738]}}
| 9       | 8       |    5.5 | {"predictions": {"user_id": ["4", "7"], "similarity": [0.9764, 0.9738]}}
+---------+---------+--------+-------------------------------------------------------------------------
40 rows in set (0.0414 sec)
```

Review the recommended similar users in the `ml_results` column next to `user_id`. For example, for user 1, users 3 and 5 are the top users predicted to be most similar. Review the similarity values in the `ml_results` column next to `similarity` to review the how similar each user is. For example, user 3 has a similarity value of 0.7922 to user 1, and user 5 has a similarity value of 0.7238.

What's Next

- Learn how to generate different types of recommendations:
  - Generate Predictions for Ratings and Rankings
  - Generate Item Recommendations for Users
  - Generate User Recommendations for Items
  - Generate Recommendations for Similar Items
- Learn how to Score a Recommendation Model.

6.7.5.10 Score a Recommendation Model

After generating predicted ratings/rankings and recommendations, you can score the model to assess its reliability. For a list of scoring metrics you can use with recommendation models, see Recommendation Model Metrics. For this use case, you use the test dataset for validation. In a real-

world use case, you should use a separate validation dataset that has the target column and ground truth values for the scoring validation. You should also use a larger number of records for training and validation to get a valid score.

Before You Begin

Review and complete the following tasks:

- Prepare Data for a Recommendation Model
- Train a Recommendation Model
- Generate Predictions for a Recommendation Model
- Generate Predictions for Ratings and Rankings
- Generate Item Recommendations for Users
- Generate User Recommendations for Items
- Generate Recommendations for Similar Items
- Generate Recommendations for Similar Users

Options for Scoring Recommendation Models

The *`options`* for `ML_SCORE` include the following:

- `threshold`: The optional threshold that defines positive feedback, and a relevant sample. Only use with ranking metrics. It can be used for either explicit or implicit feedback.
- `topk`: The optional top number of recommendations to provide. The default is `3`. Set a positive integer between 1 and the number of rows in the table.

  A `recommendation` task and ranking metrics can use both `threshold` and `topk`.
- `remove_seen`: If the input table overlaps with the training table, and `remove_seen` is `true`, then the model will not repeat existing interactions. The default is `true`. Set `remove_seen` to `false` to repeat existing interactions from the training table.
- `item_metadata`: As of MySQL 9.5.0, it defines the table that has item descriptions. It is a JSON object that has the `table_name` option as a key, which specifies the table that has item descriptions. One column must be the same as the `item_id` in the input table.
- `user_metadata`: As of MySQL 9.5.0, it defines the table that has user descriptions. It is a JSON object that has the `table_name` option as a key, which specifies the table that has user descriptions. One column must be the same as the `user_id` in the input table.
  - `table_name`: To be used with the `item_metadata` and `user_metadata` options. It specifies the table name that has item or user descriptions. It must be a string in a fully qualified format (schema_name.table_name) that specifies the table name.

Score Model

1. If not already done, load the model. You can use the session variable for the model that is valid for the duration of the connection. Alternatively, you can use the model handle previously set. For the option to set the user name, you can set it to `NULL`.

   The following example uses the session variable.

   ```
   mysql> CALL sys.ML_MODEL_LOAD(@model, NULL);
   ```

   The following example uses the model handle.

   ```
   mysql> CALL sys.ML_MODEL_LOAD('recommendation_use_case', NULL);
   ```

2. Score the model with the `ML_SCORE` routine and use the `precision_at_k` metric.

```
mysql> CALL sys.ML_SCORE(table_name, target_column_name, model_handle, metric, score, [options]);
```

Replace *`table_name`*, *`target_column_name`*, *`model_handle`*, *`metric`*, *`score`* with your own values.

The following example runs `ML_SCORE` on the testing dataset previously created.

```
mysql> CALL sys.ML_SCORE('recommendation_data.testing_dataset', 'rating', @model, 'precision_at_k', @re
```

Where:

- `recommendation_data.testing_dataset` is the fully qualified name of the validation dataset.
- `rating` is the target column name with ground truth values.
- `@model` is the session variable for the model handle.
- `precision_at_k` is the selected scoring metric.
- `@recommendation_score` is the session variable name for the score value.
- `NULL` means that no other options are defined for the routine.

3. Retrieve the score by querying the @score session variable.

```
mysql> SELECT @recommendation_score;
+-----------------------+
| @recommendation_score |
+-----------------------+
| 0.23333333432674408   |
+-----------------------+
1 row in set (0.0491 sec)
```

4. If done working with the model, unload it with the `ML_MODEL_UNLOAD` routine.

```
mysql> CALL sys.ML_MODEL_UNLOAD('recommendation_use_case');
```

To avoid consuming too much memory, it is good practice to unload a model when you are finished using it.

What's Next

- Review other Machine Learning Use Cases.

6.7.6 Topic Modeling

MySQL 9.0.1-u1 introduces topic modeling, which is an unsupervised machine learning technique that's capable of scanning a set of documents, detecting word and phrase patterns within them, and automatically clustering word groups and similar expressions that best characterize the documents.

The following tasks use a dataset generated by OCI GenAI using Meta Llama Models. The topic modeling use-case is to summarize movie plots.

To generate your own datasets to create machine learning models in MySQL HeatWave, learn how to Generate Text-Based Content.

Note

Datasets were generated using Meta Llama models. Your use of this Llama model is subject to your Oracle agreements and this Llama license agreement: https://downloads.mysql.com/docs/LLAMA_31_8B_INSTRUCT-license.pdf.

6.7.6.1 Prepare Data for Topic Modeling

This topic describes how to prepare the data to use for topic modeling. The model uses a data sample generated by OCI GenAI. To prepare the data for this use case, you set up a dataset to use for both training and testing.

You have the option to automatically Prepare Training and Testing Datasets with your own data by using the `TRAIN_TEST_SPLIT` routine.

Before You Begin

- Learn how to Prepare Data.

Prepare Data

To prepare the data for topic modeling:

1. Connect to your MySQL HeatWave Database System.
2. Create and use the database to store the data.

```
mysql> CREATE DATABASE topic_modeling_data;
mysql> USE topic_modeling_data;
```

3. Create the table to use for both training and testing.

```
mysql> CREATE TABLE movies ( description TEXT );
```

4. Insert the sample data into the table. Copy and paste the following commands.

```
INSERT INTO movies (description) VALUES ('In a post-apocalyptic wasteland, a lone survivor named Max
INSERT INTO movies (description) VALUES('A young man named Neo discovers that the world as he knows
INSERT INTO movies (description) VALUES('A wealthy family, the Corleones, is drawn into the underwor
INSERT INTO movies (description) VALUES('A group of scientists attempt to harness the power of a bla
INSERT INTO movies (description) VALUES('A young woman, Alice, finds herself in a mysterious and far
INSERT INTO movies (description) VALUES('A young man named Luke Skywalker joins forces with a group
INSERT INTO movies (description) VALUES('A team of scientists and explorers travel through a wormhol
INSERT INTO movies (description) VALUES('A young Viking named Hiccup aspires to hunt dragons like hi
INSERT INTO movies (description) VALUES('A young FBI agent, Clarice Starling, is assigned to help fi
INSERT INTO movies (description) VALUES('A young man named Harry Potter discovers that he is a wizar
INSERT INTO movies (description) VALUES('A group of criminals are given a second chance at redemptio
INSERT INTO movies (description) VALUES('A young woman, Elle Woods, is determined to win back her ex
INSERT INTO movies (description) VALUES('A group of friends travel to a remote cabin in the woods fo
INSERT INTO movies (description) VALUES('A young man named Marty McFly accidentally travels back in
INSERT INTO movies (description) VALUES('A young woman, Katniss Everdeen, volunteers to take her you
INSERT INTO movies (description) VALUES('A young man named Frodo Baggins inherits a powerful ring, w
INSERT INTO movies (description) VALUES('A young woman, Jo March, and her sisters come of age in Ame
INSERT INTO movies (description) VALUES('A group of astronauts on a mission to Mars face a critical
INSERT INTO movies (description) VALUES('A young man, Scott, discovers a hidden virtual world calle
INSERT INTO movies (description) VALUES('A group of high school students from different social cliqu
```

What's Next

- Learn how to Train a Model with Topic Modeling.

6.7.6.2 Train a Model with Topic Modeling

After preparing the data for topic modeling, you can train the model.

Before You Begin

- Review and complete all the tasks to Prepare Data for Topic Modeling.

Requirements for Topic Modeling Training

Define the following required parameters for topic modeling.

- Set the `task` parameter to `topic_modeling`.

- `document_column`: Define the column that contains the text that the model uses to generate topics and tags as output. The output is an array of word groups that best characterize the text.

Unsupported Topic Modeling Options

When MySQL HeatWave AutoML runs topic modeling, the operation is based on a single algorithm that does not require the tuning of hyperparameters. Moreover, topic modeling is an unsupervised task, which means there are no labels. Therefore, you cannot use the following options for topic modeling:

- `model_list`
- `optimization_metric`
- `exclude_model_list`
- `exclude_column_list`
- `include_column_list`

Unsupported Routines

You cannot run the following routines for topic modeling:

- `ML_EXPLAIN`
- `ML_EXPLAIN_ROW`
- `ML_EXPLAIN_TABLE`
- `ML_SCORE`

Train Model

Train the model with the `ML_TRAIN` routine and use the `movies` table previously created. Before training the model, it is good practice to define the model handle instead of automatically creating one. See Define Model Handle.

1. Optionally, set the value of the session variable, which sets the model handle to this same value.

   ```
   mysql> SET @variable = 'model_handle';
   ```

   Replace *`@variable`* and *`model_handle`* with your own definitions. For example:

   ```
   mysql> SET @model='topic_modeling_use_case';
   ```

   The model handle is set to `topic_modeling_use_case`.

2. Run the `ML_TRAIN` routine.

   ```
   mysql> CALL sys.ML_TRAIN('table_name', 'target_column_name', JSON_OBJECT('task', 'task_name'), model_ha
   ```

   Replace *`table_name`*, *`target_column_name`*, *`task_name`*, and *`model_handle`* with your own values.

   The following example runs `ML_TRAIN` on the dataset previously created.

   ```
   mysql> CALL sys.ML_TRAIN('topic_modeling_data.movies', NULL,
                            JSON_OBJECT('task', 'topic_modeling', 'document_column', 'description'), @mod
   ```

   Where:

   - `topic_modeling_data.movies` is the fully qualified name of the table that contains the training dataset (*`database_name.table_name`*).
   - `NULL` is set for the target column because topic modeling uses unlabeled data, so you cannot set a target column.

- `JSON_OBJECT('task', 'topic_modeling')` specifies the machine learning task type.
- `@model` is the session variable previously set that defines the model handle to the name defined by the user: `topic_modeling_use_case`. If you do not define the model handle before training the model, the model handle is automatically generated, and the session variable only stores the model handle for the duration of the connection. User variables are written as *`@var_name`*. Any valid name for a user-defined variable is permitted. See Work with Model Handles to learn more.

3. When the training operation finishes, the model handle is assigned to the `@model` session variable, and the model is stored in the model catalog. View the entry in the model catalog with the following query. Replace *`user1`* with your MySQL account name.

```
mysql> SELECT model_id, model_handle, train_table_name FROM ML_SCHEMA_user1.MODEL_CATALOG WHERE mode
+----------+------------------------+---------------------------+
| model_id | model_handle           | train_table_name          |
+----------+------------------------+---------------------------+
|        8 | topic_modeling_use_case | topic_modeling_data.movies |
+----------+------------------------+---------------------------+
37 rows in set (0.0449 sec)
```

What's Next

- Learn how to Generate Predictions for Topic Modeling.

6.7.6.3 Run Topic Modeling on Trained Text

After training the model, you can run topic modeling on the trained text.

To run topic modeling, use the sample data from the `movies` dataset. The dataset has no target column. When the output table generates, you can review the generated word groups and expressions for the trained text.

Before You Begin

Complete the following tasks:

- Prepare Data for Topic Modeling.
- Train a Model with Topic Modeling

Run Topic Modeling for a Table

1. If not already done, load the model. You can use the session variable for the model that is valid for the duration of the connection. Alternatively, you can use the model handle previously set. For the option to set the user name, you can set it to `NULL`.

   The following example uses the session variable.

   ```
   mysql> CALL sys.ML_MODEL_LOAD(@model, NULL);
   ```

   The following example uses the model handle.

   ```
   mysql> CALL sys.ML_MODEL_LOAD('topic_modeling_use_case', NULL);
   ```

2. Run topic modeling on the dataset by using the `ML_PREDICT_TABLE` routine.

   ```
   mysql> CALL sys.ML_PREDICT_TABLE(table_name, model_handle, output_table_name), [options]);
   ```

   Replace *`table_name`*, *`model_handle`*, and *`output_table_name`* with your own values. Add *`options`* as needed.

   As of MySQL 9.4.1, you have the option to specify the input table and output table as the same table if specific conditions are met. See Input Tables and Output Tables to learn more.

The following example runs `ML_PREDICT_TABLE` on the dataset previously created.

```
mysql> CALL sys.ML_PREDICT_TABLE('topic_modeling_data.movies', @model, 'topic_modeling_data.topic_model
```

Where:

- `topic_modeling_data.movies` is the fully qualified name of the input table that contains the data to run topic modeling for (*`database_name.table_name`*).
- `@model` is the session variable for the model handle.
- `topic_modeling_data.topic_modeling_predictions` is the fully qualified name of the output table with generated word groups and expressions (*`database_name.table_name`*).
- `NULL` sets no options for the routine.

3. Query the `ml_results` column from the output table. Review the generated word groups and expressions for the movie descriptions next to `tags`.

```
mysql> SELECT ml_results FROM topic_modeling_predictions;
+-----------------------------------------------------------------------------------------------------
| ml_results
+-----------------------------------------------------------------------------------------------------
| {"predictions": {"tags": ["dangerous", "future", "join force", "journey", "warrior", "battle", "rebel
| {"predictions": {"tags": ["machine", "world", "agent", "group rebel", "human", "real", "real world",
| {"predictions": {"tags": ["family", "empire", "rival", "criminal", "powerful"]}}
| {"predictions": {"tags": ["scientist", "astronaut", "attempt", "humanity", "massive", "work", "earth"
| {"predictions": {"tags": ["include", "mysterious", "strange", "world", "woman", "young woman", "young
| {"predictions": {"tags": ["empire", "force", "dark force", "evil", "group rebel", "join force", "wise
| {"predictions": {"tags": ["alien", "attempt", "humanity", "planet", "battle", "creature", "ensure", "
| {"predictions": {"tags": ["future", "human", "creature", "form", "learn", "secret", "discover", "youn
| {"predictions": {"tags": ["agent", "deal personal", "murder", "personal", "strange", "victim", "form"
| {"predictions": {"tags": ["wizard", "power", "dark", "learn", "school", "secret", "young", "destroy",
| {"predictions": {"tags": ["alien", "dangerous", "massive", "mission", "planet", "work", "criminal", "
| {"predictions": {"tags": ["murder", "student", "challenge", "help", "school", "face", "woman", "young
| {"predictions": {"tags": ["group", "fall", "friend", "place", "victim", "creature", "survival", "trav
| {"predictions": {"tags": ["fall", "friend", "machine", "secure", "ensure", "scientist", "travel", "de
| {"predictions": {"tags": ["place", "sister", "young", "fight", "woman", "young woman"]}}
| {"predictions": {"tags": ["deal personal", "mysterious", "personal", "secure", "criminal", "powerful"
| {"predictions": {"tags": ["wizard", "dark force", "evil", "include", "journey", "warrior", "wise", "da
| {"predictions": {"tags": ["family", "sister", "challenge", "deal", "woman", "young woman", "young"]}}
| {"predictions": {"tags": ["astronaut", "mission", "earth", "face", "group"]}}
| {"predictions": {"tags": ["world", "real", "real world", "rival", "challenge", "discover", "face", "j
| {"predictions": {"tags": ["student", "form", "learn", "school", "secret", "force", "group"]}}
+-----------------------------------------------------------------------------------------------------
21 rows in set (0.0472 sec)
```

To modify the number of word groups in the ml_results column, you can set the `topk` option. This option must be an integer greater or equal to one. The following example uses the same trained table and adds the option to limit the number of generated word groups to five.

```
mysql> CALL sys.ML_PREDICT_TABLE('topic_modeling_data.movies', @model, 'topic_modeling_data.topic_modeling
                                 JSON_OBJECT('topk', 5));
```

Query the `ml_results` column to review the top five generated word groups.

```
mysql> SELECT ml_results FROM topic_modeling_predictions_five;
+-----------------------------------------------------------------------------------------------+
| ml_results                                                                                    |
+-----------------------------------------------------------------------------------------------+
| {"predictions": {"tags": ["dangerous", "future", "join force", "journey", "warrior"]}}        |
| {"predictions": {"tags": ["machine", "world", "agent", "group rebel", "human"]}}             |
| {"predictions": {"tags": ["family", "empire", "rival", "criminal", "powerful"]}}             |
| {"predictions": {"tags": ["scientist", "astronaut", "attempt", "humanity", "massive"]}}      |
| {"predictions": {"tags": ["include", "mysterious", "strange", "world", "woman"]}}            |
| {"predictions": {"tags": ["empire", "force", "dark force", "evil", "group rebel"]}}          |
| {"predictions": {"tags": ["alien", "attempt", "humanity", "planet", "battle"]}}              |
| {"predictions": {"tags": ["future", "human", "creature", "form", "learn"]}}                  |
```

```
| {"predictions": {"tags": ["agent", "deal personal", "murder", "personal", "strange"]}}       |
| {"predictions": {"tags": ["wizard", "power", "dark", "learn", "school"]}}                    |
| {"predictions": {"tags": ["alien", "dangerous", "massive", "mission", "planet"]}}           |
| {"predictions": {"tags": ["murder", "student", "challenge", "help", "school"]}}             |
| {"predictions": {"tags": ["group", "fall", "friend", "place", "victim"]}}                   |
| {"predictions": {"tags": ["fall", "friend", "machine", "secure", "ensure"]}}                |
| {"predictions": {"tags": ["place", "sister", "young", "fight", "woman"]}}                   |
| {"predictions": {"tags": ["deal personal", "mysterious", "personal", "secure", "criminal"]}} |
| {"predictions": {"tags": ["wizard", "dark force", "evil", "include", "journey"]}}           |
| {"predictions": {"tags": ["family", "sister", "challenge", "deal", "woman"]}}               |
| {"predictions": {"tags": ["astronaut", "mission", "earth", "face", "group"]}}               |
| {"predictions": {"tags": ["world", "real", "real world", "rival", "challenge"]}}            |
| {"predictions": {"tags": ["student", "form", "learn", "school", "secret"]}}                 |
+-----------------------------------------------------------------------------------------+
21 rows in set (0.0475 sec)
```

To learn more about generating predictions for one or more rows of data, see Generate Predictions for a Row of Data.

What's Next

- Review other Machine Learning Use Cases.

6.7.7 Train a Model in an Interim Isolated Environment

The `ML_TRAIN` routine is a compute-intensive operation that may an take extended period of time to complete. To minimize training latency, MySQL HeatWave runs `ML_TRAIN` in parallel across the MySQL HeatWave cluster, consuming significant amount of ECPU and memory resources on each node. During training, these resources are exclusively allocated for AutoML and GenAI operations, and also some background activities. SQL queries offloaded to the RAPID engine are queued until `ML_TRAIN` completes. MySQL HeatWave still allows limited resource sharing with GenAI workloads and other AutoML queries, with a maximum of eight concurrent operations.

In contrast, the `ML_PREDICT_TABLE` and `ML_PREDICT_ROW` routines perform inference by applying a pre-trained machine learning model to input data and generate predictions. These computations are lightweight in terms of ECPU and memory consumption, and process fairly quickly. They can therefore be interleaved with RAPID-offloaded online queries.

`ML_TRAIN` is an infrequent operation, but each time it runs causes significant latency for the queued RAPID offloaded-queries. For this reason, you should not run `ML_TRAIN` on a production MySQL HeatWave DB System that is also used for analytics queries. The recommended architecture is to run `ML_TRAIN` on a dedicated interim MySQL HeatWave DB System provisioned exclusively for model training and brought online only when training or re-training is required. When training completes, you export and deploy the trained model to the production DB System for inference. You can then stop the interim DB System to optimize infrastructure costs.

Before You Begin

- Review how to Create a Machine Learning Model.

Workflow

The following sub-tasks show how to train a machine learning model in an interim DB System and then transfer it to a production DB System for inference.

1. Export the training data to an Object Storage bucket. See Export Training Data to Object Storage.
2. Start the interim MySQL HeatWave DB System. See Start Interim Database.
3. Send the training data from the Object Storage bucket to the interim DB System. See Import Training Data.
4. Train the model in the interim DB System. See Train Model.
5. Send the trained model to an Object Storage bucket. See Export Trained Model to Object Storage.

6. Stop the interim DB System to optimize resource usage and costs. See Stop Interim DB System.
7. Import the trained model to the production DB System. See Import Model to Production DB System.
8. Use the trained model in the production DB System. See Generate Predictions with Trained Model.

To learn more about standardizing this process, see Process Orchestration.

Note

This use case covers the requirement for a primary key or a pseudo primary key when running a model and inference on a DB System configured with MySQL HeatWave High Availability (HA). Group Replication requires that all tables have a primary key, so to successfully run the `ML_TRAIN` and `ML_PREDICT_*` routines you must exclude the pseudo primary key.

The sample data is based on the Iris Data Set Quickstart and the MySQL HeatWave Machine Learning Workshop.

Architecture Overview

This use case uses two DB Systems: one that acts as a production instance, and an interim one used exclusively for training.

The production DB System (instance1) is a MySQL HeatWave HA instance with `sql_generate_invisible_primary_key` enabled. This DB System requires that every table have a primary key, or a pseudo primary key with auto-increment.

The interim DB System (instance2) does not have HA. It is used for training the model. We recommend configuring the interim DB System with the following:

- A 512GB shape.
- Between 1-8 nodes, depending on the size of the training data set.

The following DDL statements show the structure of the training (iris_train) and testing (iris_test) tables:

```
CREATE TABLE `iris_train` (
  `my_row_id` bigint unsigned NOT NULL AUTO_INCREMENT /*!80023 INVISIBLE */,
  `sepal length` float DEFAULT NULL,
  `sepal width` float DEFAULT NULL,
  `petal length` float DEFAULT NULL,
  `petal width` float DEFAULT NULL,
  `class` varchar(16) DEFAULT NULL,
  PRIMARY KEY (`my_row_id`)
) ENGINE=InnoDB AUTO_INCREMENT=121 DEFAULT CHARSET=utf8mb4 COLLATE=utf8mb4_0900_ai_ci


CREATE TABLE `iris_test` (
  `my_row_id` bigint unsigned NOT NULL AUTO_INCREMENT /*!80023 INVISIBLE */,
  `sepal length` float DEFAULT NULL,
  `sepal width` float DEFAULT NULL,
  `petal length` float DEFAULT NULL,
  `petal width` float DEFAULT NULL,
  `class` varchar(16) DEFAULT NULL,
  PRIMARY KEY (`my_row_id`)
) ENGINE=InnoDB AUTO_INCREMENT=31 DEFAULT CHARSET=utf8mb4 COLLATE=utf8mb4_0900_ai_ci
```

The `my_row_id` column is the required column with a pseudo primary key. You must exclude this column when generating predictions and other operations.

Export Training Data to Object Storage

The first sub-task is to use one-liner MySQL Shell to export the training table (`iris_train`) from the production DB System (instance1) to an Object Storage bucket. In this example, the bucket is `demo`.

```
[opc@diagnostics ~]$ mysqlsh admin@instance1 --js -e "util.dumpSchemas(['ml_data'],'ml_data',{'osBucket
'includeTables':['ml_data.iris_train'], 'compatibility': ['strip_definers', 'strip_restricted_grants']}
NOTE: The 'targetVersion' option is set to 9.5.0. This version supports the SET_ANY_DEFINER privilege,
Acquiring global read lock
Global read lock acquired
Initializing - done
1 schemas will be dumped and within them 1 out of 3 tables, 0 views.
Gathering information - done
All transactions have been started
Locking instance for backup
Global read lock has been released
NOTE: When migrating to MySQL HeatWave Service, please always use the latest available version of MySQL
Checking for compatibility with MySQL HeatWave Service 9.5.0
Checking for potential upgrade issues.
NOTE: The value of 'targetVersion' option (9.5.0) is not greater than current version of the server (9.
Validating MySQL HeatWave Service compatibility - done
Compatibility checks finished.
Writing global DDL files
Running data dump using 4 threads.
NOTE: Progress information uses estimated values and may not be accurate.
Writing schema metadata - done
Writing DDL - done
Writing table metadata - done
Starting data dump
100% (120 rows / ~120 rows), 0.00 rows/s, 0.00 B/s uncompressed, 0.00 B/s compressed
Dump duration: 00:00:05
Total duration: 00:00:05
Schemas dumped: 1
Tables dumped: 1
Uncompressed data size: 3.92 KB
Compressed data size: 1.02 KB
Compression ratio: 3.8
Rows written: 120
Bytes written: 1.02 KB
Average uncompressed throughput: 3.92 KB/s
Average compressed throughput: 1.02 KB/s
[opc@diagnostics ~]$
```

Start Interim Database

If needed, create and start the interim DB System (instance2).

To learn more, see the following:

- Creating a DB System
- Starting, Stopping, or Restarting a DB System

Import Training Data

Use one-liner MySQL Shell to import the training table (`iris_train`) to the interim DB System (instance2) by using `loadDump`.

```
[opc@diagnostics ~]$ mysqlsh admin@instance2 --js -- util loadDump 'ml_data' --osBucketName=demo --osNa
Loading DDL and Data from OCI ObjectStorage bucket=demo, prefix='ml_data' using 4 threads.
Opening dump - done
Target is MySQL 9.5.0-u3-cloud (MySQL HeatWave Service). Dump was produced from MySQL 9.5.0-u3-cloud
Fetching dump data from remote location...
Listing files - done
Scanning metadata - done
Checking for pre-existing objects - done
Executing common preamble SQL - done
Executing schema DDL - done
Executing table DDL - done
Executing view DDL - done
Starting data load
100% (3.92 KB / 3.92 KB), 0.00 B/s (0.00 rows/s), 1 / 1 tables done
Building indexes - done
Executing common postamble SQL - done
1 chunks (120 rows, 3.92 KB) for 1 tables in 1 schemas were loaded in 0 sec (avg throughput 3.92 KB/s,
```

```
2 DDL files were executed in 0 sec.
Data load duration: 0 sec
Total duration: 1 sec
0 warnings were reported during the load.
[opc@diagnostics ~]$
```

Train Model

Train the model in the interim DB System (instance2).

1. In instance2, set the model handle for the trained model to `iris_model_handle`.

```
mysql> SET @model_handle='iris_model_handle';
Query OK, 0 rows affected (0.0013 sec)
```

2. Run `ML_TRAIN` and exclude the column with the pseudo primary key (`my_row_id`) by using the `exclude_column_list` option.

```
mysql> CALL sys.ML_TRAIN('ml_data.iris.train','class',JSON_OBJECT('task','classification','exclude_colu
Query OK, 0 rows affected (3 min 37.6367 sec)
```

Export Trained Model to Object Storage

When the training completes in instance2, export the model metadata table by using the `ML_MODEL_EXPORT` routine. Then, export this table to an Object Storage bucket.

1. In instance2, confirm the training table is in the current schema.

```
mysql> SHOW tables;
+----------------------------+
| Tables_in_ml_data          |
+----------------------------+
| iris_train                 |
+----------------------------+
1 row in set (0.0019 sec)
```

2. Run `ML_MODEL_EXPORT`. Include the model handle for the trained table and define the table to export the table to (`iris_model_handle_export`).

```
mysql> CALL sys.ML_MODEL_EXPORT(@model_handle,'ml_data.iris_model_handle_export');
Query OK, 0 rows affected (0.0262 sec)
```

3. Confirm the table is successfully created.

```
mysql> SHOW tables;
+----------------------------+
| Tables_in_ml_data          |
+----------------------------+
| iris_model_handle_export   |
| iris_train                 |
+----------------------------+
2 rows in set (0.0027 sec)
```

4. Disconnect from the DB System.

```
mysql> \q
Bye!
```

5. Use one-liner MySQL Shell to export this table to an Object Storage bucket. This example uses the `demo` bucket.

```
[opc@diagnostics ~]$ mysqlsh admin@instance2 --js -- util exportTable 'ml_data.iris_model_handle_export
--dialect=csv --osBucketName=demo --osNamespace=mysqlpm
Initializing - done
Gathering information - done
Running data dump using 1 threads.
NOTE: Progress information uses estimated values and may not be accurate.
Starting data dump
100% (1 rows / ~1 rows), 0.00 rows/s, 0.00 B/s
```

```
Dump duration: 00:00:00s
Total duration: 00:00:00s
Rows written: 1
Bytes written: 423.63 KB
Data size: 423.63 KB
Average throughput: 423.63 KB/s
[opc@diagnostics ~]$
```

Stop Interim DB System

After exporting the table to Object Storage, you can stop the interim DB System (instance2) to optimize resource usage and costs.

See Starting, Stopping, or Restarting a DB System.

Import Model to Production DB System

Truncate the existing `iris_model_handle_export` table in the production DB System (instance1). Then, import the trained model from the Object Storage bucket into the production DB System in the `iris_model_handle_export` table.

1. In instance1, truncate the `iris_model_handle_export` table that already exists in instance1.

```
mysql> TRUNCATE ml_data.iris_model_handle_export;
Query OK, 0 rows affected (0.0186 sec)
```

2. Enable `js` mode.

```
mysql> \js
Switching to JavaScript mode...
```

3. Import the table from the Object Storage bucket.

```
mysql-js> util.importTable('export_model/iris_model_handle_export.csv',{'schema':'ml_data','table':'
Importing from file 'export_model/iris_model_handle_export.csv' to table 'ml_data'.'iris_model_handl
 *.iris_model_handle_export.csv: Records: 1  Deleted: 0  Skipped: 0  Warnings: 0
  100% (423.63 KB / 423.63 KB), 0.00 B/s
NOTE: 'export_model/iris_model_handle_export.csv' (423.63 KB) was imported in 0.6603 sec at 423.63 K
Total rows affected in ml_data.iris_model_handle_export: Records: 1 Deleted: 0 Skipped: 0 Warnings:
```

4. Switch back to `sql` mode.

```
mysql-js> \sql
Switching to SQL mode... Commands end with ;
```

5. Set the model handle for the imported table.

```
mysql> SET @model_handle='iris_model_handle2';
Query OK, 0 rows affected (0.0046 sec)
```

6. Run `ML_MODEL_IMPORT` to load the data from the imported table into the `iris_model_handle_export` table.

```
mysql> CALL sys.ML_MODEL_IMPORT(NULL, JSON_OBJECT('schema','ml_data','table','iris_model_handle_expo
Query OK, 0 rows affected (0.1041 sec)
```

Generate Predictions with Trained Model

At this stage, the trained model is available. You can use it repeatedly on the production instance to generate predictions and other operations. This example uses `ML_PREDICT_ROW` to perform inference on the `iris_test` data.

If the test table includes explicit primary key columns, you can use `ML_PREDICT_TABLE` directly. However, if the table contains invisible or pseudo primary key columns (as in this example), `ML_PREDICT_ROW` should be used instead, allowing you to omit the pseudo primary key column during inference.

1. In instance1, load the model handle for the trained model.

```
mysql> CALL sys.ML_MODEL_LOAD(@model_handle,NULL);
Query OK, 0 rows affected (34.3230 sec)
```

2. Run ML_PREDICT_ROW to generate predictions on specified rows.

```
mysql> SELECT 'sepal length', 'sepal width', 'petal length', 'petal width',
sys.ML_PREDICT_ROW(JSON_OBJECT('sepal width', "sepal width",
'petal length', "petal length",
'petal width', "petal width"),
@model_handle,NULL) prediction from iris.test
limit 3;
+--------------+-------------+-------------+------------+-----------------------------------------
| sepal length | sepal width | petal length | petal width | prediction
+--------------+-------------+-------------+------------+-----------------------------------------
| 5.9          | 3           | 4.2         | 1.5        | {"Prediction": "Iris-setosa", "ml_results": {
| 6.9          | 3.1         | 5.4         | 2.1        | {"Prediction": "Iris-virginica", "ml_results"
| 5.1          | 3.7         | 1.7         | 0.5        | {"Prediction": "Iris-versicolor", "ml_results
+--------------+-------------+-------------+------------+-----------------------------------------
3 rows in set (21.9953 sec)
```

Process Orchestration

Process orchestration may become essential to standardize this mechanism. This is especially important for handling start/stop events in interim DB Systems, and coordinating data movement between production DB Systems and the interim DB Systems used to run ML_TRAIN.

In OCI, workflow and process orchestration combines services such as Resource Manager, DevOps Pipelines, Events, Functions and many more. Starting and stopping DB Systems can be handled programmatically by running the command line, such as OCI Python SDK or OCI CLI.

To learn more, see the following:

- Section 6.1, “About MySQL HeatWave AutoML”
- Object Storage Buckets
- Instance Dump Utility, Schema Dump Utility, and Table Dump Utility
- Dump Loading Utility
- Table Export Utility
- Parallel Table Import Utility
- Oracle Cloud Infrastructure CLI Command Reference
- CLI for the Database Service
- Compute Service CLI
- MySQL Database Service API: StartDBSystem
- MySQL Database Service API: StopDBSystem

What's Next

- Review other Machine Learning Use Cases.

6.8 Use MySQL HeatWave AutoML with Lakehouse

If you have external data that you want to use with MySQL HeatWave AutoML, you can load it directly from Object Storage with Lakehouse.

This topic has the following sections.

- Before You Begin
- MySQL HeatWave AutoML and Lakehouse Overview
- Workflow to Use MySQL HeatWave AutoML with Lakehouse
- MySQL HeatWave AutoML and Lakehouse Use Case
- Upload File to Object Storage
- Load Data into an External Lakehouse Table
- Use ML_PREDICT_ROW and ML_EXPLAIN_ROW with Lakehouse Data
- What's Next

Before You Begin

- Lakehouse must be enabled in the MySQL HeatWave cluster. See Additional MySQL HeatWave Lakehouse Requirements to learn more.
- Review MySQL HeatWave Lakehouse Privileges.
- Review Supported File Formats for Loading Data.

MySQL HeatWave AutoML and Lakehouse Overview

For MySQL HeatWave on OCI, you have three options to access external data from Object Storage: Pre-Authenticated Requests (PAR), Resource Principals, or Uniform Resource Identifiers (URI) (As of MySQL 9.3.1).

See the following to learn more:

- Resource Principals in the *MySQL HeatWave on OCI Service Guide*.
- Uploading an Object Storage Object to a Bucket in the *MySQL HeatWave on OCI Service Guide*.
- Object Storage Pre-Authenticated Requests in the *MySQL HeatWave on OCI Service Guide*.

After loading data from Object Storage into Lakehouse, you can start running AutoML routines. You can run the following routines as needed with no changes:

- `ML_TRAIN`
- `ML_MODEL_LOAD`
- `ML_EXPLAIN`
- `ML_PREDICT_TABLE`
- `ML_EXPLAIN_TABLE`
- `ML_SCORE`

If you run `ML_PREDICT_ROW` or `ML_EXPLAIN_ROW` on data loaded from Object Storage, you cannot use a `FROM` clause.

MySQL HeatWave AutoML commands operate on data loaded into MySQL HeatWave. If the original Lakehouse data in Object Storage is deleted or modified, this does not affect a MySQL HeatWave AutoML command until the data is unloaded from MySQL HeatWave.

Workflow to Use MySQL HeatWave AutoML with Lakehouse

The workflow to use MySQL HeatWave AutoML with Lakehouse includes the following:

1. Upload the file that has the data to a bucket in your Oracle Cloud account. See Upload File to Object Storage.

2. Load the external file into MySQL HeatWave and create an external Lakehouse table. You have two options to do this. The preferred method is to automate the process with the `heatwave_load` command. See Load Data into an External Lakehouse Table. Alternatively, you can manually load the data. See Section 4.3.12, “Load Structured Data Manually” to learn more.

3. Run MySQL HeatWave AutoML routines as needed. For `ML_PREDICT_ROW` and `ML_EXPLAIN_ROW`, you cannot use the `FROM` clause in Lakehouse tables. See Use ML_PREDICT_ROW and ML_EXPLAIN_ROW with Lakehouse Data to review how to do this.

MySQL HeatWave AutoML and Lakehouse Use Case

The following tasks use data from: Bank Marketing. This data is used for a classification machine learning model, and predicts if a client will subscribe to a term deposit. The target column is `y`.

To access the file from Object Storage, you create a pre-authenticated request. This is a temporary request that allows you to access files in Object Storage. See Object Storage Pre-Authenticated Requests to learn more.

Upload File to Object Storage

To upload the file that has the data to Object Storage:

1. Visit Bank Marketing and download the `bank+marketing.zip` file.

2. Unzip the file, and then unzip the `bank.zip` file. Refer to the `bank.csv` file for the remaining steps.

3. Access your Oracle Cloud account and upload the `bank.csv` file to Object Storage. See Uploading an Object Storage Object to a Bucket.

4. Create a pre-authenticated request for the `bank.csv` file. Select the `Object` *Pre-Authenticated Request Target* and enter the `bank.csv` *Object Name*. See Creating a Pre-Authenticated Request in Object Storage in the *MySQL HeatWave on OCI Service Guide*.

Load Data into an External Lakehouse Table

After uploading the file to Object Storage, set up the name of the table that will store the data, the format of the external file, and the pre-authorized request link to access the external file. You then use the `heatwave_load` command to allow Auto Parallel Load to automatically infer the characteristics of the table columns and load the data into an external Lakehouse table. To learn more, see Section 4.3.11, “Load Structured Data Using Lakehouse Auto Parallel Load”.

1. Connect to your MySQL HeatWave Database System.

2. Create a session variable with the characteristics of the data to load and the external file:

```
mysql> SET @input_list = '[{
  "db_name": "bank_marketing",
  "tables": [{
    "table_name": "bank_train",
    "engine_attribute": {
      "dialect": {"format": "csv", "has_header": true},
      "file": [{"par": "pre_authenticated_request"}]
    }
  }]
}]';
```

Where:

- `@input_list` is the name of the session variable.
- `db_name` identifies the database name to store the table. If the database does not exist, MySQL HeatWave automatically creates it.
- `table_name` sets the table name to store the data: `bank_train`. If the table does not exist, MySQL HeatWave automatically creates it.
- `format` defines the format of the external file: `csv`.
- `has_header` identifies a header in the external file. Auto Parallel Load then infers the column names from the first row in the file.
- `par` sets the pre-authenticated request link to access the file. Replace `pre_authenticated_request` with your own link.

If you are on MySQL 9.1.2 and earlier, you need to update `dialect` with the `field delimiter` and `record delimiter` parameters. Set `field delimiter` with a value of `","`, and `record delimiter` with a value of `"\\n"`. As of MySQL 9.2.0, Lakehouse Autopilot can automatically detect these values. See Lakehouse External Table Syntax to learn more.

3. Use the `HEATWAVE_LOAD` command to automatically load the data into an external Lakehouse table with the default settings. You have the option to test the load before implementing it. See Section 4.3.11, “Load Structured Data Using Lakehouse Auto Parallel Load” to learn more about different modes.

```
mysql> CALL sys.HEATWAVE_LOAD(CAST(@input_list AS JSON), NULL);
+-----------------------------------------+
| INITIALIZING HEATWAVE AUTO PARALLEL LOAD |
+-----------------------------------------+
| Version: 4.24                           |
|                                         |
| Load Mode: normal                       |
| Load Policy: disable_unsupported_columns |
| Output Mode: normal                     |
|                                         |
+-----------------------------------------+
6 rows in set (0.0676 sec)

+-----------------------------------------------------------------------------------------------------------
| LAKEHOUSE AUTO SCHEMA INFERENCE
+-----------------------------------------------------------------------------------------------------------
| Verifying external lakehouse tables: 1
|
| SCHEMA                   TABLE                    TABLE IS           RAW     NUM. OF      ESTIMATE
| NAME                     NAME                     CREATED      FILE SIZE     COLUMNS      ROW COUN
| ------                   -----                    --------     ---------     -------      --------
| `bank_marketing`         `bank_train`             NO          450.66 KiB          17         4.52
|
| New schemas to be created: 0
| External lakehouse tables to be created: 1
|
+-----------------------------------------------------------------------------------------------------------
10 rows in set (0.0676 sec)

+-------------------------------------------------------------------------+
| OFFLOAD ANALYSIS                                                        |
+-------------------------------------------------------------------------+
| Verifying input schemas: 1                                              |
| User excluded items: 0                                                  |
|                                                                         |
| SCHEMA                       OFFLOADABLE    OFFLOADABLE     SUMMARY OF |
| NAME                              TABLES        COLUMNS     ISSUES     |
| ------                       -----------    -----------     ---------- |
| `bank_marketing`                        1             17               |
|                                                                         |
| Total offloadable schemas: 1                                            |
```

```
|                                                                    |
+--------------------------------------------------------------------+
10 rows in set (0.0676 sec)

+-----------------------------------------------------------------------------------------------------------
| CAPACITY ESTIMATION
+-----------------------------------------------------------------------------------------------------------
| Default encoding for string columns: VARLEN (unless specified in the schema)
| Estimating memory footprint for 1 schema(s)
|
|                              TOTAL       ESTIMATED       ESTIMATED       TOTAL     DICTIONARY      V
| SCHEMA                   OFFLOADABLE   HEATWAVE NODE      MYSQL NODE      STRING        ENCODED     E
| NAME                          TABLES       FOOTPRINT       FOOTPRINT     COLUMNS        COLUMNS     C
| ------                   -----------       ---------       ---------     -------     ----------     -
| `bank_marketing`                   1        3.10 MiB        1.12 MiB          10              0
|
| Sufficient MySQL host memory available to load all tables.
| Sufficient HeatWave cluster memory available to load all tables.
|
+-----------------------------------------------------------------------------------------------------------
12 rows in set (0.0676 sec)

+-----------------------------------------------------------------------------------------------------------
| EXECUTING LOAD SCRIPT
+-----------------------------------------------------------------------------------------------------------
| HeatWave Load script generated
|   Retrieve load script containing 2 generated DDL command(s) using the query below:
| Deprecation Notice: "heatwave_load_report" will be deprecated, please switch to "heatwave_autopilot_r
|   SELECT log->>"$.sql" AS "Load Script" FROM sys.heatwave_autopilot_report WHERE type = "sql" ORDER B
|
| Adjusting load parallelism dynamically per internal/external table.
| Using current parallelism of 32 thread(s) as maximum for internal tables.
|
| Warning: Executing the generated script may alter column definitions and secondary engine flags in th
|
| Using SQL_MODE: ONLY_FULL_GROUP_BY,STRICT_TRANS_TABLES,NO_ZERO_IN_DATE,NO_ZERO_DATE,ERROR_FOR_DIVISIO
|
| Proceeding to load 1 table(s) into HeatWave.
|
| Applying changes will take approximately 7.01 s
|
+-----------------------------------------------------------------------------------------------------------
16 rows in set (0.0676 sec)

+---------------------------------------------+
| TABLE LOAD                                  |
+---------------------------------------------+
| TABLE (1 of 1): `bank_marketing`.`bank_train` |
| Commands executed successfully: 2 of 2      |
| Warnings encountered: 0                     |
| Table load succeeded!                       |
|   Total columns loaded: 17                  |
|   Elapsed time: 3.50 s                      |
|                                             |
+---------------------------------------------+
7 rows in set (0.0676 sec)

+-------------------------------------------------------------------------------+
| LOAD SUMMARY                                                                  |
+-------------------------------------------------------------------------------+
|                                                                               |
| SCHEMA                          TABLES       TABLES      COLUMNS         LOAD |
| NAME                            LOADED       FAILED       LOADED     DURATION |
| ------                          ------       ------      -------     -------- |
| `bank_marketing`                     1            0           17       3.50 s |
|                                                                               |
| Total errors encountered: 0                                                   |
| Total warnings encountered: 0                                                 |
|                                                                               |
+-------------------------------------------------------------------------------+
9 rows in set (0.0676 sec)
```

```
Query OK, 0 rows affected (0.0676 sec)
```

A summary is generated that allows you to review the details of the load and any warnings or errors.

Once the data is loaded into Lakehouse, you can use the following routines as needed with no changes:

- `ML_TRAIN`
- `ML_MODEL_LOAD`
- `ML_EXPLAIN`
- `ML_PREDICT_TABLE`
- `ML_EXPLAIN_TABLE`
- `ML_SCORE`

For `ML_PREDICT_ROW` and `ML_EXPLAIN_ROW`, you must avoid the `FROM` clause. The following task shows how to use these commands specifically for data loaded into MySQL HeatWave from Lakehouse.

Use ML_PREDICT_ROW and ML_EXPLAIN_ROW with Lakehouse Data

To avoid using the `FROM` clause, you can insert the data to predict and explain directly into a `JSON` object.

1. Train the table. Setting `NULL` to all JSON options means that the default `task` of `classification` is used for training.

   ```
   mysql> CALL sys.ML_TRAIN('bank_marketing.bank_train', 'y', JSON_OBJECT('task', 'classification'), @b
   ```

   Where:

   - `bank_marketing.bank_train` is the fully qualified name of the table that contains the training dataset (*`schema_name.table_name`*).
   - `y` is the name of the target column, which contains ground truth values.
   - `JSON_OBJECT('task', 'classification')` specifies the machine learning task type.
   - `@bank_model` is the name of the user-defined session variable that stores the model handle for the duration of the connection. User variables are written as *`@var_name`*. Any valid name for a user-defined variable is permitted. For example, `@my_model`. Learn more about Model Handles.

2. After training, load the trained model.

   ```
   mysql> CALL sys.ML_MODEL_LOAD(@bank_model, NULL);
   ```

3. Insert the data to predict and explain directly into a JSON object named `@row_input`.

   ```
   mysql> SET @row_input = JSON_OBJECT(
           'age', 37,
           'job', 'admin.',
           'marital', 'married',
           'education', 'unknown',
           'default', 'no',
           'balance', 734,
           'housing', 'yes',
           'loan', 'no',
           'contact', 'unknown',
   ```

```
        'day', 21,
        'month', 'may',
        'duration', 1106,
        'campaign', 1,
        'pdays', -1,
        'previous', 0,
        'poutcome', 'unknown',
        'y', 'no');
```

4. Run `ML_PREDICT_ROW` to generate predictions on the data in the JSON object. Optionally, use `\G` to view the output in an easily readable format.

```
mysql> SELECT sys.ML_PREDICT_ROW(@row_input, @bank_model, NULL)\G
*************************** 1. row ***************************
sys.ML_PREDICT_ROW(@row_input, @bank_model, NULL):
{"y": "no", "age": 37, "day": 21, "job": "admin.", "loan": "no", "month": "may", "pdays": -1, "balance"
"Prediction": "no", "ml_results": {"predictions": {"y": "no"}, "probabilities": {"no": 0.7052, "yes": 0
1 row in set (1.0027 sec)
```

   The prediction gives a value of `no` with a probability of 65%. This is the same as the labeled value for the row in the `y` column. This demonstrates that the trained model gave a reliable prediction.

5. Run `ML_EXPLAIN_ROW` with the `permutation_importance` prediction explainer to generate an explanation on the prediction previously generated. Optionally, use `\G` to view the output in an easily readable format.

```
mysql> SELECT sys.ML_EXPLAIN_ROW(@row_input, @bank_model,
        JSON_OBJECT('prediction_explainer', 'permutation_importance'))\G
*************************** 1. row ***************************
sys.ML_EXPLAIN_ROW(@row_input, @bank_model,
        JSON_OBJECT('prediction_explainer', 'permutation_importance')): {"y": "no", "age": 37, "day": 2
        "Notes": "month (may) had the largest impact towards predicting no, whereas duration (1106) con
        "month": "may", "pdays": -1, "balance": 734, "contact": "unknown", "default": "no", "housing":
        "Prediction": "no", "ml_results": {"notes": "month (may) had the largest impact towards predict
        "predictions": {"y": "no"}, "attributions": {"month": 0.24, "pdays": 0.02, "contact": 0.15, "de
1 row in set (5.0770 sec)
```

   The explanation determines that the `may` value in the `month` column had the greatest impact on generating the previous prediction of `no`. It also determines that `duration` contributed the most against the prediction of `no`.

What's Next

- Learn how to Load Data from Object Storage into MySQL HeatWave Cluster using MySQL HeatWave Lakehouse.
- Review how to Create a Machine Learning Model.
- Review MySQL HeatWave AutoML Routines
- Review Machine Learning Use Cases to create machine learning models with sample datasets.

6.9 Manage Machine Learning Models

The following sections describe how to manage your machine learning models.

6.9.1 The Model Catalog

MySQL HeatWave AutoML stores machine learning models in a model catalog. A model catalog is a DB System table named `MODEL_CATALOG`. MySQL HeatWave AutoML creates a model catalog for any user that creates a machine learning model.

The `MODEL_CATALOG` table is created in a schema named `ML_SCHEMA_`*`user_name`*, where the *`user_name`* is the name of the owning user.

The fully qualified name of the model catalog table is `ML_SCHEMA_user_name.MODEL_CATALOG`.

When a user creates a model, the `ML_TRAIN` routine creates the model catalog schema and table if they do not exist. `ML_TRAIN` inserts the model as a row in the `MODEL_CATALOG` table at the end of training.

A model catalog is accessible only to the owning user unless the user grants privileges on the model catalog to another user. This means that MySQL HeatWave AutoML routines can only use models that are accessible to the user running the routines. For information about giving access to the model catalog and trained models to other users, see Share a Model.

A database administrator can manage a model catalog table the same way as a regular MySQL table.

6.9.1.1 The Model Catalog Table

The `MODEL_CATALOG` table (`ML_SCHEMA_user_name.MODEL_CATALOG`) has the following columns:

- `model_id`

  A primary key, and a unique auto-incrementing numeric identifier for the model.

- `model_handle`

  A name for the model. The model handle must be unique in the model catalog. The model handle is generated or set by the user when the `ML_TRAIN` routine runs on a training dataset. The generated `model_handle` format is *`schemaName_tableName_userName_No`*, as in the following example: `heatwaveml_bench.census_train_user1_1636729526`. See Work with Model Handles to learn more.

  Note

  The format of the generated model handle is subject to change.

- `model_object`

  A string in JSON format containing the serialized MySQL HeatWave AutoML model.

  As of MySQL 9.0.0, models are stored in the `model_object_catalog` table, and the `model_object` is set to `NULL`.

- `model_owner`

  The user who initiated the `ML_TRAIN` query to create the model.

- `build_timestamp`

  A timestamp indicating when the model was created (in UNIX epoch time). A model is created when the `ML_TRAIN` routine finishes running.

- `target_column_name`

  The name of the column in the training table that was specified as the target column.

- `train_table_name`

  The name of the input table specified in the `ML_TRAIN` query.

- `model_object_size`

  The model object size, in bytes.

- `model_type`

The type of model (algorithm) selected by `ML_TRAIN` to build the model.

- `task`

  The task type specified in the `ML_TRAIN` query.

- `column_names`

  The feature columns used to train the model.

- `model_explanation`

  The model explanation generated during training. See Generate Model Explanations.

- `last_accessed`

  The last time the model was accessed. MySQL HeatWave AutoML routines update this value to the current timestamp when accessing the model.

- `model_metadata`

  Metadata for the model. If an error occurs during training or you cancel the training operation, MySQL HeatWave AutoML records the error status in this column. See Model Metadata.

- `notes`

  Use this column to record notes about the trained model. It also records any error messages that occur during model training.

The Model Object Catalog Table

As of MySQL 9.0.0, models are chunked and stored uncompressed in the `model_object_catalog` table. Each chunk is saved with the same `model_handle`.

A call to one of the following routines upgrades the model catalog, and store the model in the `model_object_catalog` table:

- `ML_TRAIN`
- `ML_MODEL_LOAD`
- `ML_EXPLAIN`
- `ML_MODEL_IMPORT`
- `ML_MODEL_EXPORT`

If the call to one of these routines is not successful or is aborted, then the previous model catalog is still available.

The `model_object_catalog` table has the following columns:

- `chunk_id`

  A primary key, and an auto-incrementing numeric identifier for the chunk. It is unique for the chunks sharing the same `model_handle`.

- `model_handle`

  A primary key, and a foreign key that references `model_handle` in the `MODEL_CATALOG` table.

- `model_object`

  A string in JSON format containing the serialized MySQL HeatWave AutoML model.

See Also

- Review Model Metadata for the Model Catalog Table.
- Review Model Handles and how to retrieve them from the Model Catalog Table.

6.9.1.2 Model Metadata

The `model_metadata` column in the model catalog allows you to view detailed information on trained models. For example, you can view the algorithm used to train the model, the columns in the training table, and values for the model explanation.

When you run the `ML_MODEL_IMPORT` routine, the imported table has a `model_metadata` column that stores the metadata for the table. If you import a model from a table, `model_metadata` stores the name of the database and table. If you import a model object, `model_metadata` stores a JSON_OBJECT that contains key-value pairs of the metadata See Section 10.2.4, “ML_MODEL_IMPORT” to learn more.

The default value for `model_metadata` is `NULL`.

This topic has the following sections.

- Model Metadata Details
- Query Model Metadata
- See Also

Model Metadata Details

The metadata in `model_metadata` is in key-value pairs in JSON format. It includes the following:

- `task: string`

  The task type specified in the `ML_TRAIN` query. The default is `classification` when used with `ML_MODEL_IMPORT`.

- `notes: string`

  The `notes` specified in the `ML_TRAIN` query. It also records any error messages that occur during model training.

- `chunks: number`

  Available as of MySQL 9.0.0. The total number of chunks that the model has been split into.

- `format: string`

  The model can be in one of the following formats (MySQL 9.0.0 and later):

  - HWMLv1.0
  - HWMLv2.0
  - ONNXv1.0
  - ONNXv2.0

  For MySQL 8.4.0 and earlier, MySQL HeatWave AutoML models are in `HWMLv1.0` format, and ONNX models are in `ONNX` format. When importing a model with the `ML_MODEL_IMPORT` routine, the default is `ONNX` format.

- `n_rows: number`

  The number of rows in the training table.

- `status: string`

  The status of the model. The default is `Ready` when used with `ML_MODEL_IMPORT`.

  - `Creating`: The model is being created.
  - `Ready`: The model is trained and active.
  - `Error`: Either training was canceled or an error occurred during training. Any error message appears in the `notes` column. The error message also appears in `model_metadata` `notes`.

- `options: JSON object literal`

  The `options` specified in the `ML_TRAIN` query.

- `n_columns: number`

  The number of columns in the training table.

- `column_names: JSON array`

  The feature columns used to train the model.

- `contamination: number`

  The contamination factor for the anomaly detection task. See Anomaly Detection Options to learn more.

- `model_quality: string`

  The quality of the model object for classification and regression tasks. For other tasks, this value is `NULL`. The value is either `low` or `high`.

- `training_time: number`

  The time in seconds taken to train the model.

- `algorithm_name: string`

  The name of the chosen algorithm.

- `training_score: number`

  The cross-validation score achieved for the model by training.

- `build_timestamp: number`

  A timestamp indicating when the model was created (UNIX epoch time). A model is created when the `ML_TRAIN` routine finishes executing.

- `hyperparameters: JSON object literal`

  Available as of MySQL 9.4.0. The hyperparameters of the selected algorithm for the model. For models trained for topic modeling, this value is empty since the topic modeling task does not involve hyperparameters.

- `n_selected_rows: number`

  The number of rows selected by adaptive sampling.

- `training_params: JSON object literal`

  Internal task dependent parameters used during `ML_TRAIN`.

- `train_table_name: string`

  The name of the input table specified in the `ML_TRAIN` query.

- `model_explanation: JSON object literal`

  The model explanation generated during training. See Generate Model Explanations.

- `n_selected_columns: number`

  The number of columns selected by feature selection.

- `target_column_name: string`

  The name of the column in the training table that was specified as the target column.

- `optimization_metric: string`

  The optimization metric used for training. See Section 10.2.16, “Optimization and Scoring Metrics” to review available metrics.

- `selected_column_names: JSON array`

  The names of the columns selected by feature selection.

- `training_drift_metric: JSON object literal`

  Contains data drift information about the training data, see: Section 6.9.8, “Analyze Data Drift”. This only applies to classification, regression, and anomaly detection (as of MySQL 9.3.2) models.

  - `mean: number`

    The mean value of drift metrics of all the training data. ≥ `0`.

  - `variance: number`

    The variance value of drift metrics of all the training data. ≥ `0`.

  Both `mean` and `variance` should be low.

- `onnx_inputs_info: JSON object literal`

  Information about the format of the ONNX model inputs. This only applies to ONNX models. See Manage External ONNX Models.

  Do not provide `onnx_inputs_info` if the model is not ONNX format. This generates an error.

  - `data_types_map: JSON object literal`

    This maps the data type of each column to an ONNX model data type. The default value is:

    ```
    JSON_OBJECT("tensor(int64)": "int64", "tensor(float)": "float32", "tensor(string)": "str_")
    ```

- `onnx_outputs_info: JSON object literal`

  Information about the format of the ONNX model outputs. This only applies to ONNX models. See Manage External ONNX Models.

  Do not provide `onnx_outputs_info` if the model is not ONNX format, or if `task` is `NULL`. This generates an error.

  - `predictions_name: string`

    This name determines which of the ONNX model outputs is associated with predictions.

- `prediction_probabilities_name: string`

  This name determines which of the ONNX model outputs is associated with prediction probabilities.

- `labels_map: JSON object literal`

  This maps prediction probabilities to predictions, known as labels.

Query Model Metadata

You can query the model metadata in the model catalog with the following command. Replace `user1` with your own user name.

```
mysql> SELECT JSON_PRETTY(model_metadata) FROM ML_SCHEMA_user1.MODEL_CATALOG\G
*************************** 1. row ***************************
JSON_PRETTY(model_metadata): {
  "task": "classification",
  "notes": null,
  "chunks": 1,
  "format": "HWMLv2.0",
  "n_rows": 4521,
  "status": "Ready",
  "options": {
    "task": "classification",
    "model_explainer": "permutation_importance",
    "prediction_explainer": "permutation_importance"
  },
  "n_columns": 16,
  "column_names": [
    "age",
    "job",
    "marital",
    "education",
    "default1",
    "balance",
    "housing",
    "loan",
    "contact",
    "day",
    "month",
    "duration",
    "campaign",
    "pdays",
    "previous",
    "poutcome"
  ],
  "contamination": null,
  "model_quality": "high",
  "training_time": 95.27952575683594,
  "algorithm_name": "XGBClassifier",
  "training_score": -0.22748015820980072,
  "build_timestamp": 1751312812,
  "hyperparameters": {
    "booster": "gbtree",
    "max_depth": 2,
    "reg_alpha": 0.0,
    "reg_lambda": 5.62341325190349,
    "n_estimators": 50,
    "learning_rate": 0.0001,
    "min_child_weight": 0
  },
  "n_selected_rows": 3176,
  "training_params": {
    "recommend": "ratings",
    "force_use_X": false,
    "recommend_k": 3,
    "remove_seen": true,
    "ranking_topk": 10,
```

```
    "lsa_components": 100,
    "ranking_threshold": 1,
    "feedback_threshold": 1
  },
  "train_table_name": "bank_marketing.bank_marketing_train",
  "model_explanation": {
    "permutation_importance": {
      "age": 0.0,
      "day": 0.0,
      "job": 0.0,
      "loan": -0.0009,
      "month": 0.0194,
      "pdays": 0.0134,
      "balance": 0.0,
      "contact": 0.0099,
      "housing": 0.005,
      "marital": 0.0,
      "campaign": 0.0,
      "default1": 0.0,
      "duration": 0.0706,
      "poutcome": 0.0169,
      "previous": 0.0077,
      "education": 0.0
    }
  },
  "n_selected_columns": 8,
  "target_column_name": "y",
  "optimization_metric": "neg_log_loss",
  "selected_column_names": [
    "contact",
    "duration",
    "housing",
    "loan",
    "month",
    "pdays",
    "poutcome",
    "previous"
  ],
  "training_drift_metric": {
    "mean": 0.3065,
    "variance": 0.3748
  }
}
```


See Also

- Analyze Data Drift
- Manage External ONNX Models
- The Model Catalog Table
- Generate Model Explanations

6.9.2 Work with Model Handles

When `ML_TRAIN` trains a model, you have the option to specify a name for the model, which is the model handle. If you do not specify a model handle name, a model handle is automatically generated that is based on the database name, input table name, the user name training the table, and a unique numerical identifier. You must use model handles to run MySQL HeatWave AutoML routines. All model handles must be unique in the model catalog.

This topic has the following sections.

- Before You Begin
- Model Handles Overview
- Query the Model Handle

- Define Model Handle
- Assign Session Variable to Model Handle
- What's Next

Before You Begin

- Review the The Model Catalog.

Model Handles Overview

If you set the model handle name to a session variable before training a model, the model handle takes that name. Otherwise, a unique model handle is automatically generated. To set your own model name, see Define Model Handle. The model handle is stored temporarily in a user-defined session variable specified in the `ML_TRAIN` call. In the following example, `@census_model` is defined as the model handle session variable with no set model handle name:

```
mysql> CALL sys.ML_TRAIN('heatwaveml_bench.census_train', 'revenue', JSON_OBJECT('task', 'classification'),
```

While the connection used to run `ML_TRAIN` remains active, that connection can retrieve the automatically generated model handle by querying the session variable. For example:

```
mysql> SELECT @census_model;
+-----------------------------------------------+
| @census_model                                 |
+-----------------------------------------------+
| census_classification_model                   |
+-----------------------------------------------+
```

Query the Model Handle

Since the session variable for a model handle is only valid for the current session, you can query the model handle name from the model catalog in new sessions.

The following example queries the model handle, the model owner, and the name of the training table from the model catalog table. Replace `user1` with your own user name.

```
mysql> SELECT model_handle, model_owner, train_table_name FROM ML_SCHEMA_user1.MODEL_CATALOG;
+-----------------------------------------------------------------------+-------------+-----------------------
| model_handle                                                          | model_owner | train_table_name
+-----------------------------------------------------------------------+-------------+-----------------------
| census_classification_model                                           | user1       | census_data.census_trai
+-----------------------------------------------------------------------+-------------+-----------------------
18 rows in set (0.0014 sec)
```

Once you have the model handle, you can use it directly in AutoML routines instead of the session variable.

The following example runs `ML_PREDICT_ROW` and uses the model handle.

```
mysql> SELECT sys.ML_PREDICT_ROW(@row_input, 'census_classification_model', NULL);
```

Define Model Handle

Before training a model, it is good practice to define your own model handle instead of automatically generating one. This allows you to easily remember the model handle for future routines on the trained model instead of having to query it, or depending on the session variable that can no longer be used when the current connection terminates.

To define your own model handle:

1. Set the value of the session variable, which sets the model handle to this same value.

   ```
   mysql> SET @variable = 'model_handle';
   ```

Replace *@variable* and *model_handle* with your own definitions. For example:

```
mysql> SET @census_model = 'census_classification_model';
```

When ML_TRAIN runs with this session variable, the model handle is set to census_test.

If you set a model handle that already appears in the model catalog, the ML_TRAIN routine returns an error.

2. Run the ML_TRAIN routine.

```
mysql> CALL sys.ML_TRAIN('table_name', 'target_column_name', JSON_OBJECT('task', 'task_name'), @vari
```

Replace *table_name*, *target_column_name*, *task_name*, and *variable* with your own values.

The following example trains a model with the model handle variable previously set

```
mysql> CALL sys.ML_TRAIN('heatwaveml_bench.census_train', 'revenue', JSON_OBJECT('task', 'classifica
```

3. After training the model, query the model catalog to confirm the model handle you defined is there. Replace user1 with your own user name.

```
mysql> SELECT model_handle, model_owner, train_table_name FROM ML_SCHEMA_user1.MODEL_CATALOG;
+-------------------------------------------------------------------+-------------+-----------------
| model_handle                                                      | model_owner | train_table_name
+-------------------------------------------------------------------+-------------+-----------------
| census_classification_model                                       | user1       | census_data.cens
+-------------------------------------------------------------------+-------------+-----------------
1 row in set (0.0014 sec)
```

Assign Session Variable to Model Handle

If you lose the session variable to a model handle due to a lost connection, you have the option of assigning a new session variable to a model handle in a new connection.

To assign a session variable to a model handle:

1. Set a variable to the model handle. If needed Query the Model Handle.

```
mysql> SET @my_model = 'model_handle';
```

The following example sets the @my_model session variable to a model handle.

```
mysql> SET @my_model = 'census_classification_model';
```

2. Confirm the session variable is assigned to the model handle by querying the session variable.

```
mysql> SELECT @my_model;
+-----------------------------------------------------+
| @my_model                                           |
+-----------------------------------------------------+
| census_classification_model                         |
+-----------------------------------------------------+
```

Alternatively, you can assign a session variable to the model handle for the most recently trained model.

1. Set a variable with the query to retrieve the most recent model handle by sorting with the build_timestamp parameter in the model catalog. Replace user1 with your own user name.

```
mysql> SET @variable = (SELECT model_handle FROM ML_SCHEMA_user1.MODEL_CATALOG ORDER BY build_timest
```

The following example sets the latest_model variable.

```
mysql> SET @latest_model = (SELECT model_handle FROM ML_SCHEMA_user1.MODEL_CATALOG ORDER BY timestam
```

2. Confirm the session variable is assigned to the latest model handle by querying the session variable.

```
mysql> SELECT @latest_model;
+-------------------------+
| @latest_model           |
+-------------------------+
| recommendation_use_case4 |
+-------------------------+
1 row in set (0.0454 sec)
```

What's Next

- Review how to Create a Machine Learning Model.
- Review Machine Learning Use Cases to create machine learning models with sample datasets.

6.9.3 Unload a Model

The `ML_MODEL_UNLOAD` routine unloads a model from MySQL HeatWave AutoML. For `ML_MODEL_UNLOAD` parameter descriptions, see ML_MODEL_UNLOAD.

Before You Begin

- Review the following
  - Train a Model
  - Load a Model

Unload a Model

If you are on MySQL 9.0.0 or higher, you can verify what models are currently loaded with the `ML_MODEL_ACTIVE` routine before and after unloading the model.

1. If you are on MySQL 9.0.0 or higher, verify what models are currently loaded with the `ML_MODEL_ACTIVE` routine.

```
mysql> CALL sys.ML_MODEL_ACTIVE('all', @model_info);
```

2. Select the session variable created to view all loaded models.

```
mysql> SELECT JSON_PRETTY(@model_info);
+------------------------------------------------------------+
| JSON_PRETTY(@model_info)                                   |
+------------------------------------------------------------+
| [
  {
    "total model size(bytes)": 50209
  },
  {
    "user1": [
      {
        "recommendation_use_case": {
          "format": "HWMLv2.0",
          "model_size(byte)": 15609
        }
      },
      {
        "recommendation_use_case2": {
          "format": "HWMLv2.0",
          "model_size(byte)": 8766
        }
      },
      {
        "recommendation_use_case3": {
          "format": "HWMLv2.0",
          "model_size(byte)": 8402
```

```
        }
      },
      {
        "recommendation_use_case4": {
          "format": "HWMLv2.0",
          "model_size(byte)": 17432
        }
      }
    ]
  }
] |
+-----------------------------------------------------------+
1 row in set (0.0411 sec)
```

3. Refer to the appropriate model handle to unload. Alternatively, use the session variable for the model handle.

   The following example unloads a model by using the model handle:

   ```
   mysql> CALL sys.ML_MODEL_UNLOAD('recommendation_use_case');
   ```

   Where:

   - `recommendation_use_case` is the model handle.

   The following example unloads a model by using the session variable for the model handle:

   ```
   mysql> CALL sys.ML_MODEL_UNLOAD(@recommendation_model);
   ```

   Where:

   - `@recommendation_model` is the assigned session variable for the model handle.

4. If you are on MySQL 9.0.0, run `ML_MODEL_ACTIVE` again to confirm the model is successfully unloaded

   ```
   mysql> CALL sys.ML_MODEL_ACTIVE('all', @model_info);
   mysql> SELECT JSON_PRETTY(@model_info);
   +-----------------------------------------------------------+
   | JSON_PRETTY(@model_info)                                  |
   +-----------------------------------------------------------+
   | [
     {
       "total model size(bytes)": 34600
     },
     {
       "user1": [
         {
           "recommendation_use_case2": {
             "format": "HWMLv2.0",
             "model_size(byte)": 8766
           }
         },
         {
           "recommendation_use_case3": {
             "format": "HWMLv2.0",
             "model_size(byte)": 8402
           }
         },
         {
           "recommendation_use_case4": {
             "format": "HWMLv2.0",
             "model_size(byte)": 17432
           }
         }
       ]
     }
   ] |
   +-----------------------------------------------------------+
   1 row in set (0.0411 sec)
   ```

The list of loaded models shows the model is unloaded.

What's Next

- Review how to Create a Machine Learning Model.
- Review Machine Learning Use Cases to create machine learning models with sample datasets.

6.9.4 View Model Details

To view the details for the models in your model catalog, query the `MODEL_CATALOG` table.

Before You Begin

- Review the following:
  - Create a Machine Learning Model
  - The Model Catalog

View Details for Your Models

The following example queries `model_id`, `model_handle`, and `model_owner`, `train_table_name` from the model catalog. Replace `user1` with your own user name.

```
mysql> SELECT model_id, model_handle, model_owner, train_table_name FROM ML_SCHEMA_user1.MODEL_CATALOG;
+----------+------------------------------------------------------+-------------+----------------------------
| model_id | model_handle                                         | model_owner | train_table_name
+----------+------------------------------------------------------+-------------+----------------------------
|        1 | classification_use_case                              | user1       | classification_data.traini
|        2 | regression_use_case                                  | user1       | regression_data.house_pric
|        3 | forecasting_use_case                                 | user1       | forecasting_data.electrici
|        4 | anomaly_use_case                                     | user1       | anomaly_detection_data.cre
|        5 | recommendation_use_case_default                      | user1       | recommendations_data.user_
|        6 | topic_modeling_use_case                              | user1       | topic_modeling_data.movies
+----------+------------------------------------------------------+-------------+----------------------------
6 rows in set (0.0436 sec)
```

Where:

- `model_id` is a unique numeric identifier for the model.
- `model_owner` is the user that created the model.
- `model_handle` is the handle by which the model is called.
- `ML_SCHEMA_user1.MODEL_CATALOG` is the fully qualified name of the `MODEL_CATALOG` table. The schema is named for the owning user.

The output displays details from only a few `MODEL_CATALOG` table columns. For other columns you can query, see The Model Catalog.

View Model Explanations

The `ML_EXPLAIN` routine generates model explanations and stores them in the model catalog. See Generate Model Explanations to learn more.

A model explanation helps you identify the features that are most important to the model overall. Feature importance is presented as an attribution value. A positive value indicates that a feature contributed toward the prediction. A negative value can have different interpretations depending on the specific model explainer used for the model. For example, a negative value for the permutation importance explainer means that the feature is not important.

To view a model explanation, you can query the `model_explanation` column from the model catalog by referencing the model handle. Review how to Query the Model Handle.

```
mysql> SELECT column FROM ML_SCHEMA_user name.MODEL_CATALOG where model_handle='model_handle';
```

The following example queries one of the model handles and views the model explanation for that model. Optionally, use `JSON_PRETTY` to view the output in an easily readable format.

```
mysql> SELECT JSON_PRETTY(model_explanation) FROM ML_SCHEMA_user1.MODEL_CATALOG where model_handle='cen
+-----------------------------------------------------------------------------------------------------------
| JSON_PRETTY(model_explanation)
+-----------------------------------------------------------------------------------------------------------
| {
  "permutation_importance": {
    "age": 0.0305,
    "sex": 0.0023,
    "race": 0.0017,
    "fnlwgt": 0.0025,
    "education": 0.0013,
    "workclass": 0.0043,
    "occupation": 0.0229,
    "capital-gain": 0.0495,
    "capital-loss": 0.0156,
    "relationship": 0.0267,
    "education-num": 0.0371,
    "hours-per-week": 0.0142,
    "marital-status": 0.0267,
    "native-country": 0.0
  }
} |
+-----------------------------------------------------------------------------------------------------------
1 row in set (0.0447 sec)
```

Where:

- `ML_SCHEMA_user1.MODEL_CATALOG` is the fully qualified name of the `MODEL_CATALOG` table. The schema is named for the user that created the model.
- `census_data.census_train_user1_1744548610842` is the model handle. See Work with Model Handles.

The output displays feature importance values for each column by using the `permutation_importance` model explainer.

Alternatively, you can query the model explanation by using the valid session variable for the model handle. Optionally, use `JSON_PRETTY` to view the output in an easily readable format.

```
mysql> SELECT JSON_PRETTY(model_explanation) FROM ML_SCHEMA_admin.MODEL_CATALOG where model_handle=@cen
+-----------------------------------------------------------------------------------------------------------
| JSON_PRETTY(model_explanation)
+-----------------------------------------------------------------------------------------------------------
| {
  "permutation_importance": {
    "age": 0.0305,
    "sex": 0.0023,
    "race": 0.0017,
    "fnlwgt": 0.0025,
    "education": 0.0013,
    "workclass": 0.0043,
    "occupation": 0.0229,
    "capital-gain": 0.0495,
    "capital-loss": 0.0156,
    "relationship": 0.0267,
    "education-num": 0.0371,
    "hours-per-week": 0.0142,
    "marital-status": 0.0267,
    "native-country": 0.0
  }
} |
+-----------------------------------------------------------------------------------------------------------
1 row in set (0.0447 sec)
```

See Work with Model Handles to learn more.

What's Next

- Review the The Model Catalog.
- Review how to Work with Model Handles.

6.9.5 Delete a Model

Users that create models or have the required privileges to a model on the `MODEL_CATALOG` table can delete them.

Before You Begin

- Review how to Create a Machine Learning Model.
- Review how to Share a Model.

Delete a Model

To delete a model from the model catalog table:

1. Query the model catalog table for the `model_id`, `model_owner`, and `train_table_name`. Identify the `model_id` for model you want to delete. Replace `user1` with your own user name.

```
mysql> SELECT model_id, model_owner, train_table_name FROM ML_SCHEMA_user1.MODEL_CATALOG;
+----------+-------------+-----------------------------------------------+
| model_id | model_owner | train_table_name                              |
+----------+-------------+-----------------------------------------------+
|        1 | user1       | ml_benchmark.sentiment_model_creation         |
|        2 | user1       | ml_data.iris_train                            |
|        3 | user1       | census_data.census_train                      |
+----------+-------------+-----------------------------------------------+
3 rows in set (0.0008 sec)
```

   The requested columns from the model catalog table display.

   In this case, the model with `model_id` 3 is deleted.

2. Delete the model from the model catalog table.

```
mysql> DELETE FROM ML_SCHEMA_user1.MODEL_CATALOG WHERE model_id = 3;
```

   Where:

   - `ML_SCHEMA_user1.MODEL_CATALOG` is the fully qualified name of the `MODEL_CATALOG` table. The schema is named for the user that created the model.
   - `model_id = 3` is the ID of the model you want to delete.

3. Confirm the model is removed from the model catalog table. Replace *`user1`* with your own user name.

```
mysql> SELECT model_id, model_owner, train_table_name FROM ML_SCHEMA_user1.MODEL_CATALOG;
+----------+-------------+-----------------------------------------------+
| model_id | model_owner | train_table_name                              |
+----------+-------------+-----------------------------------------------+
|        1 | user1       | ml_benchmark.sentiment_model_creation         |
|        2 | user1       | ml_data.iris_train                            |
+----------+-------------+-----------------------------------------------+
2 rows in set (0.0008 sec)
```

What's Next

- Review Machine Learning Use Cases to create machine learning models with sample datasets.

6.9.6 Share a Model

This topic describes how to grant other users access to a model you create.

This topic has the following sections.

- Before You Begin
- Share Your Models
- Export the Model to Share
- Set Up Other User with Required Privileges
- Import Shared Model
- Run AutoML Routines on Imported Model
- What's Next

Before You Begin

- The `ML_MODEL_EXPORT` routine is available as of MySQL 9.0.0. Review ML_MODEL_EXPORT and ML_MODEL_IMPORT for more information on required versions.
- Review MySQL HeatWave AutoML Privileges.
- To follow along with the file used in the tasks, see Use MySQL HeatWave AutoML with Lakehouse.

Share Your Models

To share a model you created, you can use the `ML_MODEL_EXPORT` and `ML_MODEL_IMPORT` routines. `ML_MODEL_EXPORT` exports the model to share to a user-defined table that both users need the required privileges to access. `ML_MODEL_IMPORT` imports the model to the user's model catalog. The other user can then run AutoML commands on the imported model.

In the following tasks, the `admin` user gives access to their model to the `user1` user. The trained table, `bank_train`, is in the `bank_marketing` database. To follow along with the file used, see Use MySQL HeatWave AutoML with Lakehouse.

Export the Model to Share

The `admin` user needs to export the model to share to a user-defined table that both users can access. In this use case, the user exports the model to their own model catalog.

1. As the `admin` user, train and load the model to export. See Train a Model and Load a Model.
2. Export the model to a table in the model catalog. Use the assigned session variable for the model handle. If you need to query the model handle, see Work with Model Handles.

   ```
   mysql> CALL sys.ML_MODEL_EXPORT (model_handle, output_table_name);
   ```

   Replace *`model_handle`* and *`output_table_name`* with your own values. For example:

   ```
   mysql> CALL sys.ML_MODEL_EXPORT(@bank_model, 'ML_SCHEMA_admin.model_export');
   ```

   Where:

   - `@bank_model` is the assigned session variable for the model handle of the trained model.
   - `ML_SCHEMA_admin.model_export` is the fully qualified name of the table that contains the training dataset (*`schema_name.table_name`*).
3. Run the `SHOW CREATE TABLE` command to confirm the table was created with the recommended parameters for importing. See ML_MODEL_IMPORT to learn more.

```
mysql> SHOW CREATE TABLE ML_SCHEMA_admin.model_export;
+--------------+-----------------------------------------------------------------------------------------
| Table        | Create Table
+--------------+-----------------------------------------------------------------------------------------
| model_export | CREATE TABLE `model_export` (
  `chunk_id` int NOT NULL AUTO_INCREMENT,
  `model_object` longtext,
  `model_metadata` json DEFAULT NULL,
  PRIMARY KEY (`chunk_id`)
) ENGINE=InnoDB AUTO_INCREMENT=2 DEFAULT CHARSET=utf8mb4 COLLATE=utf8mb4_0900_ai_ci |
+--------------+-----------------------------------------------------------------------------------------
1 row in set (0.0527 sec)
```

Set Up Other User with Required Privileges

The `admin` user needs to grant the required privileges to `user1`, so that user can access exported model and import it into their own model catalog.

1. If not done already, create the other user account (user1). See `CREATE USER`.
2. Run these commmands to grant the required privileges to the other user, so they can access the following:
   - MySQL HeatWave AutoML routines on the MySQL sys schema.
   - The model catalog for both users.
   - The database with the trained model.
   - External Lakehouse tables with the machine learning model dataset.

   See MySQL HeatWave AutoML Privileges to learn more.

```
mysql> GRANT SELECT, EXECUTE ON sys.* TO 'user1'@'%';
mysql> GRANT SELECT, ALTER, INSERT, CREATE, UPDATE, DROP, GRANT OPTION ON ML_SCHEMA_user1.* TO 'user1'@
mysql> GRANT SELECT, ALTER, INSERT, CREATE, UPDATE, DROP, GRANT OPTION ON ML_SCHEMA_admin.* TO 'user1'@
mysql> GRANT SELECT, ALTER, INSERT, CREATE, UPDATE, DROP, GRANT OPTION ON bank_marketing.* TO 'user1'@'
mysql> GRANT SELECT ON performance_schema.rpd_tables TO 'user1'@'%';
mysql> GRANT SELECT ON performance_schema.rpd_table_id TO 'user1'@'%';
mysql> GRANT SELECT ON performance_schema.rpd_query_stats TO 'user1'@'%';
mysql> GRANT SELECT ON performance_schema.rpd_ml_stats TO 'user1'@'%';
```

   Where:

   - `ML_SCHEMA_user1.*` and `ML_SCHEMA_user1.*` gives access to the model catalog for both users.
   - `bank_marketing` is the database that contains the trained table.
   - The remaining tables are required if the trained table is a Lakehouse external table.

Import Shared Model

The `user1` user can now import the exported model to their own model catalog.

1. Log in to the DB system as the other user (user1).
2. Import the model the `admin` user previously exported into the model catalog for `user1`.

```
mysql> CALL sys.ML_MODEL_IMPORT (model_object, model_metadata, model_handle);
```

   Replace *`model_object`*, *`model_metadata`*, and *`model_handle`* with your own values. For example:

```
mysql> CALL sys.ML_MODEL_IMPORT(NULL, JSON_OBJECT('schema', 'ML_SCHEMA_admin', 'table', 'model_export')
```

- `NULL` means that a model from a table is imported, and not a model object.
- `JSON_OBJECT` sets key-value pairs for the database and table of the exported table to import.
- `@bank_export` is the assigned session variable for the imported model handle.

3. Load the imported model. Use the assigned session variable set for the imported model handle in the previous command.

```
mysql> CALL sys.ML_MODEL_LOAD(@bank_export, NULL);
```

4. Optionally, query `model_object` and `model_object_size` from the model catalog for the loaded model to confirm the model imported successfully.

```
mysql> SELECT model_object, model_object_size FROM ML_SCHEMA_user1.MODEL_CATALOG WHERE model_handle=
+--------------+-------------------+
| model_object | model_object_size |
+--------------+-------------------+
| NULL         |            331860 |
+--------------+-------------------+
1 row in set (0.0478 sec)
```

Confirm the `model_object_size` is not 0.

5. Optionally, query `chunk_id` and `LENGTH(model_object)` from the model object catalog for the loaded model to confirm the model imported successfully.

```
mysql> SELECT chunk_id, LENGTH(model_object) FROM ML_SCHEMA_user1.model_object_catalog WHERE model_h
+----------+----------------------+
| chunk_id | LENGTH(model_object) |
+----------+----------------------+
|        1 |               331860 |
+----------+----------------------+
1 row in set (0.0465 sec)
```

Confirm the `chunk_id` value is 1 and `LENGTH(model_object)` is not 0.

Run AutoML Routines on Imported Model

Confirm the `user1` user can run AutoML commands. The following example generates a table of predictions for the imported model.

```
mysql> CALL sys.ML_PREDICT_TABLE(table_name, model_handle, output_table_name), [options]);
```

Replace *`table_name`*, *`model_handle`*, *`output_table_name`*), and *`options`* with your own values. For example:

```
mysql> CALL sys.ML_PREDICT_TABLE('bank_marketing.bank_train', @bank_export, 'bank_marketing.bank_predic
```

Where:

- `bank_marketing.bank_train` is the fully qualified name of the table that contains the training dataset (*`schema_name.table_name`*).
- `@bank_export` is the assigned session variable for the imported model handle.
- `bank_marketing.bank_predictions` is the fully qualified name of the output table that contains the predictions (*`schema_name.table_name`*).

Optionally, use the database with the output table and query a sample.

```
mysql> USE bank_marketing;
mysql> SELECT * FROM bank_predictions limit 5;
+-------------------+-----+--------------+---------+-----------+---------+---------+---------+------+--
| _4aad19ca6e_pk_id | age | job          | marital | education | default | balance | housing | loan | c
+-------------------+-----+--------------+---------+-----------+---------+---------+---------+------+--
```

```
|                 1 |  30 | management   | single  | tertiary  | no      |     149 | yes     | no    | unkno
|                 2 |  46 | blue-collar  | married | secondary | no      |   -1400 | yes     | no    | telep
|                 3 |  33 | entrepreneur | married | secondary | no      |    -118 | yes     | yes   | unkno
|                 4 |  43 | blue-collar  | married | secondary | no      |    2160 | no      | no    | cellu
|                 5 |  38 | management   | married | tertiary  | no      |    3452 | no      | no    | cellu
+-------------------+-----+--------------+---------+-----------+---------+---------+---------+-------+------
5 rows in set (0.0425 sec)
```

What's Next

- Review Machine Learning Use Cases to create machine learning models with sample datasets.

6.9.7 Manage External ONNX Models

MySQL HeatWave AutoML supports the upload of pre-trained models in ONNX (Open Neural Network Exchange) format to the model catalog. Load them with the `ML_MODEL_IMPORT` routine. After import, you can use MySQL HeatWave AutoML routines with ONNX models.

6.9.7.1 ONNX Models Overview

You cannot directly load models in ONNX format (`.onnx`) into a MySQL table. The models require string serialization and conversion to Base64 encoding before you use the `ML_MODEL_IMPORT` routine.

MySQL HeatWave AutoML supports the following ONNX model types:

- An ONNX model that has only one input, and it is the entire MySQL table.
- An ONNX model that has more than one input, and each input is one column in the MySQL table.

For example, MySQL HeatWave AutoML does not support an ONNX model that takes more than one input, and each input is associated with more than one column in the MySQL table.

The first dimension of the input to the ONNX model provided by the ONNX model `get_inputs()` API should be the batch size. This should be `None`, a string, or an integer. `None` or string indicate a variable batch size, and an integer indicates a fixed batch size.

Examples of input shapes:

```
[None, 2]

['batch_size', 2, 3]

[1, 14]
```

All other dimensions should be integers. For example, MySQL HeatWave AutoML does not support an input shape similar to the following:

```
input shape = ['batch_size', 'sequence_length']
```

The output of an ONNX model is a list of results. The ONNX API documentation defines the results as a numpy array, a list, a dictionary, or a sparse tensor. MySQL HeatWave AutoML only supports a numpy array, a list, and a dictionary.

- Numpy array examples:

```
array(['Iris-virginica', 'Iris-versicolor', 'Iris-versicolor', 'Iris-versicolor'], dtype=object)

array([0, 2, 0, 0], dtype=int64)

array([[0.8896357 , 0.11036429],
       [0.28360802, 0.716392  ],
       [0.9404001 , 0.05959991],
       [0.5655978 , 0.43440223]], dtype=float32)

array([[0.96875435],
       [1.081366  ],
       [0.5736201 ],
```

```
       [0.90711355]], dtype=float32)
```

- Simple list examples:

```
['Iris-virginica', 'Iris-versicolor', 'Iris-versicolor', 'Iris-versicolor']

[0, 2, 0, 0]
```

- List of lists examples:

```
[[0.8896357 , 0.110364],
[0.28360802, 0.716392],
[0.9404001 , 0.059599],
[0.5655978 , 0.434402]]

[[[0.8896357] , [0.110364]],
[[0.28360802], [0.716392]],
[[0.9404001] , [0.059599]],
[[0.5655978] , [0.434402]]]

[[0.968754],
[1.081366],
[0.573620],
[0.907113]]

[[[0.968754]],
[[1.081366]],
[[0.573620]],
[[0.907113]]]
```

- Dictionary examples:

```
{'Iris-setosa': 0.0, 'Iris-versicolor': 0.0, 'Iris-virginica': 0.999}

{0: 0.1, 1: 0.9}
```

- List of dictionaries examples:

```
[{'Iris-setosa': 0.0, 'Iris-versicolor': 0.0, 'Iris-virginica': 0.999},
{'Iris-setosa': 0.0, 'Iris-versicolor': 0.999, 'Iris-virginica': 0.0},
{'Iris-setosa': 0.0, 'Iris-versicolor': 0.589, 'Iris-virginica': 0.409},
{'Iris-setosa': 0.0, 'Iris-versicolor': 0.809, 'Iris-virginica': 0.190}]

[{0: 1.0, 1: 0.0, 2: 0.0},
{0: 0.0, 1: 0.0, 2: 1.0},
{0: 1.0, 1: 0.0, 2: 0.0},
{0: 1.0, 1: 0.0, 2: 0.0}]

[{0: 0.176, 1: 0.823},
{0: 0.176, 1: 0.823},
{0: 0.264, 1: 0.735},
{0: 0.875, 1: 0.124}]

[{0: 0.176, 1: 0.823},
{0: 0.176, 1: 0.823},
{0: 0.264, 1: 0.735},
{0: 0.875, 1: 0.124}]

[{0: 0.176, 1: 0.823},
{0: 0.176, 1: 0.823},
{0: 0.264, 1: 0.735},
{0: 0.875, 1: 0.124}]
```

For classification and regression tasks, MySQL HeatWave AutoML only supports model explainers and scoring for variable batch sizes.

For forecasting, anomaly detection and recommendation tasks, MySQL HeatWave AutoML does not support model explainers and scoring. The prediction column must contain a JSON object literal of name value keys. For example, for three outputs:

```
{output1: value1, output2: value2, output3: value3}
```

What's Next

- Learn about ONNX Model Metadata.

6.9.7.2 ONNX Model Metadata

To learn more about model metadata in the model catalog, see Model Metadata. The model metadata includes `onnx_inputs_info` and `onnx_outputs_info`.

- `onnx_inputs_info` includes `data_types_map`. See Model Metadata for the default value.
- `onnx_outputs_info` includes `predictions_name`, `prediction_probabilities_name`, and `labels_map`.

ONNX Inputs Info

Use the `data_types_map` to map the data type of each column to an ONNX model data type. For example, to convert inputs of the type `tensor(float)` to `float64`:

```
data_types_map = {"tensor(float)": "float64"}
```

MySQL HeatWave AutoML first checks the user `data_types_map`, and then the default `data_types_map` to check if the data type exists. MySQL HeatWave AutoML supports the following numpy data types:

Table 6.1 Supported numpy data types

`str_`	`unicode_`	`int8`	`int16`	`int32`	`int64`	`int_`	`uint16`
`uint32`	`uint64`	`byte`	`ubyte`	`short`	`ushort`	`intc`	`uintc`
`uint`	`longlong`	`ulonglong`	`intp`	`uintp`	`float16`	`float32`	`float64`
`half`	`single`	`longfloat`	`double`	`longdouble`	`bool_`	`datetime64`	`complex_`
`complex64`	`complex128`	`complex256`	`csingle`	`cdouble`	`clongdouble`		

The use of any other numpy data type causes an error.

ONNX Outputs Info

Use `predictions_name` to determine which of the ONNX model outputs is associated with predictions. Use `prediction_probabilities_name` to determine which of the ONNX model outputs is associated with prediction probabilities. Use use a `labels_map` to map prediction probabilities to predictions, known as labels.

For regression tasks:

- If the ONNX model generates only one output, then `predictions_name` is optional.
- If the ONNX model generates more than one output, then `predictions_name` is required.
- Do not provide `prediction_probabilities_name` as this causes an error.

For classification tasks:

- Use `predictions_name`, `prediction_probabilities_name`, or both. Failure to provide at least one causes an error.
- The model explainers SHAP, Fast SHAP, and Partial Dependence require `prediction_probabilities_name`.

Only use a `labels_map` with classification tasks. A `labels_map` requires `predictions_probabilities_name`. The use of a `labels_map` with any other task, or with `predictions_name` or without `predictions_probabilities_name` causes an error.

If the task is `NULL`, do not provide `predictions_name` or `prediction_probabilities_name` as this causes an error.

An example of a `predictions_probabilities_name` with a `labels_map` produces these labels:

```
predictions_probabilities_name = array([[0.35, 0.50, 0.15],
                                        [0.10, 0.20, 0.70],
                                        [0.90, 0.05, 0.05],
                                        [0.55, 0.05, 0.40]], dtype=float32)

labels_map = {0:'Iris-virginica', 1:'Iris-versicolor', 2:'Iris-setosa'}

labels=['Iris-versicolor', 'Iris-setosa', 'Iris-virginica', 'Iris-virginica']
```

MySQL HeatWave AutoML adds a note for ONNX models that have inputs with four dimensions about the reshaping of data to a suitable shape for an ONNX model. This would typically be for ONNX models that are trained on image data.

An example of this note added to the `ml_results` column:

```
mysql> CALL sys.ML_PREDICT_TABLE('mlcorpus_v5.mnist_test_temp', @model,
          'mlcorpus_v5.`mnist_predictions`', NULL);
Query OK, 0 rows affected (20.6296 sec)

mysql> SELECT ml_results FROM mnist_predictions;;
+-----------------------------------------------------------------------------------------------------------------
| ml_results
+-----------------------------------------------------------------------------------------------------------------
| {'predictions': {'prediction': 7}, 'Notes': 'Input data is reshaped into (1, 28, 28).', 'probabilitie
+-----------------------------------------------------------------------------------------------------------------
```

See Also

- Review The Model Catalog.

6.9.7.3 Import an External ONNX Model

This topic describes how to import an external ONNX model.

This topic has the following sections. Refer to the steps to import an ONNX model. There are also examples for your reference.

- Before You Begin
- Ways to Import External ONNX Model
- Workflow to Import an ONNX Model
- Encode ONNX File
- Prepare to Import ONNX Model as a Pre-Processed Object
- Prepare to Import ONNX Model as a Table
- Define Model Metadata
- Import ONNX Model as a Pre-processed Object
- Import ONNX Model as a Table
- ONNX Import Examples
- What's Next

Before You Begin

- Review the following:
  - ONNX Models Overview
  - ONNX Model Metadata

- Review ONNX Model Metadata.

Ways to Import External ONNX Model

You have the following ways to import an external ONNX model.

- Import model as a string: For smaller models, you can copy the encoded string and paste it into a session variable or temporary table column. You can then import the table with the copied string. To do this, you run the `ML_MODEL_IMPORT` routine and import the model as a pre-processed model object.
- Import model directly from a table (available as of MySQL 9.0.0): For larger models, you can load the entire file into a table with the appropriate parameters. You can then import the table directly into your model catalog. If needed, you can load the model in batches of smaller files. To do this, you run the `ML_MODEL_IMPORT` routine and import the model as a table.

  The table that you load the model into must have the following columns:

  - `chunk_id`: The recommended parameters are `INT AUTO_INCREMENT PRIMARY KEY`. There must be only one row in the table with `chunk_id` = 1.
  - `model_object`: The recommended parameters are `LONGTEXT NOT NULL`.
  - `model_metadata`: The recommended parameters are `JSON DEFAULT NULL`.

Workflow to Import an ONNX Model

The workflow to import an ONNX model includes the following:

1. Convert the ONNX file to Base 64 encoding and carry out sting serialization. See Encode ONNX File.
2. Depending on the size of the model, select if you want to import the model as a string in a pre-processed model object (smaller files) or as a table (larger files). Then, refer to the appropriate section to prepare the model file. See either Prepare to Import ONNX Model as a Pre-Processed Object or Prepare to Import ONNX Model as a Table.
3. Define the model metadata as needed depending on the type of machine learning task for the model. See Define Model Metadata.
4. Import the model by using the `ML_MODEL_IMPORT` routine. See either Import ONNX Model as a Pre-processed Object to import the model as a string or Import ONNX Model as a Table.

Encode ONNX File

Before importing an ONNX model, you must convert the ONNX file to Base 64 encoding and carry out string serialization. Do this with the Python base64 module. Ensure you have the appropriate version of Python installed.

To encode the ONNX file:

1. Open a terminal window (command prompt on Windows).
2. Install the ONNX library.

   ```
   pip install onnx
   ```

3. Launch Python and run the following code.

   ```
   # python3 encode_onnx_base64.py
   import onnx
   import base64

   with open("output_file_name", "wb") as f:
   ```

```
    model = onnx.load("input_file_name")
    f.write(base64.b64encode(model.SerializeToString()))
```

Replace `input_file_name` with the full file path to the ONNX file and `output_file_name` with the desired file name for the encoded file. If needed, set a file path for the output file.

The following example converts the `/Users/user1/iris.onnx` file and creates the output file `iris_base64.onnx`.

```
# python3 encode_onnx_base64.py
import onnx
import base64

with open("iris_base64.onnx", "wb") as f:
    model = onnx.load("/Users/user1/iris.onnx")
    f.write(base64.b64encode(model.SerializeToString()))
```

After encoding the ONNX file, select the method to import the model and review the appropriate steps.

- Prepare to Import ONNX Model as a Pre-Processed Object
- Prepare to Import ONNX Model as a Table

Prepare to Import ONNX Model as a Pre-Processed Object

For smaller model files, you can import the ONNX model as a string into a pre-processed object.

To prepare to import the ONNX Model as a string:

1. Open the encoded file and copy the string.
2. Connect to the DB System.
3. Copy and paste the converted string for the file into a session variable. For example:

```
mysql> SET @onnx_string_model_object='ONNX_file_string';
```

Alternatively, you can load the encoded file directly into a table column. To do this, you must create a classic MySQL session and set the appropriate `local-infile` setting for the client. The server setting `local_infile=ON` is enabled by default. Verify with your admin before using these settings. See Security Considerations for LOAD DATA LOCAL to learn more.

Depending on how you connect to your DB System (Compute Instance or VPN), follow the appropriate steps.

VPN Connection

If you connect the DB System through a VPN connection, you can load the ONNX file directly from your computer.

1. Open a terminal window (or command prompt on Windows).
2. Start MySQL Shell.

```
> mysqlsh
```

3. Connect to the DB System. To enable the loading of the file, you must create a session in classic MySQL protocol (--mysql) and set the client setting to `local-infile=1`.

```
> \connect --mysql MySQL_username@DB System_private_IP?local-infile=1
```

4. Create and use the database to store the table. For example:

```
mysql> CREATE DATABASE onnx_model;
mysql> USE onnx_model;
```

5. Create a table with only one column to store the string.

The following example creates the `onnx_temp` table with the `onnx_string` column with the `LONGTEXT` data type.

```
mysql> CREATE TABLE onnx_temp (onnx_string LONGTEXT);
```

6. Use a `LOAD DATA LOCAL INFILE` statement to load the pre-processed `.onnx` file into the temporary table. Provide the full file path for the file.

   The following example loads the `iris_base64.onnx` file with the string into the `onnx_string` column in the `onnx_temp` table.

   ```
   mysql> LOAD DATA LOCAL INFILE '/Users/user1/iris_base64.onnx'
          INTO TABLE onnx_temp
          CHARACTER SET binary
          FIELDS TERMINATED BY '\t'
          LINES TERMINATED BY '\r' (onnx_string);
   ```

7. Insert the loaded string into a session variable.

   The following example loads the loaded string in the `onnx_string` column into the `@onnx_table_model_object` session variable.

   ```
   mysql> SELECT onnx_string FROM onnx_temp INTO @onnx_table_model_object;
   ```

After preparing the model, you can Define Model Metadata.

Compute Instance Connection

If you connect the DB System through a compute instance, you need to the upload the ONNX file to the compute instance before loading the file into the DB System.

1. From a terminal window, upload the ONNX file to the appropriate directory of the compute instance.

   ```
   $> scp -v -i ssh-key.key /Users/user1/iris_base64.onnx opc@ComputeInstancePublicIP:/home/opc/
   ```

   Replace the following:

   - *`ssh-key.key`*: The full file path to the SSH key file (.key) for the compute instance.
   - */`Users/user1/iris_base64.onnx`*: The full file path to the ONNX file on your device.
   - *`opc@ComputeInstancePublicIP`*: The appropriate username and public IP for the compute instance.
   - */`home/opc/`*: The appropriate file path to upload the ONNX file to in the compute instance.

2. Once the upload successfully completes, SSH into the compute instance.

   ```
   $> ssh -i ssh-key.key opc@computeInstancePublicIP
   ```

   Replace the following:

   - *`ssh-key.key`*: The full file path to the SSH key file (.key) for the compute instance.
   - *`opc@ComputeInstancePublicIP`*: The appropriate username and public IP for the compute instance.

3. Start MySQL Shell.

   ```
   $> mysqlsh
   ```

4. Connect to the DB System. To enable the loading of the file, you must create a session in classic MySQL protocol (--mysql) and set the client setting to `local-infile=1`.

   ```
   > \connect --mysql MySQL_username@DB System_private_IP?local-infile=1
   ```

5. Create and use the database to store the table. For example:

```
mysql> CREATE DATABASE onnx_model;
mysql> USE onnx_model;
```

6. Create a table with only one column to store the string.

   The following example creates the `onnx_temp` table with the `onnx_string` column with the `LONGTEXT` data type.

```
mysql> CREATE TABLE onnx_temp (onnx_string LONGTEXT);
```

7. Use a `LOAD DATA LOCAL INFILE` statement to load the pre-processed `.onnx` file into the temporary table. If you uploaded the file to a directory other than `:/home/opc/`, provide the full file path to the file.

   The following example loads the `iris_base64.onnx` file with the string into the `onnx_string` column in the `onnx_temp` table.

```
mysql> LOAD DATA LOCAL INFILE 'iris_base64.onnx'
       INTO TABLE onnx_temp
       CHARACTER SET binary
       FIELDS TERMINATED BY '\t'
       LINES TERMINATED BY '\r' (onnx_string);
```

8. Insert the loaded string into a session variable.

   The following example loads the loaded string in the `onnx_string` column into the `@onnx_table_model_object` session variable.

```
mysql> SELECT onnx_string FROM onnx_temp INTO @onnx_table_model_object;
```

After preparing the model, you can Define Model Metadata.

Prepare to Import ONNX Model as a Table

For larger model files, you must import the model as a table. To do this, you must create a classic MySQL session and set the appropriate `local-infile` setting for the client. The server setting `local_infile=ON` is enabled by default. Verify with your admin before using these settings. See Security Considerations for LOAD DATA LOCAL to learn more.

Depending on how you connect to your DB System (Compute Instance or VPN), follow the appropriate steps.

VPN Connection

If you connect the DB System through a VPN connection, you can load the ONNX file directly from your computer.

1. Open a terminal window (or command prompt on Windows).
2. Start MySQL Shell.

```
> mysqlsh
```

3. Connect to the DB System. To enable the loading of the file, you must create a session in classic MySQL protocol (--mysql) and set the client setting to `local-infile=1`.

```
> \connect --mysql MySQL_username@DB System_private_IP?local-infile=1
```

4. Create and use the database to store the table. For example:

```
mysql> CREATE DATABASE onnx_model;
mysql> USE onnx_model;
```

5. Create a table to store the model. The table must have the three required columns to store the details for the model (`chunk_id`, `model_object`, and `model_metadata`). See ML_MODEL_IMPORT Overview. For example:

```
mysql> CREATE TABLE model_table (
      chunk_id INT AUTO_INCREMENT PRIMARY KEY,
      model_object LONGTEXT NOT NULL,
      model_metadata JSON DEFAULT NULL);
```

6. Use a `LOAD DATA LOCAL INFILE` statement to load the model. If needed, load the model in batches of files depending on the size of the model. See LOAD DATA Statement to learn more. Make sure to provide the full file path to the ONNX file on your device. The following example loads the model in three separate files into the `model_object` column in the `model_table` table previously created:

```
mysql> LOAD DATA LOCAL INFILE '/onnx_examples/x00'
       INTO TABLE model_table
       CHARACTER SET binary
       FIELDS TERMINATED BY '\t'
       LINES TERMINATED BY '\r'
       (model_object);
Query OK, 1 row affected (34.96 sec)
Records: 1  Deleted: 0  Skipped: 0  Warnings: 0

mysql> LOAD DATA LOCAL INFILE '/onnx_examples/x01'
       INTO TABLE model_table
       CHARACTER SET binary
       FIELDS TERMINATED BY '\t'
       LINES TERMINATED BY '\r'
       (model_object);
Query OK, 1 row affected (32.74 sec)
Records: 1  Deleted: 0  Skipped: 0  Warnings: 0

mysql> LOAD DATA LOCAL INFILE '/onnx_examples/x02'
       INTO TABLE model_table
       CHARACTER SET binary
       FIELDS TERMINATED BY '\t'
       LINES TERMINATED BY '\r'
       (model_object);
Query OK, 1 row affected (11.90 sec)
Records: 1  Deleted: 0  Skipped: 0  Warnings: 0
```

After preparing the model, you can Define Model Metadata.

Compute Instance Connection

If you connect the DB System through a compute instance, you need to the upload the ONNX file to the compute instance before loading the file into the DB System.

1. From a terminal window, upload the ONNX file to the appropriate directory of the compute instance.

```
$> scp -v -i ssh-key.key /Users/user1/iris_base64.onnx opc@ComputeInstancePublicIP:/home/opc/
```

   Replace the following:

   - *`ssh-key.key`*: The full file path to the SSH key file (.key) for the compute instance.
   - */`Users/user1/iris_base64.onnx`*: The full file path to the ONNX file on your device.
   - *`opc@ComputeInstancePublicIP`*: The appropriate username and public IP for the compute instance.
   - */`home/opc/`*: The appropriate file path to upload the ONNX file to in the compute instance.

2. Once the upload successfully completes, SSH into the compute instance.

```
$> ssh -i ssh-key.key opc@computeInstancePublicIP
```

Replace the following:

- *ssh-key.key*: The full file path to the SSH key file (.key) for the compute instance.
- *opc@ComputeInstancePublicIP*: The appropriate username and public IP for the compute instance.

3. Start MySQL Shell.

```
$> mysqlsh
```

4. Connect to the DB System. To enable the loading of the file, you must create a session in classic MySQL protocol (--mysql) and set the client setting to `local-infile=1`.

```
> \connect --mysql MySQL_username@DB System_private_IP?local-infile=1
```

5. Create and use the database to store the table. For example:

```
mysql> CREATE DATABASE onnx_model;
mysql> USE onnx_model;
```

6. Create a table to store the model. The table must have the three required columns to store the details for the model (`chunk_id`, `model_object`, and `model_metadata`). See ML_MODEL_IMPORT Overview. For example:

```
mysql> CREATE TABLE model_table (chunk_id INT AUTO_INCREMENT PRIMARY KEY, model_object LONGTEXT NOT
```

7. Use a `LOAD DATA LOCAL INFILE` statement to load the model. If needed, load the model in batches of files depending on the size of the model. See LOAD DATA Statement to learn more. If you uploaded the file to a directory other than `:/home/opc/`, provide the full file path to the file. The following example loads the model in three separate files into the `model_object` column in the `model_table` table previously created:

```
mysql> LOAD DATA LOCAL INFILE '/onnx_examples/x00'
       INTO TABLE model_table
       CHARACTER SET binary
       FIELDS TERMINATED BY '\t'
       LINES TERMINATED BY '\r'
       (model_object);
Query OK, 1 row affected (34.96 sec)
Records: 1  Deleted: 0  Skipped: 0  Warnings: 0

mysql> LOAD DATA LOCAL INFILE '/onnx_examples/x01'
       INTO TABLE model_table
       CHARACTER SET binary
       FIELDS TERMINATED BY '\t'
       LINES TERMINATED BY '\r'
       (model_object);
Query OK, 1 row affected (32.74 sec)
Records: 1  Deleted: 0  Skipped: 0  Warnings: 0

mysql> LOAD DATA LOCAL INFILE '/onnx_examples/x02'
       INTO TABLE model_table
       CHARACTER SET binary
       FIELDS TERMINATED BY '\t'
       LINES TERMINATED BY '\r'
       (model_object);
Query OK, 1 row affected (11.90 sec)
Records: 1  Deleted: 0  Skipped: 0  Warnings: 0
```

After preparing the model, you can Define Model Metadata.

Define Model Metadata

After preparing the ONNX model (either as a string or table), define the metadata for the model as required. See Model Metadata and ONNX Model Metadata to learn more about requirements depending on the task type of the model.

To define the metadata for the ONNX model:

1. If including the column names for the model in the metadata, you have the option to set them into a JSON object as key-value pairs.

   ```
   mysql> SET @variable = JSON_OBJECT("key","value"[,"key","value"] ...);
   ```

   For example:

   ```
   mysql> SET @column_names = JSON_OBJECT("0","f1", "1","f2", "2","f3");
   ```

2. Set the metadata for the model as required into a JSON object as key-value pairs. To learn more about metadata requirements, see ONNX Model Metadata. You can also include additional information that allows you to properly configure input tables and columns for generating predictions.

   ```
   mysql> SET @variable = JSON_OBJECT("key","value"[,"key","value"] ...);
   ```

   The following example shows how to define the metadata if you import the model as a string (pre-processed object). The `predictions_name` and `prediction_probabilities_name` variables are provided because it is a classification task. Including the `column_names` allows you to refer to the metadata to ensure that input tables for predictions have the same details. Otherwise an error generates.

   ```
   mysql> SET @model_metadata = JSON_OBJECT('task','classification',
                                            'onnx_outputs_info', JSON_OBJECT('predictions_name','label','
                                            'target_column_name','target',
                                            'train_table_name','mlcorpus.`classification_3_table`',
                                            'column_names',@column_names,
                                            'notes','user notes for the model',
                                            'training_score',0.734,
                                            'training_time',100.34,
                                            'n_rows',1000,
                                            'n_columns',3,
                                            'algorithm_name','xgboost');
   ```

   The following example shows how to define the metadata if you import the model from a table. The `predictions_name` and `prediction_probabilities_name` variables are provided because it is a classification task. After defining the metadata, update the metadata for the temporary table for the row that is `chunk_id=1`.

   ```
   mysql> SET @model_metadata = JSON_OBJECT('task','classification',
                                            'onnx_outputs_info', JSON_OBJECT('predictions_name','label','
                                            'target_column_name','target');

   mysql> UPDATE mlcorpus.model_table SET model_metadata=@model_metadata WHERE chunk_id=1;
   ```

Depending on how you prepared the model, follow the appropriate steps to import the model:

- Import ONNX Model as a Pre-processed Object
- Import ONNX Model as a Table

Import ONNX Model as a Pre-processed Object

If you followed the steps to Prepare to Import ONNX Model as a Pre-Processed Object, review the following steps to import the model as a pre-processed object.

To import the model as a pre-processed object:

1. Optionally, define the model handle for the imported model instead of automatically generating one. See Work with Model Handles.

   ```
   mysql> SET @variable = 'model_handle';
   ```

   For example:

```
mysql> SET @model = 'onnx_model_string';
```

2. Run ML_MODEL_IMPORT to import the model.

```
mysql> CALL sys.ML_MODEL_IMPORT (model_object, model_metadata, model_handle);
```

Since you are importing a pre-processed object, the model_object is defined by the string you previously set in the in either the @onnx_string_model_object or @onnx_table_model_object session variable. The model_metadata is defined by the metadata previously set in the @model_metadata session variable. The model_handle is defined by the session variable created for the model handle.

See the following example:

```
mysql> CALL sys.ML_MODEL_IMPORT(@onnx_string_model_object, @model_metadata, @model);
```

3. Confirm the model successfully loaded by querying the model_id and model_handle from the model catalog. Query the model by using the model handle previously created. Replace user1 with your own MySQL user name.

```
mysql> SELECT model_id, model_handle FROM ML_SCHEMA_user1.MODEL_CATALOG WHERE model_handle='onnx_mod
+----------+--------------------+
| model_id | model_handle       |
+----------+--------------------+
|        1 | onnx_model_table   |
+----------+--------------------+
1 row in set (0.0485 sec)
```

4. To load the model into MySQL HeatWave so you can start using it with MySQL HeatWave routines, run ML_MODEL_LOAD.

```
mysql> CALL sys.ML_MODEL_LOAD(model_handle, NULL);
```

For example:

```
mysql> CALL sys.ML_MODEL_LOAD(@model, NULL);
```

Import ONNX Model as a Table

If you followed the steps to Prepare to Import ONNX Model as a Table, review the following steps to import the model as a table.

To import the model as a table:

1. Optionally, define the model handle for the imported model instead of automatically generating one. See Work with Model Handles.

```
mysql> SET @variable = 'model_handle';
```

For example:

```
mysql> SET @model = 'onnx_model_table';
```

2. Run ML_MODEL_IMPORT to import the model.

```
mysql> CALL sys.ML_MODEL_IMPORT (model_object, model_metadata, model_handle);
```

Since you are importing a table, the model_object is set to NULL. The model_metadata is defined by the schema name and table name storing the string for the ONNX model. The metadata for the model is stored in the table when following the steps to Define Model Metadata. The model_handle is defined by the session variable created for the model handle.

See the following example:

```
mysql> CALL sys.ML_MODEL_IMPORT(NULL, JSON_OBJECT('schema', 'onnx_models', 'table', 'model_table'),
```

3. Confirm the model successfully loaded by querying the `model_id` and `model_handle` from the model catalog. Query the model by using the model handle previously created. Replace `user1` with your own MySQL user name.

```
mysql> SELECT model_id, model_handle FROM ML_SCHEMA_user1.MODEL_CATALOG WHERE model_handle='onnx_model_
+----------+--------------------+
| model_id | model_handle       |
+----------+--------------------+
|        2 | onnx_model_table   |
+----------+--------------------+
1 row in set (0.0485 sec)
```

4. To load the model into MySQL HeatWave so you can start using it with MySQL HeatWave routines, run `ML_MODEL_LOAD`.

```
mysql> CALL sys.ML_MODEL_LOAD(model_handle, NULL);
```

For example:

```
mysql> CALL sys.ML_MODEL_LOAD(@model, NULL);
```

ONNX Import Examples

Review the following additional examples for importing ONNX models.

- In the following example, a ONNX model for classification is imported. Then, the model is used to generate predictions, a score, and prediction explainers for a dataset in MySQL HeatWave.

```
mysql> SET @model = 'sklearn_pipeline_classification_3_onnx';
Query OK, 0 rows affected (0.0003 sec)

mysql> SET @model_metadata = JSON_OBJECT('task','classification',
                                         'onnx_outputs_info', JSON_OBJECT('predictions_name','label','pr
Query OK, 0 rows affected (0.0003 sec)

mysql> CALL sys.ML_MODEL_IMPORT(@onnx_encode_sklearn_pipeline_classification_3, @model_metadata, @model);
Query OK, 0 rows affected (1.2438 sec)

mysql> CALL sys.ML_MODEL_LOAD(@model, NULL);
Query OK, 0 rows affected (0.5372 sec)

mysql> CALL sys.ML_PREDICT_TABLE('mlcorpus.classification_3_predict', @model, 'mlcorpus.predictions', NUL
Query OK, 0 rows affected (0.8743 sec)

mysql> SELECT * FROM mlcorpus.predictions;
+------------------+----+----+-----+------------+-----------------------------------------------------
| _4aad19ca6e_pk_id | f1 | f2 | f3  | Prediction | ml_results
+------------------+----+----+-----+------------+-----------------------------------------------------
|                1 | a  | 20 | 1.2 |          0 | {"predictions": {"prediction": 0}, "probabilities": {"
|                2 | b  | 21 | 3.6 |          1 | {"predictions": {"prediction": 1}, "probabilities": {"
|                3 | c  | 19 | 7.8 |          1 | {"predictions": {"prediction": 1}, "probabilities": {"
|                4 | d  | 18 |   9 |          0 | {"predictions": {"prediction": 0}, "probabilities": {"
|                5 | e  | 17 | 3.6 |          1 | {"predictions": {"prediction": 1}, "probabilities": {"
+------------------+----+----+-----+------------+-----------------------------------------------------
5 rows in set (0.0005 sec)

mysql> CALL sys.ML_SCORE('mlcorpus.classification_3_table','target', @model, 'accuracy', @score, NULL);
Query OK, 0 rows affected (0.9573 sec)

mysql> SELECT @score;
+--------+
| @score |
+--------+
|      1 |
+--------+
1 row in set (0.0003 sec)

mysql> CALL sys.ML_EXPLAIN('mlcorpus.classification_3_table', 'target', @model,
                           JSON_OBJECT('model_explainer', 'shap', 'prediction_explainer', 'shap'));
Query OK, 0 rows affected (10.1771 sec)
```

```
mysql> SELECT model_explanation FROM ML_SCHEMA_root.MODEL_CATALOG WHERE model_handle=@model;
+-----------------------------------------------------+
| model_explanation                                   |
+-----------------------------------------------------+
| {"shap": {"f1": 0.0928, "f2": 0.0007, "f3": 0.0039}} |
+-----------------------------------------------------+
1 row in set (0.0005 sec)

mysql> CALL sys.ML_EXPLAIN_TABLE('mlcorpus.classification_3_predict', @model, 'mlcorpus.explanations_
                                 JSON_OBJECT('prediction_explainer', 'shap'));
Query OK, 0 rows affected (7.6577 sec)

mysql> SELECT * FROM mlcorpus.explanations_shap;
+------------------+----+----+-----+------------+----------------+-----------------+----------------
| _4aad19ca6e_pk_id | f1 | f2 | f3  | Prediction | f1_attribution | f2_attribution  | f3_attribution
+------------------+----+----+-----+------------+----------------+-----------------+----------------
|                1 | a  | 20 | 1.2 |          0 |       0.116909 |     0.000591494 |    -0.00524929
|                2 | b  | 21 | 3.6 |          1 |      0.0772133 |     -0.00110559 |     0.00219658
|                3 | c  | 19 | 7.8 |          1 |      0.0781372 | 0.0000000913938 |    -0.00324671
|                4 | d  | 18 |   9 |          0 |       0.115209 |    -0.000592354 |     0.00639341
|                5 | e  | 17 | 3.6 |          1 |      0.0767679 |      0.00110463 |     0.00219425
+------------------+----+----+-----+------------+----------------+-----------------+----------------
5 rows in set (0.0005 sec)
```

- In the following example, a ONNX model for regression is imported. Then, the model is used to generate predictions, a score, and prediction explainers for a dataset in MySQL HeatWave.

```
mysql> SET @model = 'sklearn_pipeline_regression_2_onnx';
Query OK, 0 rows affected (0.0003 sec)

mysql> SET @model_metadata = JSON_OBJECT('task','regression', 'onnx_outputs_info',JSON_OBJECT('predic
Query OK, 0 rows affected (0.0003 sec)

mysql> CALL sys.ML_MODEL_IMPORT(@onnx_encode_sklearn_pipeline_regression_2, @model_metadata, @model);
Query OK, 0 rows affected (1.0652 sec)

mysql> CALL sys.ML_MODEL_LOAD(@model, NULL);
Query OK, 0 rows affected (0.5141 sec)

mysql> CALL sys.ML_PREDICT_TABLE('mlcorpus.regression_2_table', @model, 'mlcorpus.predictions', NULL)
Query OK, 0 rows affected (0.8902 sec)

mysql> SELECT * FROM mlcorpus.predictions;
+------------------+----+----+-----+--------+------------+-----------------------------------------
| _4aad19ca6e_pk_id | f1 | f2 | f3  | target | Prediction | ml_results
+------------------+----+----+-----+--------+------------+-----------------------------------------
|                1 | a  | 20 | 1.2 |   22.4 |     22.262 | {"predictions": {"prediction": 22.2620391
|                2 | b  | 21 | 3.6 |   32.9 |    32.4861 | {"predictions": {"prediction": 32.4861145
|                3 | c  | 19 | 7.8 |   56.8 |    56.2482 | {"predictions": {"prediction": 56.2481536
|                4 | d  | 18 |   9 |   31.8 |       31.8 | {"predictions": {"prediction": 31.8000011
|                5 | e  | 17 | 3.6 |   56.4 |    55.9861 | {"predictions": {"prediction": 55.9861145
+------------------+----+----+-----+--------+------------+-----------------------------------------
5 rows in set (0.0005 sec)

mysql> CALL sys.ML_SCORE('mlcorpus.regression_2_table','target', @model, 'r2', @score, NULL);
Query OK, 0 rows affected (0.8688 sec)

mysql> SELECT @score;
+--------------------+
| @score             |
+--------------------+
| 0.9993192553520203 |
+--------------------+
1 row in set (0.0003 sec)

mysql> CALL sys.ML_EXPLAIN('mlcorpus.regression_2_table', 'target', @model,
                           JSON_OBJECT('model_explainer', 'partial_dependence',
                                       'columns_to_explain', JSON_ARRAY('f1'),
                                       'prediction_explainer', 'shap'));
Query OK, 0 rows affected (9.9860 sec)
```

```
m
mysql> CALL sys.ML_EXPLAIN_TABLE('mlcorpus.regression_2_predict', @model, 'mlcorpus.explanations',
                                  JSON_OBJECT('prediction_explainer', 'shap'));
Query OK, 0 rows affected (8.2625 sec)

mysql> SELECT * FROM mlcorpus.explanations;
+------------------+----+----+-----+------------+----------------+----------------+----------------+----
| _4aad19ca6e_pk_id | f1 | f2 | f3  | Prediction | f1_attribution | f2_attribution | f3_attribution | ml_
+------------------+----+----+-----+------------+----------------+----------------+----------------+----
|                1 | a  | 20 | 1.2 |     22.262 |       -10.7595 |       -4.25162 |       -2.48331 | {"p
|                2 | b  | 21 | 3.6 |    32.4861 |        2.33657 |       -8.50325 |        -1.1037 | {"p
|                3 | c  | 19 | 7.8 |    56.2482 |        14.8361 |              0 |        1.65554 | {"p
|                4 | d  | 18 |   9 |       31.8 |       -15.2433 |        4.25162 |        3.03516 | {"p
|                5 | e  | 17 | 3.6 |    55.9861 |        8.83008 |        8.50325 |        -1.1037 | {"p
+------------------+----+----+-----+------------+----------------+----------------+----------------+----
5 rows in set (0.0006 sec)
```

- An example with task set to `NULL`.

```
mysql> SET @model = 'tensorflow_recsys_onnx';

mysql> CALL sys.ML_MODEL_IMPORT(@onnx_encode_tensorflow_recsys, NULL, @model);
Query OK, 0 rows affected (1.0037 sec)

mysql> CALL sys.ML_MODEL_LOAD(@model, NULL);
Query OK, 0 rows affected (0.5116 sec)

mysql> CALL sys.ML_PREDICT_TABLE('mlcorpus.recsys_predict', @model, 'mlcorpus.predictions', NULL);
Query OK, 0 rows affected (0.8271 sec)

mysql> SELECT * FROM mlcorpus.predictions;
+------------------+---------+-------------+--------------------------+--------------------------------
| _4aad19ca6e_pk_id | user_id | movie_title | Prediction               | ml_results
+------------------+---------+-------------+--------------------------+--------------------------------
|                1 | a       | A           | {"output_1": ["0.7558"]} | {"predictions": {"prediction":
|                2 | b       | B           | {"output_1": ["1.0443"]} | {"predictions": {"prediction":
|                3 | c       | A           | {"output_1": ["0.8483"]} | {"predictions": {"prediction":
|                4 | d       | B           | {"output_1": ["1.2986"]} | {"predictions": {"prediction":
|                5 | e       | C           | {"output_1": ["1.1568"]} | {"predictions": {"prediction":
+------------------+---------+-------------+--------------------------+--------------------------------
5 rows in set (0.0005 sec)
```

What's Next

- Review how to Create a Machine Learning Model.
- Review Machine Learning Use Cases to create machine learning models with sample datasets.

6.9.8 Analyze Data Drift

MySQL HeatWave AutoML includes data drift detection for the following models:

- Classification
- Regression
- Anomaly detection (as of MySQL 9.3.2)

Before You Begin

- Review how to Create a Machine Learning Model.
- Review use cases for Classification Data and Regression Analysis.

Data Drift Detection Overview

Machine learning typically makes an assumption that the training data and test data are similar. Over time, the similarity between the training data and the test data can decrease. This is known as data drift.

You can monitor data drift in the model catalog and when running the `ML_PREDICT_ROW` and `ML_PREDICT_TABLE` routines.

For the model catalog, the `model_metadata` column includes the `training_drift_metric` JSON object literal, which contains `mean` and `variance` numeric values. See Model Metadata.

`mean` and `variance` indicate the quality of the trained drift detector, and both values should be low. The more important value is `mean`, and if it is greater than 1.0, then drift evaluation for the test results might not be reliable.

For the `ML_PREDICT_ROW` and `ML_PREDICT_TABLE` routines, the *`options`* parameter includes the `additional_details` boolean value. If this option is enabled, the `ml_results` column includes the `drift` JSON object literal, which contains the `metric` numeric value and the `attribution_percent` JSON object literal.

- `metric` indicates the similarity between training and test data. A low value indicates similar values. A value grater than 1.0 indicates data drift, and the prediction results are questionable.
- `attribution_percent` indicates the top three features that contribute to data drift for each result. The higher the percentage value, the greater the contribution.

Workflow to Analyze Data Drift

The workflow to analyze data drift includes the following:

1. Run `ML_TRAIN` to train the machine learning model with either the `classification` or `regression` task.
2. When training is complete, query the `model_metadata` column and review the `mean` and `variance` values.
3. Run the `ML_PREDICT_ROW` or `ML_PREDICT_TABLE` routines on the trained model with the `additional_details` option set to `true`.
4. Review the `drift` parameter in `ml_results`.

Analyze Data Drift in Model Metadata

To analyze data drift in model metadata:

1. Train the model with `ML_TRAIN`.

```
mysql> CALL sys.ML_TRAIN('table_name', 'target_column_name', JSON_OBJECT('task', 'task_name'), @vari
```

Replace *`table_name`*, *`target_column_name`*, *`task_name`*, and *`variable`* with your own values. For example:

```
mysql> CALL sys.ML_TRAIN('census_data.census_train', 'revenue', JSON_OBJECT('task', 'classification'
```

Where:

- `census_data.census_train` is the fully qualified name of the table that contains the training dataset (*`schema_name.table_name`*).
- `revenue` is the name of the target column, which contains ground truth values.
- `JSON_OBJECT('task', 'classification')` specifies the machine learning task type.
- `@census_model` is the name of the user-defined session variable that stores the model handle for the duration of the connection. User variables are written as *`@var_name`*. Any valid name for a user-defined variable is permitted. For example, `@my_model`. Learn more about Model Handles.

2. Query the `model_metadata` column from the model catalog. Optionally, use `JSON_PRETTY` to view the output in an easily readable format.

```
mysql> SELECT JSON_PRETTY(model_metadata) FROM ML_SCHEMA_user1.MODEL_CATALOG WHERE model_handle=model_h
```

Replace *`user1`* with your own user name and *`model_handle`* with your own model handle. For example:

```
mysql> SELECT JSON_PRETTY(model_metadata) FROM ML_SCHEMA_user1.MODEL_CATALOG WHERE model_handle=@census
+-----------------------------------------------------------+
| JSON_PRETTY(model_metadata)                               |
+-----------------------------------------------------------+
| {
  "task": "classification",
  "notes": null,
  "chunks": 1,
  "format": "HWMLv2.0",
  "n_rows": 100,
  "status": "Ready",
  "options": {
    "task": "classification",
    "model_explainer": "permutation_importance",
    "prediction_explainer": "permutation_importance"
  },
  "n_columns": 14,
  "column_names": [
    "age",
    "workclass",
    "fnlwgt",
    "education",
    "education-num",
    "marital-status",
    "occupation",
    "relationship",
    "race",
    "sex",
    "capital-gain",
    "capital-loss",
    "hours-per-week",
    "native-country"
  ],
  "contamination": null,
  "model_quality": "high",
  "training_time": 73.90254211425781,
  "algorithm_name": "RandomForestClassifier",
  "training_score": -0.35963335633277893,
  "build_timestamp": 1744377124,
  "n_selected_rows": 80,
  "training_params": {
    "recommend": "ratings",
    "force_use_X": false,
    "recommend_k": 3,
    "remove_seen": true,
    "ranking_topk": 10,
    "lsa_components": 100,
    "ranking_threshold": 1,
    "feedback_threshold": 1
  },
  "train_table_name": "census_data.census_train",
  "model_explanation": {
    "permutation_importance": {
      "age": -0.0057,
      "sex": 0.0002,
      "race": 0.0001,
      "fnlwgt": 0.0103,
      "education": 0.0108,
      "workclass": 0.0189,
      "occupation": 0.0,
      "capital-gain": 0.0304,
      "capital-loss": 0.0,
      "relationship": 0.0195,
```

```
      "education-num": 0.0152,
      "hours-per-week": 0.0235,
      "marital-status": 0.0099,
      "native-country": 0.0
    }
  },
  "n_selected_columns": 11,
  "target_column_name": "revenue",
  "optimization_metric": "neg_log_loss",
  "selected_column_names": [
    "age",
    "capital-gain",
    "education",
    "education-num",
    "fnlwgt",
    "hours-per-week",
    "marital-status",
    "race",
    "relationship",
    "sex",
    "workclass"
  ],
  "training_drift_metric": {
    "mean": 0.3535,
    "variance": 0.0597
  }
} |
+-----------------------------------------------------------+
1 row in set (0.0009 sec)
```

Where:

- `JSON_PRETTY` displays the information in an easily readable format.
- `ML_SCHEMA_user1.MODEL_CATALOG` refers to the model catalog name. Replace `user1` with your own user name.
- `model_handle` refers to the session variable for the trained model, `@census_model`. Learn more about Model Handles.

For `training_drift_metric`, the output generates a `mean` value of 0.3535 and a `variance` value of 0.0597, which indicates acceptable data drift.

Analyze Data Drift Detection with ML_PREDICT_TABLE

To analyze data drift detection with a table of predictions:

1. If not done already, train the model to use. See Analyze Data Drift in Model Metadata.
2. Load the trained model. Update `@census_model` with your own session variable for the trained model.

```
mysql> CALL sys.ML_MODEL_LOAD(@census_model, NULL);
```

3. Run `ML_PREDICT_TABLE` to generate a table of predictions.

```
mysql> CALL sys.ML_PREDICT_TABLE(table_name, model_handle, output_table_name), [options]);
```

Replace *`table_name`*, *`model_handle`*, *`output_table_name`*),and *`options`* with your own values. For example:

```
mysql> CALL sys.ML_PREDICT_TABLE('census_data.`census_test`', @census_model, 'census_data.`census_te
                                 JSON_OBJECT('additional_details', true));
```

Where:

- `census_data.census_test` is the fully qualified name of the test dataset table (*`database_name.table_name`*).

- `@census_model` is the session variable that contains the model handle. See Work with Model Handles.
- `census_data.census_test_predictions` is the output table where predictions are stored.
- `JSON_OBJECT` includes the `additional_details` option set to `true`, so `ml_results` includes values for `metric` and `attribution_percent`.

4. Since a `metric` value over 1.0 indicates data drift, query rows in the output table that only have a metric value over 1.0.

```
mysql> SELECT ml_results FROM table_name WHERE JSON_EXTRACT(ml_results, '$.drift.metric') > 1.0;
```

Replace *`table_name`* with your own value. For example:

```
mysql> SELECT ml_results FROM census_test_predictions WHERE JSON_EXTRACT(ml_results, '$.drift.metric') >
+-----------------------------------------------------------------------------------------------------------
| ml_results
+-----------------------------------------------------------------------------------------------------------
| {"predictions": {"revenue": "<=50K"}, "probabilities": {"<=50K": 0.67, ">50K": 0.33}, "drift": {"metr
| {"predictions": {"revenue": "<=50K"}, "probabilities": {"<=50K": 0.9, ">50K": 0.1}, "drift": {"metric
| {"predictions": {"revenue": "<=50K"}, "probabilities": {"<=50K": 0.99, ">50K": 0.01}, "drift": {"metr
| {"predictions": {"revenue": "<=50K"}, "probabilities": {"<=50K": 0.78, ">50K": 0.22}, "drift": {"metr
| {"predictions": {"revenue": "<=50K"}, "probabilities": {"<=50K": 0.97, ">50K": 0.03}, "drift": {"metr
| {"predictions": {"revenue": ">50K"}, "probabilities": {"<=50K": 0.32, ">50K": 0.68}, "drift": {"metri
| {"predictions": {"revenue": "<=50K"}, "probabilities": {"<=50K": 0.96, ">50K": 0.04}, "drift": {"metr
| {"predictions": {"revenue": "<=50K"}, "probabilities": {"<=50K": 0.89, ">50K": 0.11}, "drift": {"metr
| {"predictions": {"revenue": "<=50K"}, "probabilities": {"<=50K": 0.91, ">50K": 0.09}, "drift": {"metr
| {"predictions": {"revenue": "<=50K"}, "probabilities": {"<=50K": 0.78, ">50K": 0.22}, "drift": {"metr
| {"predictions": {"revenue": "<=50K"}, "probabilities": {"<=50K": 0.89, ">50K": 0.11}, "drift": {"metr
| {"predictions": {"revenue": "<=50K"}, "probabilities": {"<=50K": 0.62, ">50K": 0.38}, "drift": {"metr
+-----------------------------------------------------------------------------------------------------------
12 rows in set (0.0014 sec)
```

The output displays the rows with high metric values (> 1.0), indicating data drift.

Analyze Data Drift Detection with ML_PREDICT_ROW

To anayze data drift detection with one or more rows of predictions:

1. If not done already, train the model to use. See Analyze Data Drift in Model Metadata.
2. Load the trained model. Update `@census_model` with your own session variable for the trained model.

```
mysql> CALL sys.ML_MODEL_LOAD(@census_model, NULL);
```

3. Run `ML_PREDICT_ROW` to generate predictions for a defined number of rows.

```
mysql> SELECT sys.ML_PREDICT_ROW(JSON_OBJECT("output_col_name", schema.`input_col_name`,
                                             "output_col_name", schema.`input_col_name`, ...),
                                             model_handle, options) FROM input_table_name LIMIT N;
```

The following example generates predictions for three rows of the table. The output is similar to the previous example.

```
mysql> SELECT sys.ML_PREDICT_ROW(JSON_OBJECT(
 "age", census_test.`age`,
 "workclass", census_test.`workclass`,
 "fnlwgt", census_test.`fnlwgt`,
 "education", census_test.`education`,
 "education-num", census_test.`education-num`,
 "marital-status", census_test.`marital-status`,
 "occupation", census_test.`occupation`,
 "relationship", census_test.`relationship`,
 "race", census_test.`race`,
 "sex", census_test.`sex`,
```

```
 "capital-gain", census_test.`capital-gain`,
 "capital-loss", census_test.`capital-loss`,
 "hours-per-week", census_test.`hours-per-week`,
 "native-country", census_test.`native-country`),
 @census_model, JSON_OBJECT('additional_details', TRUE))FROM census_data.census_test LIMIT 3;
+-------------------------------------------------+
| sys.ML_PREDICT_ROW(JSON_OBJECT(
"age", census_test.`age`,
"workclass", census_test.`workclass`,
"fnlwgt", census_test.`fnlwgt`,
"education", census_test.`education`,
"education-num", census_test.`education-num`,
"ma                                             |
+-------------------------------------------------+
|{                                                |
|    "age": 37,                                   |
|    "sex": "Male",                               |
|    "race": "White",                             |
|    "fnlwgt": 99146,                             |
|    "education": "Bachelors",                    |
|    "workclass": "Private",                      |
|    "Prediction": "<=50K",                       |
|    "ml_results": {                              |
|        "drift": {                               |
|            "metric": 0,                         |
|            "attribution_percent": {             |
|                "age": 0,                        |
|                "fnlwgt": 46.67,                 |
|                "capital-gain": 0}},             |
|        "predictions": {                         |
|            "revenue": "<=50K"},                 |
|        "probabilities": {                       |
|            ">50K": 0.42,                        |
|            "<=50K": 0.58}},                     |
|    "occupation": "Exec-managerial",             |
|    "capital-gain": 0,                           |
|    "capital-loss": 1977,                        |
|    "relationship": "Husband",                   |
|    "education-num": 13,                         |
|    "hours-per-week": 50,                        |
|    "marital-status": "Married-civ-spouse",      |
|    "native-country": "United-States"}           |
|{                                                |
|    "age": 34,                                   |
|    "sex": "Male",                               |
|    "race": "White",                             |
|    "fnlwgt": 27409,                             |
|    "education": "9th",                          |
|    "workclass": "Private",                      |
|    "Prediction": "<=50K",                       |
|    "ml_results": {                              |
|        "drift": {                               |
|            "metric": 0.1,                       |
|            "attribution_percent": {             |
|                "fnlwgt": 25,                    |
|                "education": 33.31,              |
|                "workclass": 16.22}},            |
|        "predictions": {                         |
|            "revenue": "<=50K"},                 |
|        "probabilities": {                       |
|            ">50K": 0.24,                        |
|            "<=50K": 0.76}},                     |
|    "occupation": "Craft-repair",                |
|    "capital-gain": 0,                           |
|    "capital-loss": 0,                           |
|    "relationship": "Husband",                   |
|    "education-num": 5,                          |
|    "hours-per-week": 50,                        |
|    "marital-status": "Married-civ-spouse",      |
|    "native-country": "United-States"}           |
|{                                                |
|    "age": 30,                                   |
```

```
|    "sex": "Female",                                |
|    "race": "White",                                |
|    "fnlwgt": 299507,                               |
|    "education": "Assoc-acdm",                      |
|    "workclass": "Private",                         |
|    "Prediction": "<=50K",                          |
|    "ml_results": {                                 |
|        "drift": {                                  |
|            "metric": 0.26,                         |
|            "attribution_percent": {                |
|                "relationship": 21.36,              |
|                "education-num": 28.33,             |
|                "hours-per-week": 33.21}},          |
|        "predictions": {                            |
|            "revenue": "<=50K"},                    |
|        "probabilities": {                          |
|            ">50K": 0.01,                           |
|            "<=50K": 0.99}},                        |
|    "occupation": "Other-service",                  |
|    "capital-gain": 0,                              |
|    "capital-loss": 0,                              |
|    "relationship": "Unmarried",                    |
|    "education-num": 12,                            |
|    "hours-per-week": 40,                           |
|    "marital-status": "Separated",                  |
|    "native-country": "United-States"}              |
+----------------------------------------------------+
10 rows in set (6.8109 sec)
```

Where:

- The first `JSON_OBJECT` has output column names and key-value pairs of the columns in the trained table.
- `@census_model` is the session variable that contains the model handle. Learn more about Model Handles.
- The second `JSON_OBJECT` includes the `additional_details` option set to `true`, so `ml_results` includes values for `metric` and `attribution_percent`.
- `census_data.census_test` is the fully qualified name of the test dataset table (*`database_name.table_name`*).
- The `LIMIT` of 3 means that the output includes a maximum of three rows from the trained table.

The output allows you to review data drift values for the selected rows.

What's Next

- Review Machine Learning Use Cases to create machine learning models with sample datasets.

6.10 Track Progress for MySQL HeatWave AutoML Routines

You can monitor the progress of MySQL HeatWave AutoML routines through Performance Schema queries made from a second MySQL Client session.

Before You Begin

- Review the following:
  - Create a Machine Learning Model
  - Manage Machine Learning Models

Progress Tracking Overview

Progress tracking supports all MySQL HeatWave AutoML Routines. For each routine, progress tracking tracks each individual operation and stage within routines. Completed percentage values are also included.

Note

`ML_MODEL_IMPORT` only supports progress tracking for models in ONNX format. It does not support progress tracking for models in MySQL HeatWave AutoML format. See Manage External ONNX Models to learn more.

Track Progress for ML_TRAIN

This example displays the stages for `ML_TRAIN` only. Other MySQL HeatWave AutoML routines may have different stages to review.

To track progress while training a machine learning model:

1. Connect to your DB system from two different MySQL Client sessions.
2. From the first MySQL Client session, train a machine learning model. See Train a Model. The following example trains a classification machine learning model.

```
mysql> CALL sys.ML_TRAIN('census_data.census_train', 'revenue', JSON_OBJECT('task', 'classification'
```

3. While `ML_TRAIN` runs, use the second MySQL Client session and query the performance_schema.rpd_query_stats table, which stores data and statistics for for the last 100 completed queries. You can limit the output to the two most recent entries sorted by `QUERY_ID` in descending order. Optionally, use `\G` to display the output in an easily readable format. The output is similar to the following:

```
mysql> SELECT * FROM performance_schema.rpd_query_stats ORDER BY QUERY_ID DESC limit 2\G
*************************** 131. row ***************************
                     QUERY_ID: 140
                 STATEMENT_ID: 7881043
               CONNECTION_ID: 41436
                  QUERY_TEXT: ML_TRAIN
                  QEXEC_TEXT: {
                                 "status": "In Progress",
                                 "completedSteps": [
                                    "Initialization",
                                    "Preprocessing"
                              ],
                              "completionPercentage": 22,
                              "totalRunTime": null,
                              "startTime": "04/22/2025, 15:09:23:287333 UTC",
                              "progressItems": [
                                 {
                                     "type": "Initialization",
                                     "status": "Completed",
                                     "lastUpdated": "04/22/2025, 15:09:28:235097 UTC",
                                     "completionPercentage": 100
                                 },
                                 {
                                     "type": "Preprocessing",
                                     "status": "Completed",
                                     "lastUpdated": "04/22/2025, 15:09:29:060563 UTC",
                                     "completionPercentage": 100
                                 },
                                 {
                                     "type": "Algorithm Selection",
                                     "status": "In Progress",
                                     "lastUpdated": "04/22/2025, 15:09:38:649341 UTC",
                                     "completionPercentage": 98
                                 },
                                 {
                                     "type": "Row Sampling",
```

```
                                        "status": "Not Started",
                                        "lastUpdated": "04/22/2025, 15:09:28:227493 UTC",
                                        "completionPercentage": 0
                                    },
                                    {
                                        "type": "Column Selection",
                                        "status": "Not Started",
                                        "lastUpdated": "04/22/2025, 15:09:28:227911 UTC",
                                        "completionPercentage": 0
                                    },
                                    {
                                        "type": "Hyperparameter Optimization",
                                        "status": "Not Started",
                                        "lastUpdated": "04/22/2025, 15:09:28:228325 UTC",
                                        "completionPercentage": 0
                                    },
                                    {
                                        "type": "Model Generation",
                                        "status": "Not Started",
                                        "lastUpdated": "04/22/2025, 15:09:28:228760 UTC",
                                        "completionPercentage": 0
                                    },
                                    {
                                        "type": "Prediction Explainer",
                                        "status": "Not Started",
                                        "lastUpdated": "04/22/2025, 15:09:28:229296 UTC",
                                        "completionPercentage": 0
                                    },
                                    {
                                        "type": "Model Explainer",
                                        "status": "Not Started",
                                        "lastUpdated": "04/22/2025, 15:09:28:229704 UTC",
                                        "completionPercentage": 0
                                    }
                                ],
                                "details": {
                                    "trainingScore": null,
                                    "optimizationMetric": null,
                                    "options": {
                                        "task": "classification"
                                    },
                                    "operation": "training",
                                    "user_name": "admin",
                                    "table_name": "census_train",
                                    "schema_name": "census_data",
                                    "model_handle": "census_data.census_train_admin_1745334557047",
                                    "target_column_name": "revenue"
                                }
                            }
                    QKRN_TEXT: {}
                     QEP_TEXT: {}
    INTERNAL_OFFLOAD_FAIL_TEXT:
    EXTERNAL_OFFLOAD_FAIL_TEXT:
OFFLOAD_FAIL_QUERY_SUBSTR_START: 0
  OFFLOAD_FAIL_QUERY_SUBSTR_LEN: 0
            EXPORT_FILE_PREFIX: NULL
*************************** 130. row ***************************
                    QUERY_ID: 139
                STATEMENT_ID: 7881035
                CONNECTION_ID: 41436
                   QUERY_TEXT: ML_LOAD_TABLE
                   QEXEC_TEXT: {
                                   "status": "Completed",
                                   "completedSteps": [],
                                   "completionPercentage": 100,
                                   "totalRunTime": "3.77 Seconds",
                                   "startTime": "04/22/2025, 15:09:18:046900 UTC",
                                   "progressItems": [],
                                   "details": {}
                               }
                    QKRN_TEXT: {}
                     QEP_TEXT: {}
```

```
     INTERNAL_OFFLOAD_FAIL_TEXT:
     EXTERNAL_OFFLOAD_FAIL_TEXT:
OFFLOAD_FAIL_QUERY_SUBSTR_START: 0
  OFFLOAD_FAIL_QUERY_SUBSTR_LEN: 0
             EXPORT_FILE_PREFIX: NULL
131 rows in set (0.0753 sec)
```

Refer to the second entry to confirm that the table was successfully loaded. It shows that the `ML_LOAD_TABLE` operation has a `status` of `Completed`, and a `completionPercentage` of `100`. Refer to the first entry to review the status and completion percentage for each stage of `ML_TRAIN`. It shows that the `ML_TRAIN` operation has a `status` of `In Progress`, with sub-tasks displaying values for `completionPercentage`.

4. When the status shows `Completed` for all `ML_TRAIN` tasks, run the same query again from either MySQL Client session for `performance_schema.rpd_query_stats`. Limit the query to display the most recent `QUERY_ID`, which for this example is `143`. The output is similar to the following:

```
mysql> SELECT * FROM performance_schema.rpd_query_stats WHERE QUERY_ID=143\G
*************************** 134. row ***************************
                    QUERY_ID: 143
                STATEMENT_ID: 7884317
               CONNECTION_ID: 41436
                  QUERY_TEXT: ML_UNLOAD_MODEL
                  QEXEC_TEXT: {
    "status": "Completed",
    "completedSteps": [],
    "completionPercentage": 100,
    "totalRunTime": "0.64 Seconds",
    "startTime": "04/22/2025, 15:13:37:392368 UTC",
    "progressItems": [],
    "details": {
        "operation": "unload_model",
        "user_name": "admin",
        "model_handle": "census_data.census_train_admin_1745334557047"
    }
}
                   QKRN_TEXT: {}
                    QEP_TEXT: {}
     INTERNAL_OFFLOAD_FAIL_TEXT:
     EXTERNAL_OFFLOAD_FAIL_TEXT:
OFFLOAD_FAIL_QUERY_SUBSTR_START: 0
  OFFLOAD_FAIL_QUERY_SUBSTR_LEN: 0
             EXPORT_FILE_PREFIX: NULL
1 row in set (0.0729 sec)
```

Confirm that the `status` shows `Completed` and the `completionPercentage` shows `100`.

Track Status and Completion Percentage

You have the option to query only the status or completed percentage of a routine. This example loads and generates a table of predictions for the model previously trained.

1. Connect to your DB system from two different MySQL Client sessions.
2. From the first MySQL Client session, train a machine learning model. See Train a Model. The following example trains a classification machine learning model.

```
mysql> CALL sys.ML_TRAIN('census_data.census_train', 'revenue', JSON_OBJECT('task', 'classification'
```

3. Load the trained model. Replace *`@census_model`* with your own model handle. See Load a Model.

```
mysql> CALL sys.ML_MODEL_LOAD(@census_model, NULL);
```

4. Use `ML_PREDICT_TABLE` to generate a table of predictions. See Generate Predictions for a Table. For example:

```
mysql> CALL sys.ML_PREDICT_TABLE('census_data.census_train', @census_model, 'census_data.census_trai
```

5. While the routine runs, use the second MySQL Client session and create a session variable for `QEXEC_TEXT`. Query the `QUERY_TEXT` column for `ML_PREDICT_TABLE`, order the table by `QUERY_ID` in descending order, and only query the most recent entry.

```
mysql> SELECT QEXEC_TEXT INTO @progress_json
       FROM performance_schema.rpd_query_stats
       WHERE QUERY_TEXT='ML_PREDICT_TABLE'
       ORDER BY QUERY_ID
       DESC limit 1;
```

6. From the second MySQL Client session, query the completion percentage for the routine.

```
mysql> SELECT JSON_EXTRACT(@progress_json, '$.completionPercentage');
+-------------------------------------------------------+
| JSON_EXTRACT(@progress_json, '$.completionPercentage') |
+-------------------------------------------------------+
| 67                                                    |
+-------------------------------------------------------+
1 row in set (0.0463 sec)
```

7. Query the current status for the routine.

```
mysql> SELECT JSON_EXTRACT(@progress_json, '$.status');
+----------------------------------------+
| JSON_EXTRACT(@progress_json, '$.status') |
+----------------------------------------+
| "In Progress"                          |
+----------------------------------------+
1 row in set (0.0439 sec)
```

What's Next

- Review Machine Learning Use Cases to create machine learning models with sample datasets.
- Score a Model

6.11 Monitor the Status of MySQL HeatWave AutoML

You can monitor the status of MySQL HeatWave AutoML by querying the `rapid_ml_status` variable or by querying the `ML_STATUS` column of the `performance_schema.rpd_nodes` table.

Before You Begin

- Review how to Track Progress for MySQL HeatWave AutoML Routines.

Query the rapid_ml_status Variable

The `rapid_ml_status` variable provides the status of MySQL HeatWave AutoML. Possible values are `ON` and `OFF`.

- `ON`: MySQL HeatWave AutoML is up and running.
- `OFF`: MySQL HeatWave AutoML is down.

The following example queries the `rapid_ml_status` status variable directly.

```
mysql> SHOW GLOBAL STATUS LIKE 'rapid_ml_status';
+-----------------+-------+
| Variable_name   | Value |
+-----------------+-------+
| rapid_ml_status | ON    |
+-----------------+-------+
```

The following example queries the the `rapid_ml_status` status through the `performance_schema.global_status` table.

```
mysql> SELECT VARIABLE_NAME, VARIABLE_VALUE
       FROM performance_schema.global_status
       WHERE VARIABLE_NAME LIKE 'rapid_ml_status';
+-----------------+----------------+
| VARIABLE_NAME   | VARIABLE_VALUE |
+-----------------+----------------+
| rapid_ml_status | ON             |
+-----------------+----------------+
```

Query the ML_STATUS Column

The MySQL HeatWave plugin writes MySQL HeatWave AutoML status information to the `ML_STATUS` column of the `performance_schema.rpd_nodes` table after each AutoML query. Possible values include:

- `UNAVAIL_MLSTATE`: MySQL HeatWave AutoML is not available.
- `AVAIL_MLSTATE`: MySQL HeatWave AutoML is available.
- `DOWN_MLSTATE`: MySQL HeatWave AutoML is down.

`ML_STATUS` is reported for each MySQL HeatWave node.

You can query the `ML_STATUS` column of the `performance_schema.rpd_nodes` table.

To following example retrieves `ID`, `STATUS`, and `ML_STATUS` for each MySQL HeatWave node from the `performance_schema.rpd_nodes` table:

```
mysql> SELECT ID, STATUS, ML_STATUS FROM performance_schema.rpd_nodes;
+----+---------------+---------------+
| ID | STATUS        | ML_STATUS     |
+----+---------------+---------------+
|  1 | AVAIL_RNSTATE | AVAIL_MLSTATE |
|  0 | AVAIL_RNSTATE | AVAIL_MLSTATE |
+----+---------------+---------------+
```

Resolve a Down Status for MySQL HeatWave AutoML

If `rapid_ml_status` is `OFF` or `ML_STATUS` reports `DOWN_MLSTATE` for any MySQL HeatWave node, you can restart the MySQL HeatWave Cluster in the MySQL HeatWave Console. Be aware that restarting interrupts any analytics queries that are running.

See the following to learn more:

- Managing a MySQL HeatWave Cluster in the *MySQL HeatWave on OCI Service Guide*.
- Starting,Stopping, or Restarting a MySQL HeatWave Cluter in the *MySQL HeatWave on AWS Service Guide*.
- Restarting a Database in the *MySQL HeatWave for Azure Service Guide*.

What's Next

- Review Machine Learning Use Cases to create machine learning models with sample datasets.

Chapter 7 Perform AI-Powered Search and Content Generation

Table of Contents

7.1 About MySQL HeatWave GenAI 453
7.2 Additional MySQL HeatWave GenAI Requirements 454
7.3 MySQL HeatWave GenAI Roles and Privileges 455
7.4 Supported Models and Languages 456
7.5 Authenticate OCI Generative AI Service 461
7.6 Generate Text-Based Content 462
7.6.1 Generate New Content 462
7.6.2 Summarize Content 465
7.6.3 Summarize Unstructured Files 469
7.7 Set Up a Vector Store 471
7.7.1 About MySQL HeatWave Vector Store 471
7.7.2 Ingest Files into a Vector Store 473
7.7.3 Update a Vector Store 481
7.8 Generate Vector Embeddings 483
7.9 Perform Vector Search with RAG 486
7.9.1 Run Retrieval-Augmented Generation 486
7.9.2 Use Your Own Embeddings 494
7.9.3 Automatic Vector Index Creation 506
7.10 Start a Conversational Chat 506
7.10.1 Run MySQL HeatWave Chat 506
7.10.2 View Chat Session Details 510
7.11 Generate SQL Queries From Natural-Language Statements 511

This chapter describes MySQL HeatWave GenAI.

7.1 About MySQL HeatWave GenAI

MySQL HeatWave GenAI is a feature of MySQL HeatWave that lets you communicate with unstructured data in MySQL HeatWave using natural-language queries. It uses a familiar SQL interface which makes it is easy to use for content generation, summarization, and retrieval-augmented generation (RAG).

Using MySQL HeatWave GenAI, you can perform natural-language searches in a single step using either in-database or external large language models (LLMs). All the elements that are necessary to use MySQL HeatWave GenAI with proprietary data are integrated and optimized to work with each other.

Note

This chapter assumes that you are familiar with MySQL HeatWave DB Systems.

MySQL HeatWave GenAI includes the following:

- **In-Database and OCI Generative AI Service LLMs**

  MySQL HeatWave GenAI uses large language models (LLMs) to enable natural language communication in multiple languages. You can use the capabilities of the LLMs to search data as well as generate or summarize content. However, as these LLMs are trained on public data, the responses to your queries are generated based on information available in the public data sources. To produce more relevant results, you can use the LLM capabilities of MySQL HeatWave GenAI with the vector store functionality to perform a vector search with RAG.

- **In-Database Vector Store**

  MySQL HeatWave GenAI provides an inbuilt vector store that you can use to store enterprise-specific proprietary content and perform vector-based similarity search across documents. Queries that you ask are automatically encoded with the same embedding model as the vector store without requiring any additional inputs or running a separate service. The vector store also provides valuable context for LLMs for RAG use cases.

- **Retrieval-Augmented Generation**

  MySQL HeatWave GenAI retrieves content from the vector store and provides it as context to the LLM along with the query. This process of generating an augmented prompt is called retrieval-augmented generation (RAG), and it helps MySQL HeatWave GenAI produce more contextually relevant, personalised, and accurate results.

- **MySQL HeatWave Chat**

  This is an inbuilt chatbot that extends the LLMs capabilities as well as vector store and RAG functionalities of MySQL HeatWave GenAI to let you ask multiple follow-up questions about a topic in a single session. You can use MySQL HeatWave Chat to build customized chat applications by specifying custom settings, prompt, chat history length, and number of citations to be used for generating a response.

  MySQL HeatWave Chat also provides a graphical interface integrated with MySQL Shell for VS Code. See Set Up MySQL Shell for VS Code.

Benefits

MySQL HeatWave GenAI lets you integrate generative AI into the applications, providing an integrated end-to-end pipeline including vector store generation, vector search with RAG, and an inbuilt chatbot.

Some key benefits of using MySQL HeatWave GenAI are described below:

- The natural-language processing (NLP) capabilities of the LLMs let non-technical users have human-like conversations with the system in natural language.
- The in-database integration of LLMs and embedding generation eliminates the need for using external solutions, and ensures the security of the proprietary content.
- The in-database integration of LLMs, vector store, and embedding generation simplifies complexity of applications that use these features.
- The cost of running natural-language queries is significantly low as MySQL HeatWave GenAI is available at no additional cost for MySQL HeatWave users.
- MySQL HeatWave GenAI integrates with other in-database capabilities such as machine learning, analytics, and Lakehouse.

What's Next

- Review the Supported Languages, Embedding Models, and LLMs.
- To learn more about MySQL HeatWave GenAI, see the MySQL HeatWave GenAI: Technical Overview.

7.2 Additional MySQL HeatWave GenAI Requirements

To use MySQL HeatWave GenAI, you need the following:

- A MySQL HeatWave DB system that meets the following requirements:
  - The MySQL HeatWave DB system must be version 9.0.0 - Innovation or higher.

For more information, see Creating a DB system and Editing a DB system in the *MySQL HeatWave on OCI Service Guide*, or Creating a DB system and Editing a DB system in the *MySQL HeatWave on AWS Service Guide*.

- The recommended shape of the MySQL HeatWave Cluster that is added to the DB System is `HeatWave.512GB`.

  Note

  MySQL HeatWave GenAI is supported on DB Systems with 32 or less MySQL HeatWave Cluster nodes only.

  For more information, see Adding a MySQL HeatWave Cluster and Editing a MySQL HeatWave Cluster in the *MySQL HeatWave on OCI Service Guide*, or Creating a MySQL HeatWave Cluster and Editing a MySQL HeatWave Cluster in the *MySQL HeatWave on AWS Service Guide*.

- MySQL HeatWave Lakehouse must be enabled on the DB system.

  For more information, see Managing a MySQL HeatWave Cluster in the *MySQL HeatWave on OCI Service Guide*, or Creating a MySQL HeatWave Cluster in the *MySQL HeatWave on AWS Service Guide*.

- To create a vector store, you need an Object Storage bucket for storing files that you want the vector store to ingest.

  Vector store can ingest files in the following formats: PPT, TXT, HTML, DOC, and PDF. Each file can be up to 100 MB in size.

  If you want to use the uniform resource identifier for accessing files in the Object Storage bucket that you want to ingest into the vector store, you need to enable the DB System to access the Oracle Cloud Infrastructure (OCI) Object Storage bucket. For more information, see Resource Principals.

- If you want to use the OCI Generative AI Service with MySQL HeatWave GenAI, you need to enable the DB System to access OCI services. For more information, see Section 7.5, “Authenticate OCI Generative AI Service”.

What's Next

Learn about the required MySQL HeatWave GenAI roles and privileges.

7.3 MySQL HeatWave GenAI Roles and Privileges

To perform the following MySQL HeatWave GenAI functions, ask the admin user to grant you the required roles and privileges:

- To create a vector store using asynchronous load, the following role and privileges are required:
  - The `mysql_task_user` role:

    ```
    mysql> GRANT 'mysql_task_user' TO 'user_name'@'%';
    ```

  - The `VECTOR_STORE_LOAD_EXEC` privilege:

    ```
    mysql> GRANT VECTOR_STORE_LOAD_EXEC ON *.* TO 'user_name'@'%';
    ```

  - The `SELECT` privilege on the following `performance_schema` and `sys` schema tables:

    ```
    mysql> GRANT SELECT ON 'performance_schema'.'rpd_nodes' TO 'user_name'@'%';
    mysql> GRANT SELECT ON 'performance_schema'.'rpd_table_id' TO 'user_name'@'%';
    mysql> GRANT SELECT ON 'performance_schema'.'rpd_tables' TO 'user_name'@'%';
    mysql> GRANT SELECT ON 'sys'.'vector_store_load_metadata' TO 'user_name'@'%';
    mysql> GRANT SELECT ON 'sys'.'vector_store_load_tables' TO 'user_name'@'%';
    ```

- The EXECUTE privilege on the following stored procedures in the sys schema:

```
mysql> GRANT EXECUTE ON PROCEDURE 'sys'.'vector_store_load_current_schema' TO 'user_name'@'%';
mysql> GRANT EXECUTE ON PROCEDURE 'sys'.'vector_store_load' TO 'user_name'@'%';
```

- The CREATE, ALTER, and EVENT privileges on the schema where you want to set up the vector store table:

```
mysql> GRANT CREATE, ALTER, EVENT ON 'schema_name'.* TO 'user_name'@'%';
```

- To run the batch queries using ML_GENERATE_TABLE, ML_RAG_TABLE, and ML_EMBED_TABLE, the following privileges are required:
  - SELECT and ALTER privileges on the input table:

```
mysql> GRANT SELECT, ALTER ON input_schema.input_table TO 'user_name'@'%';
```

  - SELECT, INSERT, CREATE, DROP, ALTER, UPDATE privileges on the schema where the output table is created.

```
mysql> GRANT SELECT, INSERT, CREATE, DROP, ALTER, UPDATE ON output_schema.* TO 'user_name'@'%';
```

For more information, see Privileges Provided by MySQL and Default MySQL Privileges.

What's Next

Learn about the available LLMs, embedding Models, and languages.

7.4 Supported Models and Languages

This topic provides the list of languages that MySQL HeatWave GenAI supports and the embedding models as well as large language models (LLMs) that are available in MySQL HeatWave.

This topic contains the following sections:

- Before You Begin
- View Available Models
- MySQL HeatWave In-Database LLMs
- OCI Generative AI Service LLMs
- MySQL HeatWave In-Database Embedding Models
- OCI Generative AI Service Embedding Models
- Languages
- What's Next

Before You Begin

- Review the MySQL HeatWave GenAI requirements and privileges.
- Connect to your MySQL HeatWave DB System.

View Available Models

As of MySQL 9.3.2, you can view the list of available models as shown below:

```
mysql> SELECT * FROM sys.ML_SUPPORTED_LLMS;
```

The output is similar to the following:

```
+--------------------------+--------------------------------------------+-------------------+--------
| provider                 | model_id                                   | availability_date | capabi
+--------------------------+--------------------------------------------+-------------------+--------
| HeatWave                 | llama3.1-8b-instruct-v1                    | 2025-05-20        | ["GENE
| HeatWave                 | llama3.2-1b-instruct-v1                    | 2025-05-20        | ["GENE
| HeatWave                 | llama3.2-3b-instruct-v1                    | 2025-05-20        | ["GENE
| HeatWave                 | mistral-7b-instruct-v3                     | 2025-05-20        | ["GENE
| HeatWave                 | all_minilm_l12_v2                          | 2024-07-01        | ["TEXT
| HeatWave                 | multilingual-e5-small                      | 2024-07-24        | ["TEXT
| OCI Generative AI Service | xai.grok-4-fast-non-reasoning             | 2025-09-25        | ["GENE
| OCI Generative AI Service | xai.grok-4-fast-reasoning                 | 2025-09-25        | ["GENE
| OCI Generative AI Service | xai.grok-code-fast-1                      | 2025-09-11        | ["GENE
| OCI Generative AI Service | openai.gpt-oss-120b                       | 2025-08-28        | ["GENE
| OCI Generative AI Service | openai.gpt-oss-20b                        | 2025-08-28        | ["GENE
| OCI Generative AI Service | google.gemini-2.5-flash                   | 2025-08-28        | ["GENE
| OCI Generative AI Service | google.gemini-2.5-pro                     | 2025-08-28        | ["GENE
| OCI Generative AI Service | google.gemini-2.5-flash-lite              | 2025-08-28        | ["GENE
| OCI Generative AI Service | xai.grok-4                                | 2025-07-22        | ["GENE
| OCI Generative AI Service | cohere.command-latest                     | 2025-07-02        | ["GENE
| OCI Generative AI Service | cohere.command-plus-latest                | 2025-07-02        | ["GENE
| OCI Generative AI Service | cohere.embed-v4.0                         | 2025-07-01        | ["TEXT
| OCI Generative AI Service | xai.grok-3-fast                           | 2025-05-21        | ["GENE
| OCI Generative AI Service | xai.grok-3                                | 2025-05-21        | ["GENE
| OCI Generative AI Service | xai.grok-3-mini-fast                      | 2025-05-21        | ["GENE
| OCI Generative AI Service | xai.grok-3-mini                           | 2025-05-21        | ["GENE
| OCI Generative AI Service | meta.llama-4-maverick-17b-128e-instruct-fp8 | 2025-05-12      | ["GENE
| OCI Generative AI Service | meta.llama-4-scout-17b-16e-instruct        | 2025-05-12        | ["GENE
| OCI Generative AI Service | cohere.embed-multilingual-image-v3.0       | 2025-05-12        | ["TEXT
| OCI Generative AI Service | cohere.command-a-03-2025                   | 2025-05-12        | ["GENE
| OCI Generative AI Service | meta.llama-3.3-70b-instruct                | 2025-01-15        | ["GENE
| OCI Generative AI Service | meta.llama-3.2-11b-vision-instruct         | 2024-11-13        | ["GENE
| OCI Generative AI Service | cohere.command-r-08-2024                   | 2024-11-13        | ["GENE
| OCI Generative AI Service | meta.llama-3.2-90b-vision-instruct         | 2024-11-13        | ["GENE
| OCI Generative AI Service | cohere.command-r-plus-08-2024              | 2024-11-01        | ["GENE
| OCI Generative AI Service | meta.llama-3.1-405b-instruct               | 2024-09-18        | ["GENE
| OCI Generative AI Service | cohere.embed-english-light-v3.0            | 2023-11-30        | ["TEXT
| OCI Generative AI Service | cohere.embed-english-v3.0                  | 2023-11-30        | ["TEXT
| OCI Generative AI Service | cohere.embed-multilingual-light-v3.0       | 2023-11-30        | ["TEXT
| OCI Generative AI Service | cohere.embed-multilingual-v3.0             | 2023-11-30        | ["TEXT
| OCI Generative AI Service | cohere.embed-english-light-v2.0            | 2023-09-08        | ["TEXT
| OCI Generative AI Service | ModelEndpointOCID                          | 2025-04-22        | ["GENER
+--------------------------+--------------------------------------------+-------------------+--------
```

The list contains IDs of both in-MySQL HeatWave models as well as Oracle Cloud Infrastructure (OCI) Generative AI service models. It also includes OCIDs of the OCI Dedicated AI Cluster model endpoints that are detected. However, to view or use the OCI Generative AI Service models in this list, you need to enable the database system to access OCI services. For more information, see Section 7.5, “Authenticate OCI Generative AI Service”.

MySQL HeatWave In-Database LLMs

The following in-database LLMs are available in MySQL HeatWave:

- As of MySQL 9.3.1:
  - `llama3.2-3b-instruct-v1`: supports English (`en`), French (`fr`), German (`de`), Hindi (`hi`), Italian (`it`), Portuguese (`pt`), Spanish (`es`), and Thai (`th`).

    This is the default LLM used by MySQL HeatWave GenAI for content generation.
  - `llama3.2-1b-instruct-v1`: supports English (`en`), French (`fr`), German (`de`), Hindi (`hi`), Italian (`it`), Portuguese (`pt`), Spanish (`es`), and Thai (`th`).
  - `llama3.1-8b-instruct-v1`: supports English (`en`), French (`fr`), German (`de`), Hindi (`hi`), Italian (`it`), Portuguese (`pt`), Spanish (`es`), and Thai (`th`).
  - `mistral-7b-instruct-v3`: supports English (`en`), French (`fr`), German (`de`), and Spanish (`es`).

- As of MySQL 9.0.1-u1:
  - `llama3-8b-instruct-v1`: supports English (`en`), French (`fr`), German (`de`), Hindi (`hi`), Italian (`it`), Portuguese (`pt`), Spanish (`es`), and Thai (`th`).

    Note

    As of MySQL 9.4.0, support for `llama3-8b-instruct-v1` has been deprecated, and as of MySQL 9.4.1 MySQL HeatWave GenAI automatically replaces it with `llama3.2-3b-instruct-v1`.

- As of MySQL 9.0.0:
  - `mistral-7b-instruct-v1`: supports English (`en`).

    As of MySQL 9.0.1-u1, `mistral-7b-instruct-v1` supports English (`en`), French (`fr`), German (`de`), and Spanish (`es`).

    Note

    As of MySQL 9.4.0, support for `mistral-7b-instruct-v1` has been deprecated, and as of MySQL 9.4.1, MySQL HeatWave GenAI automatically replaces it with `llama3.2-3b-instruct-v1`.

  - `llama2-7b-v1`: supports English (`en`).

    Note

    As of MySQL 9.0.1-u1, support for `llama2-7b-v1` has been deprecated, and as of MySQL 9.4.1, MySQL HeatWave GenAI automatically replaces it with `llama3.2-3b-instruct-v1`.

As of MySQL 9.3.1, the in-database LLMs are included in all MySQL HeatWave Cluster shapes. However, the availability of LLMs depends on the MySQL HeatWave Cluster shape that you are using. The `HeatWave.512GB` shape supports all the in-database LLMs. Whereas, the `HeatWave.Free` and `HeatWave.32GB` shapes support `llama3.2-3b-instruct-v1` and `llama3.2-1b-instruct-v1` only.

In earlier versions of MySQL, the LLMs are included only in the `HeatWave.512GB` shape.

OCI Generative AI Service LLMs

MySQL HeatWave GenAI lets you use OCI Generative AI service LLMs.

As of MySQL 9.4.1, MySQL HeatWave GenAI lets you use OCI Generative AI Service Vision Large Language Models (VLLMs) for generating text-based content pertaining to visual images.

As of MySQL 9.4.0, MySQL HeatWave GenAI lets you use Dedicated AI Clusters for hosting or fine tuning your OCI GenAI Service LLMs.

However, to use the OCI Generative AI Service with MySQL HeatWave GenAI, you need to enable the database system to access OCI services. For more information, see Section 7.5, “Authenticate OCI Generative AI Service”.

Note

These LLMs are available only in the regions supported by the OCI Generative AI Service. For more information, see Pretrained Foundational Models in Generative AI.

As of MySQL 9.3.2, use the inbuilt model explorer to view the full list of available LLMs. This is because the list of OCI Generative AI Service models available in MySQL HeatWave might change before a new version of MySQL is released.

In previous versions of MySQL, the following OCI Generative AI service LLMs are available in MySQL HeatWave:

- As of MySQL 9.2.2:
  - `meta.llama-3.2-90b-vision-instruct`
  - `meta.llama-3.3-70b-instruct`
- As of MySQL 9.1.2:
  - `cohere.command-r-08-2024`
  - `cohere.command-r-plus-08-2024`
  - `meta.llama-3.1-405b-instruct`

To learn about deprecation and removal times of OCI Generative AI service LLMs, see Retiring the Models.

As of MySQL 9.3.1, using OCI Generative AI Service LLMs is supported for all MySQL HeatWave Cluster shapes. In earlier versions of MySQL, the using OCI Generative AI Service LLMs is supported only for the `HeatWave.512GB` shape.

MySQL HeatWave In-Database Embedding Models

The following in-database embedding models are available in MySQL HeatWave:

- `multilingual-e5-small`:
  - As of MySQL 9.3.0, MySQL HeatWave GenAI uses this embedding model, by default, for encoding text or documents in all supported languages.

    In earlier versions of MySQL, MySQL HeatWave GenAI uses this embedding model, by default, for encoding text or documents in all supported languages other than English.
  - As of MySQL 9.2.1, can be used for encoding text or documents in all supported languages.

    In earlier versions of MySQL, can be used for encoding text or documents in all supported languages other than English.
  - This embedding model is available as of MySQL 9.0.1-u1.
- `minilm`:
  - In MySQL 9.2.2 and earlier versions, MySQL HeatWave GenAI uses this embedding model, by default, for encoding English text or documents.
  - As of MySQL 9.2.1, can be used for encoding text or documents in any supported language.

    In earlier versions of MySQL, can be used for encoding text or documents in English only.

  `all_minilm_l12_v2` is an alternative ID for the same model.

OCI Generative AI Service Embedding Models

As of MySQL 9.0.1-u1, MySQL HeatWave GenAI lets you use OCI Generative AI service embedding models. As of MySQL 9.4.0, MySQL HeatWave GenAI also lets you use Dedicated AI Clusters for hosting or fine tuning your OCI GenAI Service embedding models.

Note

These models are available only in the regions supported by the OCI Generative AI Service. For more information, see Pretrained Foundational Models in Generative AI.

However, to use the OCI Generative AI Service with MySQL HeatWave GenAI, you need to enable the database system to access OCI services. For more information, see Section 7.5, “Authenticate OCI Generative AI Service”.

As of MySQL 9.3.2, use the inbuilt model explorer to view the full list of available embedding models. This is because the list of OCI Generative AI Service models available in MySQL HeatWave might change before a new version of MySQL is released.

In previous versions of MySQL, the following OCI Generative AI service embedding models are available in MySQL HeatWave:

- `cohere.embed-english-v3.0`:
  - As of MySQL 9.2.1, can be used for encoding text or files in any supported language.

    In earlier versions of MySQL, can be used for encoding text or files in English only.
- `cohere.embed-multilingual-v3.0`:
  - As of MySQL 9.2.1, can be used for encoding text or files in any supported language.

    In earlier versions of MySQL, can be used for encoding text or files in supported languages other than English.

Languages

As of MySQL 9.0.1-u1, MySQL HeatWave GenAI supports natural-language communication, ingesting documents, as well as generating text-based content in multiple languages. The quality of the generated text outputs depends on the training and ability of the LLM to work with the language.

As of MySQL 9.0.0, MySQL HeatWave GenAI supports natural-language communication, ingesting documents, as well as generating text-based content in English (`en`).

Following is a combined list of languages supported by MySQL HeatWave In-Database LLMs and OCI Generative AI Service LLMs:

- Arabic (`ar`)
- Bengali (`bn`)
- Burmese (`my`)
- Chinese (`zh`)
- Czech (`cs`)
- Dutch (`nl`)
- English (`en`)
- French (`fr`)
- German (`de`)
- Hebrew (`he`)

- Hindi (`hi`)
- Indonesian (`id`)
- Italian (`it`)
- Japanese (`ja`)
- Khmer (`km`)
- Korean (`ko`)
- Lao (`lo`)
- Malay (`ms`)
- Persian (`fa`)
- Polish (`pl`)
- Portuguese (`pt`)
- Spanish (`es`)
- Tagalog (`tl`)
- Thai (`th`)
- Turkish (`tr`)
- Urdu (`ur`)
- Vietnamese (`vi`)

However, not all LLMs support all the languages. To learn which LLM supports which language, see MySQL HeatWave In-Database LLMs and OCI Generative AI Service LLMs.

Note

To set the value of the `language` parameter in MySQL HeatWave GenAI routines that support this parameter, do not use the language name to specify the language. Use the two-letter `ISO 639-1` code for the language instead. For example, to use French, use the `ISO 639-1` code for French, which is `fr`.

What's Next

- Learn how to authenticate OCI generative AI service.
- Learn more about generating text-based content.

7.5 Authenticate OCI Generative AI Service

To enable the DB system to access OCI services, perform the following steps in OCI:

1. Create a new dynamic group or update a dynamic group's matching rules, and the following matching rule to include all MySQL HeatWave DB systems in the defined compartment:

   ```
   ALL{resource.type = 'mysqldbsystem', resource.compartment.id = 'ocid1.compartment.oc1..AlphanumericS
   ```

2. Add policies for the dynamic group that grant the dynamic group access to the target service's APIs:

   ```
   allow dynamic-group IdentityDomainName/GroupName to use generative-ai-chat in compartment Compartmer
   ```

```
allow dynamic-group IdentityDomainName/GroupName to use generative-ai-text-embedding in compartment Com
allow dynamic-group IdentityDomainName/GroupName to inspect generative-ai-model in compartment Compartm
```

Replace the following:

- *IdentityDomainName*: the identity domain name.

  Note

  If the dynamic group belongs to the default identity domain, you can omit specifying the identity domain name.

- *GroupName*: the dynamic group name
- *CompartmentID*: the compartment ID of the DB system

For more information, see Resource Principals.

In addition, for resource principal authentication, MySQL HeatWave automatically sets the `rapid_ml_genai` system variable. The default value of the variable is the compartment ID of the DB system you are using.

To use a different compartment for accessing the OCI Generative AI service, you can set the `rapid_ml_genai_session` system variable to specify the ID of the compartment you want to use, as shown below:

```
SET rapid_ml_genai_session = '{"compartment_id": "ocid1.compartment.oc1..AlphanumericString"}';
```

The compartment ID specified in the `rapid_ml_genai_session` system variable must match the compartment ID you used to create the dynamic group and add the OCI Generative AI Service authentication policy.

What's Next

Learn how to perform the following tasks:

- Generate Text-Based Content
- Set Up a Vector Store
- Generate Vector Embeddings
- Perform a Vector Search
- Start a Conversational Chat

7.6 Generate Text-Based Content

The topics in this section describe how to generate and summarize text-based content using MySQL HeatWave GenAI.

7.6.1 Generate New Content

The following sections in this topic describe how to generate new text-based content using MySQL HeatWave GenAI:

- Before You Begin
- Generate Content
- Run Batch Queries

- What's Next

Before You Begin

- Review the MySQL HeatWave GenAI requirements and privileges.
- If you want to use the OCI Generative AI Service LLMs, complete the steps to authenticate OCI Generative AI Service.
- Connect to your MySQL HeatWave DB System.
- To Run Batch Queries, add the natural-language queries to a column in a new or existing table.

Generate Content

To generate text-based content using MySQL HeatWave GenAI, perform the following steps:

1. To define your natural-language query, set the `@query` variable:

   ```
   mysql> SET @query="QueryInNaturalLanguage";
   ```

   Replace *`QueryInNaturalLanguage`* with a natural-language query of your choice. For example:

   ```
   mysql> SET @query="Write an article on Artificial intelligence in 200 words.";
   ```

2. To generate text-based content, pass the query to the LLM using the `ML_GENERATE` routine with the `task` parameter set to `generation`:

   ```
   mysql> SELECT sys.ML_GENERATE(@query,
   JSON_OBJECT("task", "generation", "model_id", "ModelID", "language", "Language"));
   ```

   Replace the following:

   - *`ModelID`*: LLM to use.
   - *`Language`*: the two-letter `ISO 639-1` code for the language you want to use. Default language is `en`, which is English. To view the list of supported languages, see Languages.

     The `language` parameter is supported as of MySQL 9.0.1-u1.

   For example:

   ```
   mysql> SELECT sys.ML_GENERATE(@query,
   JSON_OBJECT("task", "generation", "model_id", "mistral-7b-instruct-v3", "language", "en"));
   ```

   Text-based content that is generated by the LLM in response to your query is printed as output. It looks similar to the text output shown below:

   ```
   | {"text": "\nTitle: Artificial Intelligence in 2023: A New Era of
   Innovation\n\nArtificial Intelligence (AI) has been a transformative
   force, reshaping various sectors and revolutionizing our daily lives.
   In 2023, AI continues to evolve at an unprecedented pace, offering
   unparalleled opportunities for innovation and growth.\n\nOne significant
   development is the advancement of machine learning algorithms, enabling
   AI systems to learn from vast amounts of data more efficiently than
   ever before. This progress has led to improvements in areas such as image
   recognition, natural language processing, and predictive analytics, making
   AI more intuitive and user-friendly.\n\nAnother notable trend is the
   increasing integration of AI into everyday devices, from smartphones to
   home appliances. These advancements not only enhance convenience but also
   contribute to energy savings and improved safety features.\n\nIn the
   realm of healthcare, AI is playing a crucial role in diagnostics, drug
   discovery, and personalized treatment plans. By analyzing vast amounts of
   medical data, AI can help doctors make more accurate diagnoses and develop
   targeted therapies, ultimately improving patient outcomes.\n\nHowever,
   with great power comes great responsibility. As AI becomes more pervasive,"} |
   ```

Run Batch Queries

To run multiple `generation` queries in parallel, use the `ML_GENERATE_TABLE` routine. This method is faster than running the `ML_GENERATE` routine multiple times.

Note

In versions older than MySQL 9.2.1, to alter an existing table or create a new table, MySQL requires you to set the sql-require-primary-key system variable to `0`.

The `ML_GENERATE_TABLE` routine is supported as of MySQL 9.0.1-u1.

To run the steps in this section, you can create a new database `demo_db` and table `input_table`:

```
mysql> CREATE DATABASE demo_db;
mysql> USE demo_db;
mysql> CREATE TABLE input_table (id INT AUTO_INCREMENT, Input TEXT, primary key (id));
mysql> INSERT INTO input_table (Input) VALUES('Describe what is MySQL in 50 words.');
mysql> INSERT INTO input_table (Input) VALUES('Describe Artificial Intelligence in 50 words.');
mysql> INSERT INTO input_table (Input) VALUES('Describe Machine Learning in 50 words.');
```

To run batch queries using `ML_GENERATE_TABLE`, perform the following steps:

1. In the `ML_GENERATE_TABLE` routine, specify the table columns containing the input queries and for storing the generated text-based responses:

   ```
   mysql> CALL sys.ML_GENERATE_TABLE("InputDBName.InputTableName.InputColumn", "OutputDBName.OutputTableNa
   JSON_OBJECT("task", "generation", "model_id", "ModelID", "language", "Language"));
   ```

   Replace the following:

   - *`InputDBName`*: the name of the database that contains the table column where your input queries are stored.
   - *`InputTableName`*: the name of the table that contains the column where your input queries are stored.
   - *`InputColumn`*: the name of the column that contains input queries.
   - *`OutputDBName`*: the name of the database that contains the table where you want to store the generated outputs. This can be the same as the input database.
   - *`OutputTableName`*: the name of the table where you want to create a new column to store the generated outputs. This can be the same as the input table. If the specified table doesn't exist, a new table is created.
   - *`OutputColumn`*: the name for the new column where you want to store the output generated for the input queries.
   - *`ModelID`*: LLM to use.
   - *`Language`*: the two-letter `ISO 639-1` code for the language you want to use. Default language is `en`, which is English. To view the list of supported languages, see Languages.

   For example:

   ```
   mysql> CALL sys.ML_GENERATE_TABLE("demo_db.input_table.Input", "demo_db.output_table.Output",
   JSON_OBJECT("task", "generation", "model_id", "mistral-7b-instruct-v3", "language", "en"));
   ```

2. View the contents of the output table:

   ```
   mysql> SELECT * FROM output_table\G
   *************************** 1. row ***************************
       id: 1
   ```

```
Output: {"text": "\nMySQL is an open-source relational database
management system (RDBMS) that uses SQL (Structured Query
Language) to manage and manipulate data. It's widely used for
web-based applications, including those built with PHP, Python,
and Java. MySQL offers high performance, reliability, and
scalability, making it a popular choice for developers
worldwide.\n\n[DONE]", "error": null}
*************************** 2. row ***************************
    id: 2
Output: {"text": "\nArtificial Intelligence (AI) refers to the
simulation of human intelligence in machines that are programmed
to think, learn, and make decisions like humans. AI can perform
tasks such as visual perception, speech recognition,
decision-making, and language translation. It's a rapidly
evolving field with applications in various sectors including
healthcare, finance, and entertainment.\n\n[END]",
"error": null}
*************************** 3. row ***************************
    id: 3
Output: {"text": "\nMachine Learning is a method of data
analysis that automates the building of analytical models. It's
based on algorithms and statistical models that computer
systems use to perform tasks without being explicitly programmed,
learning from data instead.\n\n[END]", "error": null}
```

As of MySQL 9.3.0, the output table generated using the `ML_GENERATE_TABLE` routine contains an additional details for error reporting. In case the routine fails to generate output for specific rows, details of the errors encountered and default values used are added for the row in the output column.

If you created a new database for testing the steps in this section, ensure that you delete the database to avoid being billed for it:

```
mysql> DROP DATABASE demo_db;
```

To learn more about the available routine options, see ML_GENERATE_TABLE Syntax.

What's Next

Learn how to Summarize Existing Content.

7.6.2 Summarize Content

The following sections in this topic describe how to summarize exiting content using MySQL HeatWave GenAI:

- Before You Begin
- Summarize Text
- Run Batch Queries
- What's Next

Before You Begin

- Review the MySQL HeatWave GenAI requirements and privileges.
- If you want to use the OCI Generative AI Service LLMs, complete the steps to authenticate OCI Generative AI Service.
- Connect to your MySQL HeatWave DB System.
- To Run Batch Queries, add the natural-language queries to a column in a new or existing table.

Summarize Text

To summarize text, perform the following steps:

1. To define the text that you want to summarize, set the `@text` variable:

   ```
   mysql> SET @text="TextToSummarize";
   ```

   Replace *`TextToSummarize`* with the text that you want to summarize.

   For example:

   ```
   mysql> SET @text="Artificial Intelligence (AI) is a rapidly growing field that has the potential
   to revolutionize how we live and work. AI refers to the development of computer systems
   that can perform tasks that typically require human intelligence, such as visual perception,
   speech recognition, decision-making, and language translation. One of the most significant
   developments in AI in recent years has been the rise of machine learning, a subset of AI that
   allows computers to learn from data without being explicitly programmed. Machine learning
   algorithms can analyze vast amounts of data and identify patterns, making them increasingly
   accurate at predicting outcomes and making decisions. AI is already being used in a variety
   of industries, including healthcare, finance, and transportation. In healthcare, AI is being
   used to develop personalized treatment plans for patients based on their medical history and
   genetic makeup. In finance, AI is being used to detect fraud and make investment
   recommendations In transportation, AI is being used to develop self-driving cars and improve
   traffic flow. Despite the many benefits of AI, there are also concerns about its potential impact
   on society. Some worry that AI could lead to job displacement, as machines become more capable
   of performing tasks traditionally done by humans.";
   ```

2. To generate the text summary, pass the original text to the LLM using the `ML_GENERATE` routine, with the `task` parameter set to `summarization`:

   ```
   mysql> SELECT sys.ML_GENERATE(@query,
   JSON_OBJECT("task", "summarization", "model_id", "ModelID", "language", "Language"));
   ```

   Replace the following:

   - *`ModelID`*: LLM to use.
   - *`Language`*: the two-letter `ISO 639-1` code for the language you want to use. Default language is `en`, which is English. To view the list of supported languages, see Languages.

     The `language` parameter is supported as of MySQL 9.0.1-u1.

   For example:

   ```
   mysql> SELECT sys.ML_GENERATE(@text,
   JSON_OBJECT("task", "summarization", "model_id", "mistral-7b-instruct-v3", "language", "en"));
   ```

   A text summary generated by the LLM in response to your query is printed as output. It looks similar to the text output shown below:

   ```
   | {"text": " Artificial Intelligence (AI) is a growing field that uses computer
   systems to perform tasks requiring human intelligence, such as visual perception
   and decision-making. Machine learning, a subset of AI, allows computers to learn
   from data without being explicitly programmed. AI is already being used in various
   industries, including healthcare, finance, and transportation. However, there are
   concerns about job displacement due to its increasing capabilities."} |
   ```

Run Batch Queries

To run multiple `summarization` queries in parallel, use the `ML_GENERATE_TABLE` routine. This method is faster than running the `ML_GENERATE` routine multiple times.

Note

In versions older than MySQL 9.2.1, to alter an existing table or create a new table, MySQL requires you to set the sql-require-primary-key system variable to `0`.

The `ML_GENERATE_TABLE` routine is supported as of MySQL 9.0.1-u1.

To run the steps in this section, create a new database `demo_db` and table `input_table`:

```
mysql> CREATE DATABASE demo_db;
mysql> USE demo_db;
mysql> CREATE TABLE input_table (id INT AUTO_INCREMENT, Input TEXT, primary key (id));
mysql> INSERT INTO input_table (Input) VALUES(
  CONCAT(
  'MySQL is a widely used open-source relational database management system or RDBMS that ',
  'is based on the SQL standard. It is designed to be highly scalable, reliable, and secure, ',
  'making it an ideal choice for businesses of all sizes. MySQL uses a client-server ',
  'architecture, where the server stores and manages the data, while clients connect to the ',
  'server to access and manipulate the data. The MySQL server can be installed on a variety ',
  'of operating systems, including Linux, Windows, and macOS. One of the key features of MySQL ',
  'is its support for stored procedures, which allow developers to create reusable blocks of ',
  'code that can be executed multiple times. This makes it easier to manage complex database ',
  'operations and reduces the amount of code that needs to be written. MySQL also supports ',
  'a wide range of data types, including integers, floating-point numbers, dates, and strings. ',
  'It also has built-in support for encryption, which helps to protect sensitive data from ',
  'unauthorized access. Another important feature of MySQL is its ability to handle large ',
  'amounts of data. It can scale horizontally by adding more servers to the cluster, or ',
  'vertically by upgrading the hardware.'
  )
);
mysql> INSERT INTO input_table (Input) VALUES(
  CONCAT(
  'Artificial Intelligence or AI refers to the simulation of human intelligence in machines ',
  'that are programmed to think and act like humans. The goal of AI is to create systems that ',
  'can function intelligently and independently, exhibiting traits associated with human ',
  'intelligence such as reasoning, problem-solving, perception, learning, and understanding ',
  'language. There are two main types of AI: narrow or weak AI, and general or strong AI. ',
  'Narrow AI is designed for a specific task and is limited in its abilities, while general ',
  'AI has the capability to understand or learn any intellectual task that a human being can. ',
  'AI technologies include machine learning, which allows systems to improve their performance ',
  'based on data, and deep learning, which involves the use of neural networks to model complex ',
  'patterns. Other AI techniques include natural language processing, robotics, and expert systems. ',
  'AI has numerous applications across various industries, including healthcare, finance, ',
  'transportation, and education. It has the potential to revolutionize the way we live and work ',
  'by automating tasks, improving efficiency, and enabling new innovations. However, there are ',
  'also concerns about the impact of AI on employment, privacy, and safety.'
  )
);
mysql> INSERT INTO input_table (Input) VALUES(
   CONCAT(
  'Machine learning is a subset of artificial intelligence that involves the development of ',
  'algorithms and statistical models that enable systems to improve their performance on a ',
  'specific task over time by learning from data. At its core, machine learning is about ',
  'using data to train machines to make predictions or decisions without being explicitly ',
  'programmed to do so. There are many different types of machine learning, including ',
  'supervised learning, unsupervised learning, and reinforcement learning. In supervised learning, ',
  'the algorithm is trained on labeled data, meaning that the input data has been categorized or ',
  'classified by a human. The goal of supervised learning is to enable the machine to make predictions
  'based on this training data. Unsupervised learning, on the other hand, involves training the ',
  'algorithm on unlabeled data. In this case, the algorithm must identify patterns and relationships ',
  'in the data on its own. This type of learning is often used for tasks such as clustering or anomaly
  'detection. Reinforcement learning involves an agent interacting with an environment and learning by
  'trial and error. The agent receives feedback in the form of rewards or punishments, which it uses ',
  'to improve its behavior over time. This type of learning is often used in game playing or robotics.'
  )
);
```

To run batch queries using `ML_GENERATE_TABLE`, perform the following steps:

1. In the `ML_GENERATE_TABLE` routine, specify the table columns containing the input queries and for storing the generated text summaries:

   ```
   mysql> CALL sys.ML_GENERATE_TABLE("InputDBName.InputTableName.InputColumn", "OutputDBName.OutputTabl
   JSON_OBJECT("task", "summarization", "model_id", "ModelID", "language", "Language"));
   ```

   Replace the following:

- *`InputDBName`*: the name of the database that contains the table column where your input queries are stored.
- *`InputTableName`*: the name of the table that contains the column where your input queries are stored.
- *`InputColumn`*: the name of the column that contains input queries.
- *`OutputDBName`*: the name of the database that contains the table where you want to store the generated outputs. This can be the same as the input database.
- *`OutputTableName`*: the name of the table where you want to create a new column to store the generated outputs. This can be the same as the input table. If the specified table doesn't exist, a new table is created.
- *`OutputColumn`*: the name for the new column where you want to store the output generated for the input queries.
- *`ModelID`*: LLM to use.
- *`Language`*: the two-letter `ISO 639-1` code for the language you want to use. Default language is `en`, which is English. To view the list of supported languages, see Languages.

For example:

```
mysql> CALL sys.ML_GENERATE_TABLE("demo_db.input_table.Input", "demo_db.output_table.Output",
JSON_OBJECT("task", "summarization", "model_id", "mistral-7b-instruct-v3", "language", "en"));
```

2. View the contents of the output table:

```
mysql> SELECT * FROM output_table\G
*************************** 1. row ***************************
    id: 1
Output: {"text": " MySQL is an open-source RDBMS based on the
SQL standard, designed for scalability, reliability, and
security. It uses a client-server architecture, supports stored
procedures, a wide range of data types, encryption, and can
handle large amounts of data by scaling horizontally or
vertically.", "error": null}
*************************** 2. row ***************************
    id: 2
Output: {"text": " Artificial Intelligence (AI) is the
simulation of human intelligence in machines, designed to
think and act like humans. It includes two types: narrow AI
for specific tasks and general AI with broader capabilities.
Techniques include machine learning, deep learning, natural
language processing, robotics, and expert systems. AI has
various applications across industries but raises concerns
about employment, privacy, and safety.", "error": null}
*************************** 3. row ***************************
    id: 3
Output: {"text": " Machine learning is a subset of artificial
intelligence that involves training machines to make predictions
or decisions without being explicitly programmed. It includes
supervised, unsupervised, and reinforcement learning. Supervised
learning uses labeled data, while unsupervised learning uses
unlabeled data. Reinforcement learning involves an agent
interacting with an environment and learning through trial
and error.", "error": null}
```

As of MySQL 9.3.0, the output table generated using the `ML_GENERATE_TABLE` routine contains an additional details for error reporting. In case the routine fails to generate output for specific rows, details of the errors encountered and default values used are added for the row in the output column.

If you created a new database for testing the steps in this section, ensure that you delete the database to avoid being billed for it:

```
mysql> DROP DATABASE demo_db;
```

To learn more about the available routine options, see ML_GENERATE_TABLE Syntax.

What's Next

Learn how to Summarize Unstructured Files.

7.6.3 Summarize Unstructured Files

As of MySQL 9.4.1, MySQL HeatWave GenAI lets you generate text summaries for unstructured files available in Object Storage. You need to first ingest these documents into vector store tables using Auto Parallel Load. The generated table contains text segments that are used as input text for the `ML_GENERATE` and `ML_GENERATE_TABLE` routines to generate summaries of the content available in the ingested files.

This topic contains the following sections:

- Before You Begin
- Generate Summary for a Single File
- Generate Summaries for Multiple Files
- What's Next

Before You Begin

- Review the requirements to set up a vector store.

  To Generate Summary for a Single File, create an Object Storage bucket with the name `demo_bucket`. Download the MySQL HeatWave technical brief PDF, then upload it to `demo_bucket`. To Generate Summaries for Multiple Files, also download the MySQL HeatWave on AWS brief PDF and upload it to `demo_bucket`.

- Complete the steps to Ingest Files Using Auto Parallel Load with the `split_by` option set to `document` for creating one text segment per document in the resulting vector store table.

  If a file contains more than 1 million characters, all content beyond this limit is truncated. Therefore, in such cases, it is recommended to use alternative `split_by` option values such as `page` or `paragraph`. You can later concatenate the resulting text segments to reconstruct the full document.

Generate Summary for a Single File

Perform the following steps:

- Copy the text segment of the file from the vector store table to the `@document` variable:

  ```
  mysql> SELECT segment INTO @document FROM DBName.VectorStoreTableName WHERE document_id=0;
  ```

  Replace the following:

  - *`DBName`*: the name of the database.
  - *`VectorStoreTableName`*: the name of the vector store table.

  For example:

  ```
  mysql> SELECT segment INTO @document FROM demo_db.demo_embeddings_apl WHERE document_id=0;
  ```

- Generate a summary of the text segments using the ML_GENERATE routine:

```
mysql> CALL sys.ML_GENERATE(@document, JSON_OBJECT("task", "summarization", "model_id", "ModelID", "max_t
```

Replace the following: *ModelID* with the LLM to use.

- *ModelID*: the LLM to use.
- *MaxTokens*: specify the maximum token value depending on how long you want the summary.

For example:

```
mysql> SELECT sys.ML_GENERATE(@document, JSON_OBJECT("task", "summarization", "model_id", "mistral-7b-ins
```

It take a few minutes for this call to complete. When completed, a text summary generated by the LLM is printed as output. It looks similar to the text output shown below:

```
+-----------------------------------------------------------------------------------------------------------
| sys.ML_GENERATE(@document, JSON_OBJECT("task", "summarization", "model_id", "mistral-7b-instruct-v3", "
+-----------------------------------------------------------------------------------------------------------
| {"text": " MySQL HeatWave is a fully managed cloud service that provides integrated, automated, and sec
+-----------------------------------------------------------------------------------------------------------
```

Generate Summaries for Multiple Files

Perform the following steps:

- Using the database name, vector store table name, and document IDs, copy the text segments of the files from the vector store table to different variables:

```
mysql> SELECT segment INTO @document1 FROM DBName.VectorStoreTableName WHERE document_id=DocumentID;
mysql> SELECT segment INTO @document2 FROM DBName.VectorStoreTableName WHERE document_id=DocumentID;
...
```

These segments can be from the same or different vector store tables.

For example:

```
mysql> SELECT segment INTO @document1 FROM demo_db.demo_embeddings_apl WHERE document_id=0;
mysql> SELECT segment INTO @document2 FROM demo_db.demo_embeddings_apl WHERE document_id=1;
```

- Add these text segments to a table:

```
mysql> CREATE TABLE input_table (id INT AUTO_INCREMENT, Input TEXT, primary key (id));
mysql> INSERT INTO input_table (Input) VALUES(@document1);
mysql> INSERT INTO input_table (Input) VALUES(@document2);
```

- Generate summaries of the text segments using the ML_GENERATE_TABLE routine:

```
mysql> CALL sys.ML_GENERATE_TABLE("InputDBName.InputTableName.InputColumn", "OutputDBName.OutputTableName
JSON_OBJECT("task", "summarization", "model_id", "ModelID", "language", "Language"));
```

Replace the following:

- *InputDBName*: the name of the database that contains the table column where your input queries are stored.
- *InputTableName*: the name of the table that contains the column where your input queries are stored.
- *InputColumn*: the name of the column that contains input queries.
- *OutputDBName*: the name of the database that contains the table where you want to store the generated outputs. This can be the same as the input database.

- *`OutputTableName`*: the name of the table where you want to create a new column to store the generated outputs. This can be the same as the input table. If the specified table doesn't exist, a new table is created.
- *`OutputColumn`*: the name for the new column where you want to store the output generated for the input queries.
- *`ModelID`*: LLM to use.
- *`Language`*: the two-letter `ISO 639-1` code for the language you want to use. Default language is `en`, which is English. To view the list of supported languages, see Languages.

For example:

```
mysql> CALL sys.ML_GENERATE_TABLE("demo_db.input_table.Input", "demo_db.output_table.Output",
JSON_OBJECT("task", "summarization", "model_id", "mistral-7b-instruct-v3", "language", "en"));
```

It take a few minutes for this call to complete.

- View the contents of the output table:

```
mysql> SELECT * FROM output_table\G
*************************** 1. row ***************************
    id: 1
Output: {"text": "\nThe text is a business and technical brief for MySQL HeatWave on AWS, a fully man
*************************** 2. row ***************************
    id: 2
Output: {"text": " MySQL HeatWave is a fully managed cloud service that provides integrated, automate
```

What's Next

Learn more about Setting Up a Vector Store.

7.7 Set Up a Vector Store

Using the inbuilt vector store and retrieval-augmented generation (RAG), you can load and query unstructured documents stored in Object Storage using natural language within the MySQL HeatWave ecosystem.

The sections in this topic describe how to set up an inbuilt vector store.

7.7.1 About MySQL HeatWave Vector Store

This section describes the Vector Store functionality available in MySQL HeatWave.

About Vector Store

MySQL HeatWave vector store is a relational database that lets you load unstructured data to MySQL HeatWave Lakehouse. It automatically parses unstructured data formats, which include PDF (including scanned PDF files), PPT, TXT, HTML, and DOC file formats, from Object Storage. Then, it segments the parsed data, creates vector embeddings, and stores them for MySQL HeatWave GenAI to perform semantic searches.

MySQL HeatWave vector store uses the native `VECTOR` data type to store unstructured data in a multi-dimensional space. Each point in a vector store represents the vector embedding of the corresponding data. Semantically similar data is placed closer in the vector space.

The large language models (LLMs) available in MySQL HeatWave GenAI are trained on publicly available data. Therefore, the responses generated by these LLMs are based on publicly available information. To generate content relevant to your proprietary data, you must store your proprietary enterprise data, which has been converted to vector embeddings, in a vector store. This enables the in-database retrieval-augmented generation (RAG) system to perform a semantic search in the

proprietary data stored in the vector stores to find appropriate content, which is then fed to the LLM to help it generate more accurate and relevant responses.

About Vector Processing

To create vector embeddings, MySQL HeatWave GenAI uses in-database embedding models, which are encoders that converts sequence of words and sentences from documents into numerical representations. These numerical values are stored as vector embeddings in the vector store and capture the semantics of the data and relationships to other data.

A vector distance function measures the similarity between vectors by calculating the mathematical distance between two multi-dimensional vectors.

MySQL HeatWave GenAI encodes your queries using the same embedding model that is used to encode the ingested data to create the vector store. It then uses the right distance function to find relevant content with similar semantic meaning from the vector store to perform RAG.

About Optical Character Recognition

Optical Character Recognition (OCR) in MySQL HeatWave GenAI lets you extract and encode text from images stored in unstructured documents. The text extracted from images is converted into vector embeddings and stored in a vector store the same way regular text in unstructured documents is encoded and stored in a vector store.

OCR is supported as of MySQL 9.1.0.

- As of MySQL 9.1.2, OCR is enabled by default when you ingest files using Auto Parallel Load or Asynchronous Load.

  However, when OCR is enabled, the loading process slows down because MySQL HeatWave GenAI scans all images available in the files and pages of scanned documents that you are ingesting into the vector store. If OCR is not required for the documents that you are ingesting, you can disable OCR to speed up the loading process.

- As of MySQL 9.1.1, you need to enable OCR when you ingest files using Auto Parallel Load or Asynchronous Load.

- As of MySQL 9.1.0, OCR is only supported in Auto Parallel Load. It is not supported in Asynchronous Load. However, you need to enable OCR while ingesting files using Auto Parallel Load.

MySQL HeatWave GenAI supports OCR in the following unstructured data formats: PDF (including scanned PDF files), DOC, DOCX, PPT, and PPTX. However, MySQL HeatWave GenAI doesn't support OCR in TXT and HTML files. Images stored in TXT and HTML files are ignored while ingesting the files.

OCR in MySQL HeatWave GenAI also has the following limitations:

- MySQL HeatWave GenAI might not be able to extract and process the text from images with 100% accuracy. However, if there are minor character recognition errors, the overall meaning of the text is still preserved.

- In some cases, text-like figures in images might incorrectly be treated as regular text.

- MySQL HeatWave GenAI doesn't support OCR for Scalable Vector Graphic (SVG) images in PDF files.

About Ingesting Complex Documents

As of MySQL 9.4.2, MySQL HeatWave GenAI supports enhanced document parsing for PDF files using Vision Language Models (VLMs). This helps improves the document parsing quality by extracting complex structures like tables, and enhances RAG as well as vector search queries. To specify the VLM to use, you can use the `document_parser_model` option, which is available in Auto Parallel Load only.

For more information, see Ingest Files Using Auto Parallel Load.

What's Next

Learn how to Ingest Files into a Vector Store.

7.7.2 Ingest Files into a Vector Store

This section describes how to generate vector embeddings for files or folders stored in Object Storage, and load the embeddings into a vector store table.

The following sections in this topic describe how to ingest files into a vector store:

- Before You Begin
- Ingest Files Using Asynchronous Load
- Using the Uniform Resource Identifier with Asynchronous Load
- Using a Pre-Authenticated Request with Asynchronous Load
- Ingest Files Using Auto Parallel Load
- Clean Up
- What's Next

Before You Begin

- Review the MySQL HeatWave GenAI requirements and privileges.
- If not already done, create an Oracle Cloud Infrastructure (OCI) bucket for storing files that you want to ingest into the vector store. Then, upload the files to the bucket.

  Vector store can ingest files in the following formats: PDF, PPT, TXT, HTML, and DOC.

  To run the steps in this topic, create an Object Storage bucket with the name `demo_bucket`. Download the MySQL HeatWave user guide PDF, then upload it to `demo_bucket`.
- If you want to use the OCI Generative AI Service Embedding Models when you Ingest Files Using Auto Parallel Load, complete the steps to authenticate OCI Generative AI Service.
- Connect to your MySQL HeatWave DB System.

Note

X protocol is not supported for Ingest Files Using Asynchronous Load. To set up a vector store using this method, ensure that you use the classic MySQL protocol while connecting to the database:

```
mysqlsh -uDBSystemAdminName -pDBSystemPassword -hDBSystemPrivateIP --sqlc
```

- To create and store vector store tables using the steps described in this topic, you can create a new database `demo_db`:

```
mysql> CREATE DATABASE demo_db;
```

Ingest Files Using Asynchronous Load

The `VECTOR_STORE_LOAD` routine creates and loads vector embeddings asynchronously into the vector store. You can ingest the source files into the vector store using the following methods:

- Using the Uniform Resource Identifier
- Using a Pre-Authenticated Request

Using the Uniform Resource Identifier with Asynchronous Load

This section describes how to load source documents from the bucket into the vector table using the uniform resource identifier (URI) of the object.

Note

To use this method, you need to enable the DB system to access the OCI Object Storage bucket. For more information, see Resource Principals.

To set up a new vector store using an object URI, perform the following steps:

1. To create the vector store table, use a new or existing database:

   ```
   mysql> USE DBName;
   ```

   Replace *DBName* with the database name.

   For example:

   ```
   mysql> USE demo_db;
   ```

2. As of MySQL 9.3.1, this step is not required. MySQL HeatWave no longer uses the `mysql_task_management` schema, so you can delete the schema if you are not using it.

   In earlier versions of MySQL, if you are loading a vector store table on a database system for the first time, create a new schema dedicated for task management:

   ```
   SELECT mysql_task_management_ensure_schema();
   ```

   This creates a new schema, `mysql_task_management`, which keeps track of the tasks that run in the background. This schema contains internal tables that store the task details such as task ID and task logs. These internal tables must not be altered as it can cause the asynchronous task created by `VECTOR_STORE_LOAD` to fail.

3. Optionally, to specify a name for the vector store table and language to use, set the `@options` variable:

   ```
   mysql> SET @options = JSON_OBJECT("table_name", "VectorStoreTableName", "language", "Language");
   ```

   Replace the following:

   - *VectorStoreTableName*: the name you want for the vector store table.
   - *Language*: the two-letter `ISO 639-1` code for the language you want to use. Default language is `en`, which is English. To view the list of supported languages, see Languages.

     The `language` parameter is supported as of MySQL 9.0.1-u1.

   To view the list of available routine options, see Section 10.3.3, “VECTOR_STORE_LOAD”.

   For example:

   ```
   mysql> SET @options = JSON_OBJECT("table_name", "demo_embeddings", "language", "en");
   ```

   As of MySQL 9.1.2, Optical Character Recognition (OCR) is enabled by default.

4. To ingest the file from the Object Storage, create vector embeddings, and load the vector embeddings into MySQL HeatWave, use the `VECTOR_STORE_LOAD` routine:

   ```
   mysql> CALL sys.VECTOR_STORE_LOAD("oci://BucketName@Namespace/Path/Filename", @options);
   ```

   Replace the following:

- *BucketName*: the OCI Object Storage bucket name.
- *Namespace*: the name of the bucket namespace.
- *Path*: path to the folder that contains the source file.
- *Filename*: the filename with the file extension.

For example:

```
mysql> CALL sys.VECTOR_STORE_LOAD("oci://demo_bucket@demo_namespace/heatwave-en.pdf", @options);
```

This creates an asynchronous task that runs in background and loads the vector embeddings into the specified vector store table. The output of the VECTOR_STORE_LOAD routine contains the following:

- An ID of the task which was created.
- A task query that you can use to track the progress of task.

If MySQL HeatWave GenAI detects multiple files with the same or different file formats in a single load, it creates a separate table for every format it finds. The table name for each format is the specified or default table name followed by the format. For example, demo_embeddings_pdf is the name of the table that contains PDF files.

5. After the task is completed, verify that embeddings are loaded in the vector store table:

```
mysql> SELECT COUNT(*) FROM VectorStoreTableName;
```

For example:

```
mysql> SELECT COUNT(*) FROM demo_embeddings;
```

If you see a numerical value in the output, your embeddings are successfully loaded in the vector store table.

6. To view the details of the vector store table, use the following statement:

```
mysql> DESCRIBE demo_embeddings;
+-------------------+---------------+------+-----+---------+-------+
| Field             | Type          | Null | Key | Default | Extra |
+-------------------+---------------+------+-----+---------+-------+
| document_name     | varchar(1024) | NO   |     | NULL    |       |
| metadata          | json          | NO   |     | NULL    |       |
| document_id       | int unsigned  | NO   | PRI | NULL    |       |
| segment_number    | int unsigned  | NO   | PRI | NULL    |       |
| segment           | varchar(1024) | NO   |     | NULL    |       |
| segment_embedding | vector(384)   | NO   |     | NULL    |       |
| segment_metadata  | json          | NO   |     | NULL    |       |
+-------------------+---------------+------+-----+---------+-------+
```

Using a Pre-Authenticated Request with Asynchronous Load

This section describes how to ingest source documents from the using pre-authenticated requests (PAR). Use this method if OCI bucket access is not enabled on your DB system.

Note

For confidential data, Using the Uniform Resource Identifier with Asynchronous Load is recommended for ingesting the source files into the vector store as it is a more secure method.

To set up a new vector store, perform the following steps:

1. To create the vector store table, use a new or existing database:

   ```
   mysql> USE DBName;
   ```

   Replace *DBName* with the database name.

   For example:

   ```
   mysql> USE demo_db;
   ```

2. As of MySQL 9.3.1, this step is not required.

   In earlier versions of MySQL, if you are loading a vector store table on a database system for the first time, create a new schema dedicated for task management:

   ```
   SELECT mysql_task_management_ensure_schema();
   ```

   This creates a new schema, `mysql_task_management`, which keeps track of the tasks that run in the background. This schema contains internal tables that store the task details such as task ID and task logs. These internal tables must not be altered as it can cause the asynchronous task created by `VECTOR_STORE_LOAD` to fail.

3. Optionally, to specify a name for the vector store table and language to use, set the `@options` variable:

   ```
   mysql> SET @options = JSON_OBJECT("table_name", "VectorStoreTableName", "language", "Language");
   ```

   Replace the following:

   - *`VectorStoreTableName`*: the name you want for the vector store table.
   - *`Language`*: the two-letter `ISO 639-1` code for the language you want to use. Default language is `en`, which is English. To view the list of supported languages, see Languages.

     The `language` parameter is supported as of MySQL 9.0.1-u1.

   To view the list of available routine options, see Section 10.3.3, “VECTOR_STORE_LOAD”.

   For example:

   ```
   mysql> SET @options = JSON_OBJECT("table_name", "demo_embeddings_par", "language", "en");
   ```

   As of MySQL 9.1.2, Optical Character Recognition (OCR) is enabled by default.

4. To ingest the file from the Object Storage, create vector embeddings, and load the vector embeddings into MySQL HeatWave, use the `VECTOR_STORE_LOAD` routine:

   ```
   mysql> CALL sys.VECTOR_STORE_LOAD("PAR", @options);
   ```

   Replace *PAR* with PAR of the bucket, folder, or file that you want to use to set up the vector store.

   To learn how to create PAR for your Object Storage bucket, see Creating a PAR Request in .

   Note

   If you are creating a PAR for a folder or Object Storage bucket, then select *Enable Object Listing* in the *Create Pre-Authenticated Request* dialog to enable object listing.

   For example:

   ```
   mysql> CALL sys.VECTOR_STORE_LOAD("https://demo.objectstorage.demo-region.oci.customer-oci.com/p/demo-u
   ```

```
@options);
```

This creates a task that runs in background and loads the vector embeddings into the specified vector store table. The output of the `VECTOR_STORE_LOAD` routine contains the following:

- An ID of the task which was created.
- A task query that you can use to track the progress of task.

If MySQL HeatWave GenAI detects files with different file formats in a single load, it creates a separate table for every format it finds. The table name for each format is the specified or default table name followed by the format. For example, `demo_embeddings_par_pdf` is the name of the table that contains PDF files.

5. After the task is completed, verify that embeddings are loaded in the vector store table:

```
mysql> SELECT COUNT(*) FROM VectorStoreTableName;
```

For example:

```
mysql> SELECT COUNT(*) FROM demo_embeddings_par;
```

If you see a numerical value in the output, your embeddings are successfully loaded in the vector store table.

6. To view the details of the vector store table, use the following statement:

```
mysql> DESCRIBE demo_embeddings_par;
+-------------------+---------------+------+-----+---------+-------+
| Field             | Type          | Null | Key | Default | Extra |
+-------------------+---------------+------+-----+---------+-------+
| document_name     | varchar(1024) | NO   |     | NULL    |       |
| metadata          | json          | NO   |     | NULL    |       |
| document_id       | int unsigned  | NO   | PRI | NULL    |       |
| segment_number    | int unsigned  | NO   | PRI | NULL    |       |
| segment           | varchar(1024) | NO   |     | NULL    |       |
| segment_embedding | vector(384)   | NO   |     | NULL    |       |
| segment_metadata  | json          | NO   |     | NULL    |       |
+-------------------+---------------+------+-----+---------+-------+
```

Ingest Files Using Auto Parallel Load

The `HEATWAVE_LOAD` routine creates and loads vector embeddings into the vector store using auto parallel load.

To ingest files using the `HEATWAVE_LOAD` routine, perform the following steps:

1. To create a vector store table, use a new or existing database:

```
mysql> USE DBName;
```

Replace *DBName* with the name you want for the new database.

For example:

```
mysql> USE demo_db;
```

2. To ingest the file from Object Storage and create vector embeddings in a new vector store table, set the `@dl_tables` variable:

```
SET @dl_tables = '[{
        "db_name": "DBName",
        "tables": [{
                "table_name": "VectorStoreTableName",
                "engine_attribute": {
                    "dialect": {
                        "format": "FileFormat",
```

```
                        "language": "Language",
                        "ocr": OCRBoolValue,
                        "document_parser_model": "DocumentParserModel",
                       "embed_model_id": "EmbeddingModel"
                    },
                    "file": [{"par": "PAR"}],
                    "chunking": {
                        "split_by": "SplitBy",
                        "max": Max,
                        "by": "By",
                        "truncate": Truncate,
                        "overlap": Overlap
                    }
                }

            }]
    }]';
```

Replace the following:

- *`DBName`*: the database name.
- *`VectorStoreTableName`*: the name you want for the vector store table.
- *`FileFormat`*: the formats of the files to be ingested into the vector store table. The supported file formats are `html`, `pdf`, `ppt`, `pptx` `txt`, `doc`, and `docx`. To ingest multiple files with different unstructured data file formats into the vector store table in a single load, replace *`FileFormat`* with `auto_unstructured`.
- *`Language`*: the two-letter `ISO 639-1` code for the language you want to use. Default language is `en`, which is English. To view the list of supported languages, see Languages.

  The `language` parameter is supported as of MySQL 9.0.1-u1.
- *`OCRBoolValue`*: set to `true` to enable Optical Character Recognition (OCR) or `false` to disable OCR.

  As of MySQL 9.1.2, OCR is enabled by default. In MySQL 9.1.0 and 9.1.1, to enable OCR, set the `ocr` parameter to `true`.
- *`DocumentParserModel`*: ID of an LLM that you want to use for parsing the files to be ingested. The specified LLM must be capable of processing both textual and visual inputs. This helps improve parsing of PDF documents that have a complex structure, or contain complex tables and images. By default, this is an empty string. For possible values, to learn how to view the list of Oracle Cloud Infrastructure (OCI) GenAI Service LLMs available in MySQL HeatWave, see OCI Generative AI Service LLMs.

  To use a document parsing model, OCR must be enabled. Therefore, if you are specifying a document parsing model, ensure that the `ocr` parameter is not set to `false`.

  Note

  This parameter is supported for PDF files only.

  The `document_parser_model` parameter is available as of MySQL 9.4.2.
- *`EmbeddingModel`*: ID of the embedding model you want to use. Default value is `multilingual-e5-small`.

  To view the lists of available embedding models, see MySQL HeatWave In-Database Embedding Models and OCI Generative AI Service Embedding Models. OCI Generative AI Service models are supported as of MySQL 9.4.1.

- *PAR*: the pre-authenticated request (PAR) detail of the bucket, folder, or file that you want to use to set up the vector store.

  To learn how to create PAR for your Object Storage bucket, see Creating a PAR request in Object Storage.

  Note

  If you are creating a PAR for a folder or Object Storage bucket, then select *Enable Object Listing* to enable object listing in the *Create Pre-Authenticated Request* dialog while creating the PAR.

- As of MySQL 9.3.2, you can optionally set the following `chunking` parameters for customized text segmentation during vector store creation:

  - *`SplitBy`*: method to use for splitting the text into segments. It can be one of the following:

    - `page`: for text segmentation based on the pagination available in the document. This segmentation method is supported for PDF and PPT documents only. If used for other documents, the routine falls back to the `document` text segmentation method for unsupported documents.

    - `paragraph`: for text segmentation based on the paragraphs identified in the document. Wherein, a paragraph is a piece of text separated from another piece of text using `\n\n` characters.

    - `sentence`: for text segmentation based on the sentences identified in the document. Wherein, a sentence is a sequence of words that is separated from another sequence of words using a punctuation that marks the end of a sentence: `.`, `!`, or `?`. For PDF files, when OCR is enabled, any sentence that spills into the next page is considered as a two separate sentences.

    - `document`: for putting an entire document into one text segment. In case the `max` parameter is not set, you cannot overlap across documents.

    - `recursive`: for using the default text segmentation method used while creating vector store tables in previous versions of MySQL. This method provides backward compatibility with vector store tables created in previous versions of MySQL.

    Default value is `recursive`. As of MySQL 9.4.2, if the `document_parser_model` parameter is set, the default value of the `split_by` parameter changes to `page`.

  - *`Max`*: maximum number of characters or words to be included in each segment. If left unspecified, there is no maximum size enforcement on the chunks of text. For words, this value can be up to `100000` and for characters, this value can be up to `1000000`. This is unspecified by default.

  - *`Overlap`*: maximum number of characters or words to overlap between segments on each side. For words, this value can be up to `50000` and for characters, this value can be up to `500000`. Default value is `0`.

  - *`By`*: the unit to use for defining the maximum and overlap text segment limits using the `max` and `overlap` parameter. It can be set to `characters` or `words`. Default value is `characters`.

  - *`Truncate`*: set to `true` to enable truncation of text segments that are too large for the embedding model to handle without truncation. However, when tructation is enabled, the end

of the text segment is trimmed out, and you might lose the information from the part of the text segment that is trimmed out.

Set to `false` to disable truncation and instead throw an error when a text segment that is too large for the embeeding model is found. In this case, the entire load fails and exits with an error. You can adjust the text segmentation method used to ensure that the size of the text segments is within the segment size limit of the embedding model.

Default value is `true`.

For example:

```
mysql> SET @dl_tables = '[{
        "db_name": "demo_db",
        "tables": [{
                "table_name": "demo_embeddings_apl",
                "engine_attribute": {
                    "dialect": {
                        "format": "pdf",
                       "language": "en",
                       "embed_model_id": "multilingual-e5-small"
                    },
                    "file": [{"par": "https://demo.objectstorage.demo-region.oci.customer-oci.com/p/dem
                    "chunking": {
                        "split_by": "recursive"
                    }
                }
            }]
    }]';
```

3. To prepare for loading the vector embeddings into the MySQL HeatWave system, set the `@options` variable:

```
mysql> SET @options = JSON_OBJECT('mode', 'normal');
```

4. To load the vector embeddings into MySQL HeatWave, use the `HEATWAVE_LOAD` routine:

```
mysql> CALL sys.HEATWAVE_LOAD(CAST(@dl_tables AS JSON), @options);
```

This creates and stores the vector embeddings in the specified vector store table.

5. Verify that embeddings are loaded in the vector store table:

```
mysql> SELECT COUNT(*) FROM VectorStoreTableName;
```

For example:

```
mysql> SELECT COUNT(*) FROM demo_embeddings_apl;
```

If you see a numerical value in the output, your embeddings are successfully loaded in the table.

6. To view the details of the vector store table, use the following statement:

```
mysql> DESCRIBE demo_embeddings_apl;
+------------------+--------------+------+-----+---------+-------+
| Field            | Type         | Null | Key | Default | Extra |
+------------------+--------------+------+-----+---------+-------+
| document_name    | varchar(1024) | NO   |     | NULL    |       |
| metadata         | json         | NO   |     | NULL    |       |
| document_id      | int unsigned | NO   | PRI | NULL    |       |
| segment_number   | int unsigned | NO   | PRI | NULL    |       |
| segment          | varchar(1024) | NO   |     | NULL    |       |
| segment_embedding | vector(384)  | NO   |     | NULL    |       |
| segment_metadata | json         | NO   |     | NULL    |       |
+------------------+--------------+------+-----+---------+-------+
```

Clean Up

To avoid being billed for the resources that you created for this topic, ensure that you delete them:

- Delete the database:

```
mysql> DROP DATABASE demo_db;
```

- Delete `demo_bucket`. For more information, see Deleting the Bucket.

What's Next

- Learn how to Update the Vector Store.
- Learn how to Perform Vector Search With Retrieval-Augmented Generation.
- Learn how to Generate Text Summaries for Documents Ingested Using Auto Parallel Load.
- Learn how to Start a Conversational Chat.

7.7.3 Update a Vector Store

To keep up with the changes and updates in the documents in your Object Storage, you must update the vector embeddings loaded in the vector store table on a regular basis. This ensures that the responses generated by MySQL HeatWave GenAI are up-to-date. And, it deletes the embeddings that are no longer useful.

The following sections in this topic describe how to update a vector store:

- Before You Begin
- Load Data Incrementally into the Vector Store Table
- Delete and Recreate the Vector Store
- What's Next

Before You Begin

Complete the steps to set up a vector store.

Load Data Incrementally into the Vector Store Table

Incremental load refreshes the vector embeddings for documents that have already been ingested into the vector store, and for the new documents that are available in the bucket, it creates and loads the vector embeddings into the vector store. For documents that have been deleted from the bucket, incremental load deletes the vector embeddings from the vector store.

Incremental loading for vector store tables is supported in MySQL 9.0.1-u1.

The examples in this section use the vector store table `demo_embeddings` created in Using the Uniform Resource Identifier with Asynchronous Load. However, you can update the examples to use a different vector store table name.

To update the embeddings in the vector store table using incremental load, perform the following steps:

1. Check that the vector embeddings are loaded in the vector store table you want to update:

```
mysql> SELECT COUNT(*) FROM VectorStoreTableName;
```

Replace *`VectorStoreTableName`* with the name of the vector store table you want to update.

For example:

```
mysql> SELECT COUNT(*) FROM demo_embeddings;
```

If you see a numerical value in the output, the embeddings are loaded in the table.

2. To specify vector store table to update, set the `@dl_tables` variable:

```
mysql> SET @dl_tables = '[
      {
        "db_name": "DBName",
        "tables": [
        {
          "table_name": "VectorStoreTableName"
        }]
      }
    ]';
```

Replace the following:

- *`DBName`*: the name of database that contains the vector store table.
- *`VectorStoreTableName`*: the vector store table name.

For example:

```
mysql> SET @dl_tables = '[
      {
        "db_name": "demo_db",
        "tables": [
        {
          "table_name": "demo_embeddings"
        }]
      }
    ]';
```

3. To enable incremental loading, set the `refresh_external_tables` parameter in the `@options` variable:

```
mysql> SET @options = JSON_OBJECT("mode", "normal", "refresh_external_tables", true);
```

4. To load the new and updated vector embeddings into the vector store, use the `HEATWAVE_LOAD` routine:

```
mysql> CALL sys.HEATWAVE_LOAD(CAST(@dl_tables AS JSON), @options);
```

This updates the vector store table with the new and updated embeddings.

5. Verify that the embeddings are updated in the vector store table:

```
mysql> SELECT COUNT(*) FROM VectorStoreTableName;
```

For example:

```
mysql> SELECT COUNT(*) FROM demo_embeddings;
```

If you see a numerical value in the output which is different than the one you saw in step 1, then the vector store table is successfully updated. In some cases, this value might not change even though the vector store table is successfully refreshed.

Delete and Recreate the Vector Store

To delete and recreate the vector store table and vector embeddings, perform the following steps:

1. Delete the vector store table:

```
mysql> DROP TABLE VectorStoreTableName;
```

Replace *VectorStoreTableName* with the vector store table name.

2. To create new embeddings for the updated documents, repeat the steps to set up a vector store.

What's Next

- Learn how to Generate Vector Embeddings.
- Learn how to Perform Vector Search With Retrieval-Augmented Generation.

7.8 Generate Vector Embeddings

This section describes how to generate vector embeddings using the `ML_EMBED_ROW` MySQL HeatWave GenAI routine. Vector embeddings are a numerical representation of the text that capture the semantics of the data and relationships to other data. You can pass the text string in the routine manually or use data from tables in your database. To embed multiple rows of text stored in a table in a single run, you can even run a batch query.

Using this method, you can create vector embedding tables that you can use to perform similarity searches using the `DISTANCE()` function, without setting up a vector store.

Note

This method does not support embedding unstructured data. To learn how to create vector embeddings for unstructured data, see Section 7.7, “Set Up a Vector Store”.

This topic contains the following sections:

- Before You Begin
- Generate a Vector Embedding for Specified Text
- Run Batch Queries
- What's Next

Before You Begin

- Review the MySQL HeatWave GenAI requirements and privileges.
- If you want to use the OCI Generative AI Service Embedding Models, complete the steps to authenticate OCI Generative AI Service.
- Connect to your MySQL HeatWave DB System.
- To Run Batch Queries, add the text that you want to embed to a column in a new or existing table.

Generate a Vector Embedding for Specified Text

To generate a vector embedding using MySQL HeatWave GenAI, perform the following steps:

1. To define the text that you want to encode, set the `@text` variable:

```
mysql> SET @text="TextToEncode";
```

Replace *TextToEncode* with the text that you want to encode. For example:

```
mysql> SET @text="MySQL HeatWave GenAI lets you communicate with unstructured data using natural-lan
```

2. To generate a vector embedding for the specified text, pass the text to the embedding model using the `ML_EMBED_ROW` routine:

```
mysql> SELECT sys.ML_EMBED_ROW(@text, JSON_OBJECT("model_id", "EmbeddingModel")) into @text_embedding;
```

Replace *`EmbeddingModel`* with ID of the embedding model you want to use. To view the lists of available embedding models, see MySQL HeatWave In-Database Embedding Models and OCI Generative AI Service Embedding Models.

For example:

```
mysql> SELECT sys.ML_EMBED_ROW(@text, JSON_OBJECT("model_id", "all_minilm_l12_v2")) into @text_embeddin
```

The routine returns a `VECTOR`, and this commad stores it in the `@text_embedding` variable.

3. Print the vector embedding stored in the `@text_embedding` variable:

```
mysql> SELECT @text_embedding;
```

The output, which is a binary representation of the specified text, looks similar to the following:

```
-----------------------------------------------------+
| 0xEB0FE93C21737D3C4ED2F0BC6DCC06BD0668DBBB15D1ABBBDF3E03BD09DC21BC229512BD06B602BD4824F6BCFFEF
70BDFFB53C3D98A2BB3C02E0D63DD3C79BBC8A5B45BD29B32F3C63040DBD17AE91BDECEEED3C3B8A003D05E03CBB42B2
203D9F41643D404E4BBDEA4B93BBACED3D3DA0C2C43C8D2B55BCFED9103E280EA93D8424503CEE8A143D7A9BD03C0553
AFBD203443BD9276E23C024D15BDE8A6C83D8FD2E9BC401DA33AD661AFBDCA06633CEBB1823D11335C3D14B292BDE380
28BB335F273B55BA053D3472BFBDF60A64BD4B91B3BC6039BE3D37A5763DA85A863D4D08363B3F1E6DBD732F74BDA3BA
0B3C0B436BBC4E4081BDBD4866BDB4F1AB3C9E0692BCAE0160BB6D782D3DD321B83DC521813DE090D7BD8A96EEBC334F
FDBCD0F146BD17217D3DAD8A333C3B427A3DE93EDC3C673712BDE8B21FBC52011A3A3A058A3D0BC534BD8A19AB3B8C1B
493C731CB03BDC0894BD6E7180BD44AC2DBC5131B03C108D8B3D61E13FBCB2B69A3C10423B3DFDAC73BD33883A3DEB35
7E3D3C27363D34599BBD5D9A523D63441D3D95F8EFBC7D74CE3BD420AEBD5A7B313CB1A6BDBCEAFAA53C07A01EBD02FF
373D8912B73C4E16D5BD00EC99BD50A0ED3B121129BB44AB1D3D56CA8A3D2BF183BD500A653C37F5B1BB2EEA3DBD443F
B13DE5FD6E3C0ACC27BDF4F724BD73EB193D32A2773D37FCA8BDA279D0BA13730EBC94238A3DBD72D13DC3072B3D0FF8
F33A8404B53DFFC7983D79FD213DB5A0E7BBDBE31A3D7C7A8DBDFDA55CBA99FEA73D28E9F1BC87C1AD3D7934563BD80E
4EBD51ED65BCE5199DBC122A08BD7B35A43B344EADBD07E41C3D028CA13D6385933CB2A623BDA866093DE4BE843D48F7
72B8A9DA973D152E733D55FF363D56AAC33C0F18703D1F39043D45AB0D3CA7992C3C9053F3BDFE1718BEB199F43C579F
8F3D7489B1BDCE51B6BDA864CDBC8BC8B13D681E83BD7EFB0ABD01A5E8BD7B5949BDFAEA713DAADBE7BC417FACBD5050
A33D8BBC263C560A3BBD96AC13BD4E5D19BC3691C13D5F78FEBC383990BDED5E65BD12AD23BD139B883CB678763B4A94
3F3B3CF6DB3CC0FF0B3DC45AA13C00D893BCCC560A3D2EC4EBBABF3F10BD8FE40F3C1FEE8EBD80829B3C10F186BDAA0A
1E3D7245EE3CDFB41BBDC0BD08BC209B813D337860BD7D5621BDBD5850BCDEB93B3DD6ED30BD1E526DBBF98F703DB288
4ABC35A25B3C9895213C6040EEBC8DA4473C6690083EE09C2EB77D972FBD60B7180A69C488BCAE6529BDF98CABBD175D
CA3D85BA56BDE540C73CE677F43CB382C03D4C79A63D10E2163CF3EB3ABDCED222BCFF398BBC9E268BBD669E93BD68A7
383CF162293D4E138C3CB3E0C13DB96E243D7DD984BDD3C3ABBC818188BD7DA8593B679E623DADE31D3D01DED53C3927
7ABD2460B4BC4ABFDEBCA7224EBD5F9C233AB77405BC8B532DBDEE7F15BCFD05C43D1D5E953D1FD8B3BCF1EBED3C1BAC
3BBCC9FF2B3D097588BA2F14FB3B5BB6383DF85C71BD7A877A3D2542C3BD53824E3D576B9FBD5C247BBD9AD0C03DEB77
F7BCA5804B3DCFF8A7BCBAFF893D6EDBACBDBB20A83C24884BBDDF61C0BD527A92BD4822583D99687EBDFD682DBDE80E
65BD717093BC46A3993D6A4F143E15EE8FBC2A81A7BDF811EFBCFBCF883D3F739CBC2BC0863DFBBADDBCC72B823C9B9C
E2BC631448BD203017BD853B94BD42BC723D4A1807BBB888753C99A10B3D5390613CAF9E623DC7F5A5BDF4A1653CA17E
F6BBBFAC35BC17394BBB3DF39A3D7ACCD83D0DEACCBDEF29C3BC2814633DBD40870BC6CFDE3C759CE9BC754AD03CA084
9D3D64CE9B3C5569293CD36D57BDE9A267BA6CFA503D00BB36BDAE16833C01188DBDA0737B3DDB913C3C0FD08FBD8C9D
F6BC0D2CE23CFE0771BCC4F6583D9E730EBD6E8CA4BCBECB8C3DE1F09B3B3D4034BBE53806BD92AE80BCF2F8D7BC2385
493C679C493DC2324CBD266765BC4FC494BC79364C3D8AB7303DE0E4B8BCBDEF903DCE3DD6BC220727BC589FC2BC22A9
4DBDAA82983BC78C8F3DCB6AC2BA31089DBCEB49FC3C2510EC3B3F05083D90DDDBBDA1E15ABD80362E3D9936A3BDF401
D03C6CDB893DF3684ABDFB2DB7BB76ED95BD03A3FE3C2371B5BDBD5F7EBB1AACF7BB156BE53C1E8A9BBC440E2CBDFF1B
99BD |
```

Run Batch Queries

To encode multiple rows of text strings stored in a table column, in parallel, use the `ML_EMBED_TABLE` routine. This method is faster than running the `ML_EMBED_ROW` routine multiple times.

Note

In versions older than MySQL 9.2.1, to alter an existing table or create a new table, MySQL requires you to set the sql-require-primary-key system variable to `0`.

The `ML_EMBED_TABLE` routine is available in MySQL 9.0.1-u1.

To run the steps in this section, create a new database `demo_db` and table `input_table`:

```
mysql> CREATE DATABASE demo_db;
mysql> USE demo_db;
mysql> CREATE TABLE input_table (id INT AUTO_INCREMENT, Input TEXT, primary key (id));
mysql> INSERT INTO input_table (Input) VALUES('Describe what is MySQL in 50 words.');
mysql> INSERT INTO input_table (Input) VALUES('Describe Artificial Intelligence in 50 words.');
mysql> INSERT INTO input_table (Input) VALUES('Describe Machine Learning in 50 words.');
```

To run batch queries using `ML_EMBED_TABLE`, perform the following steps:

1. Call the `ML_EMBED_TABLE` routine:

   ```
   mysql> CALL sys.ML_EMBED_TABLE("InputDBName.InputTableName.InputColumn", "OutputDBName.OutputTableNa
   JSON_OBJECT("model_id", "EmbeddingModel"));
   ```

   Replace the following:

   - *`InputDBName`*: the name of the database that contains the table column where your input queries are stored.
   - *`InputTableName`*: the name of the table that contains the column where your input queries are stored.
   - *`InputColumn`*: the name of the column that contains input queries.
   - *`OutputDBName`*: the name of the database that contains the table where you want to store the generated outputs. This can be the same as the input database.
   - *`OutputTableName`*: the name of the table where you want to create a new column to store the generated outputs. This can be the same as the input table. If the specified table doesn't exist, a new table is created.
   - *`OutputColumn`*: the name for the new column where you want to store the output generated for the input queries.
   - *`EmbeddingModel`*: ID of the embedding model to use. To view the lists of available embedding models, see MySQL HeatWave In-Database Embedding Models and OCI Generative AI Service Embedding Models.

   For example:

   ```
   mysql> CALL sys.ML_EMBED_TABLE("demo_db.input_table.Input", "demo_db.output_table.Output",
   JSON_OBJECT("model_id", "all_minilm_l12_v2"));
   ```

2. View the contents of the output table:

   ```
   mysql> SELECT * FROM output_table;
   +----+-------------------------------------------------------------------------------------------------
   | id | Output
   +----+-------------------------------------------------------------------------------------------------
   |  1 | 0xFF50873DB76340BD6EE860BDDFDB8C3BF8110ABE7569F83B7D91B63D8F5C003D15F996BDAAB8883D702CB4BD322
   |  2 | 0xE4C75E3DA70A91BD0E3A96BB1E29D7BA6B3AD1BD0256763C8A248F3DE7A85B3D758165BD277FCD3DB6F854BDD7E
   |  3 | 0xA8CAE33C4B9EBFBD3AC704BD8F6DBDBC4B24B4BD9747723DF992273DC8ED8F3CC4B384BDB800A53DF07B0ABD181
   +----+-------------------------------------------------------------------------------------------------
   ```

   As of MySQL 9.3.0, the output table generated using the `ML_EMBED_TABLE` routine contains an additional column called `details` for error reporting. In case the routine fails to generate output for specific rows, details of the errors encountered and default values used are added for the rows in this additional column.

   ```
   mysql> DESCRIBE output_table;
   +---------+--------------+------+-----+---------+-------+
   | Field   | Type         | Null | Key | Default | Extra |
   +---------+--------------+------+-----+---------+-------+
   | id      | int          | NO   | PRI | 0       |       |
   | Output  | vector(2048) | YES  |     | NULL    |       |
   | details | json         | YES  |     | NULL    |       |
   ```

```
+---------+-------------+------+-----+---------+-------+
```

As of MySQL 9.2.1, to specify the embedding model used to generate the vector embeddings, the routine adds the following comment for the VECTOR column in the output table:

```
'GENAI_OPTIONS=EMBED_MODEL_ID=EmbeddingModelID'
```

For example:

```
mysql> SHOW CREATE TABLE output_table;
+--------------+-----------------------------------------------------------------------------------------------------
| Table        | Create Table
+--------------+-----------------------------------------------------------------------------------------------------
| output_table | CREATE TABLE `output_table` (
  `id` int NOT NULL DEFAULT '0',
  `Output` vector(2048) DEFAULT NULL COMMENT 'GENAI_OPTIONS=EMBED_MODEL_ID=minilm',
  `details` json DEFAULT NULL,
  PRIMARY KEY (`id`)
) ENGINE=InnoDB DEFAULT CHARSET=utf8mb4 COLLATE=utf8mb4_0900_ai_ci |
+--------------+-----------------------------------------------------------------------------------------------------
```

This lets you use tables generated using this routine for context retrieval while running retrieval-augmented generation (RAG) as well as MySQL HeatWave Chat.

3. If you created a new database for testing the steps in this section, ensure that you delete the database to avoid being billed for it:

```
mysql> DROP DATABASE demo_db;
```

What's Next

- Learn how to Use Your Own Embeddings With Retrieval-Augmented Generation.
- Learn how to Start a Conversational Chat.

7.9 Perform Vector Search with RAG

When you enter a query, MySQL HeatWave GenAI performs a vector-based similarity search to retrieve similar content from the vector store and embedding tables available in the DB system. It provides the retrieved content as context to the LLM. This helps the LLM to produce more relevant and accurate results for your query. This process is called as retrieval-augmented generation (RAG).

As of MySQL 9.2.1, you can use both inbuilt vector store tables and tables containing your own vector embeddings for running RAG with vector search.

The topics in this section describe how to perform RAG with vector search.

7.9.1 Run Retrieval-Augmented Generation

The `ML_RAG` routine runs retrieval-augmented generation which aims to generate more accurate responses for your queries.

As of MySQL 9.5.0, you can use hybrid search, which combines semantic search and keyword-based search to effectively retrieve data that requires searching on specific keywords as well semantically similar segments.

As of MySQL 9.2.1, for context retrieval, the `ML_RAG` routine uses the name of the embedding model used to embed the input query to find relevant vector store tables that contain vector embeddings from the same embedding model.

In earlier versions of MySQL, for context retrieval, the `ML_RAG` routine uses the language specified for generating the output and the name of the embedding model used to embed the input query to find relevant vector store tables that contain information in the same language and vector embeddings from the same embedding model.

This topic contains the following sections:

- Before You Begin
- Retrieve Context and Generate Relevant Content
- Retrieve Context Without Generating Content
- Retrieve Context and Generate Content Using Hybrid Search
- Run Batch Queries
- Clean Up
- What's Next

Before You Begin

- Complete the steps to set up a vector store.

  The examples in this topic use the vector store table `demo_embeddings` created in the section Ingesting Files Using the URI with Asynchronous Load.

- To Run Batch Queries, add the natural-language queries to a column in a new or existing table.

Retrieve Context and Generate Relevant Content

To enter a natural-language query, retrieve the context, and generate results using RAG, perform the following steps:

1. To specify the table for retrieving the vector embeddings to use as context, set the `@options` variable:

   ```
   mysql> SET @options = JSON_OBJECT(
     "vector_store", JSON_ARRAY("DBName.VectorStoreTableName"),
     "model_options", JSON_OBJECT("language", "Language")
   );
   ```

   Replace the following:

   - *`DBName`*: the name of the database that contains the vector store table.
   - *`VectorStoreTableName`*: the name of the vector store table.
   - *`Language`*: the two-letter `ISO 639-1` code for the language you want to use. Default language is `en`, which is English. To view the list of supported languages, see Languages.

     The `language` parameter is supported as of MySQL 9.0.1-u1.

   For example:

   ```
   mysql> SET @options = JSON_OBJECT(
     "vector_store", JSON_ARRAY("demo_db.demo_embeddings"),
     "model_options", JSON_OBJECT("language", "en")
   );
   ```

   To learn more about the available routine options, see ML_RAG Syntax.

2. To define your natural-language query, set the `@query` variable:

   ```
   mysql> SET @query="AddYourQuery";
   ```

   Replace *`AddYourQuery`* with your natural-language query.

   For example:

```
mysql> SET @query="What is AutoML?";
```

3. To retrieve the augmented prompt, use the `ML_RAG` routine:

```
mysql> CALL sys.ML_RAG(@query,@output,@options);
```

4. Print the output:

```
mysql> SELECT JSON_PRETTY(@output);
```

Text-based content that is generated by the LLM in response to your query is printed as output. The output generated by RAG is comprised of the following parts:

- The text section contains the text-based content generated by the LLM as a response for your query.
- For some LLMs, the output might contain license information.
- The citations section shows the segments and documents it referred to as context.

The output looks similar to the following:

```
| {
  "text": " AutoML is a machine learning technique that uses algorithms to automatically generate and
  "license": "Your use of this Llama model is subject to the Llama 3.2 Community License Agreement ava
  "citations": [
    {
      "segment": "| {   \"text\": \" AutoML is a machine learning technique that uses algorithms to a
      "distance": 0.0725,
      "document_name": "https://demo_namespace.objectstorage.demo_region.oci.customer-oci.com/p/demo_
      "segment_number": 2278
    },
    {
      "segment": "| {   \"text\": \" AutoML is a machine learning technique that automates the proces
      "distance": 0.0743,
      "document_name": "https://demo_namespace.objectstorage.demo_region.oci.customer-oci.com/p/demo_
      "segment_number": 2386
    },
    {
      "segment": "| {   \"text\": \" AutoML is a subfield of machine learning that focuses on automat
      "distance": 0.0762,
      "document_name": "https://demo_namespace.objectstorage.demo_region.oci.customer-oci.com/p/demo_
      "segment_number": 2583
    }
  ],
  "vector_store": [
    "`demo_db`.`demo_embeddings`"
  ],
  "retrieval_info": {
    "method": "n_citations",
    "threshold": 0.0743
  }
} |
```

To continue running more queries in the same session, repeat steps 3 to 5.

Retrieve Context Without Generating Content

To enter a natural-language query and retrieve the context without generating a response for the query, perform the following steps:

1. To specify the table for retrieving the vector embeddings and to skip generation of content, set the `@options` variable:

```
mysql> SET @options = JSON_OBJECT("vector_store", JSON_ARRAY("DBName.VectorStoreTableName"), "skip_gene
```

Replace the following:

- *DBName*: the name of the database that contains the vector store table.
- *VectorStoreTableName*: the name of the vector store table.

For example:

```
mysql> SET @options = JSON_OBJECT("vector_store", JSON_ARRAY("demo_db.demo_embeddings"), "skip_gener
```

2. To define your natural-language query, set the `@query` variable:

```
mysql> SET @query="AddYourQuery";
```

Replace *AddYourQuery* with your natural-language query.

For example:

```
mysql> SET @query="What is AutoML?";
```

3. To retrieve the augmented prompt, use the `ML_RAG` routine:

```
mysql> CALL sys.ML_RAG(@query,@output,@options);
```

4. Print the output:

```
mysql> SELECT JSON_PRETTY(@output);
```

Semantically similar text segments used as content for the query and the name of the documents they were found in are printed as output.

The output looks similar to the following:

```
| {
  "citations": [
    {
      "segment": "| {   \"text\": \" AutoML is a machine learning technique that uses algorithms t
      "distance": 0.0725,
      "document_name": "https://demo_namespace.objectstorage.demo_region.oci.customer-oci.com/p/de
      "segment_number": 2278
    },
    {
      "segment": "| {   \"text\": \" AutoML is a machine learning technique that automates the pro
      "distance": 0.0743,
      "document_name": "https://demo_namespace.objectstorage.demo_region.oci.customer-oci.com/p/de
      "segment_number": 2386
    },
    {
      "segment": "| {   \"text\": \" AutoML is a subfield of machine learning that focuses on auto
      "distance": 0.0762,
      "document_name": "https://demo_namespace.objectstorage.demo_region.oci.customer-oci.com/p/de
      "segment_number": 2583
    }
  ],
  "vector_store": [
    "`demo_db`.`demo_embeddings`"
  ],
  "retrieval_info": {
    "method": "n_citations",
    "threshold": 0.0743
  }
} |
```

To continue running more queries in the same session, repeat steps 3 to 5.

Retrieve Context and Generate Content Using Hybrid Search

As of MySQL 9.5.0, you can use both keyword-based search with vector-based semantic search, which helps improve the quality of search results and accuracy of the response generated for your query.

However, vector store tables created using earlier versions of MySQL only support semantic search. To use hybrid or keyword search with vector store tables created in earlier versions of MySQL, you need to reload the tables.

To enter a natural-language query, retrieve the context, and enhance the accuracy of generated results using hybrid search, perform the following steps:

1. To specify the table for retrieving the vector embeddings to use as context, set the `@options` variable:

   ```
   mysql> SET @options = JSON_OBJECT(
     "vector_store", JSON_ARRAY("DBName.VectorStoreTableName"),
     "keyword_search", true
   );
   ```

   Replace the following:

   - *`DBName`*: the name of the database that contains the vector store table.
   - *`VectorStoreTableName`*: the name of the vector store table.

   For example:

   ```
   mysql> SET @options = JSON_OBJECT(
     "vector_store", JSON_ARRAY("demo_db.demo_embeddings"),
     "keyword_search", true
   );
   ```

   To learn more about the available routine options, see ML_RAG Syntax.

2. To define your natural-language query, set the `@query` variable:

   ```
   mysql> SET @query="AddYourQuery";
   ```

   Replace *`AddYourQuery`* with your natural-language query.

   For example:

   ```
   mysql> SET @query="What is AutoML?";
   ```

3. To retrieve the augmented prompt, use the `ML_RAG` routine:

   ```
   mysql> CALL sys.ML_RAG(@query,@output,@options);
   ```

4. Print the output:

   ```
   mysql> SELECT JSON_PRETTY(@output);
   ```

   Text-based content that is generated by the LLM in response to your query is printed as output. The output generated is comprised of the following parts:

   - The text section contains the text-based content generated by the LLM as a response for your query.
   - For some LLMs, the output might contain license information.
   - The citations section shows the segments and documents it referred to as context. Segments with a `distance` score are derived from semantic search, and segments with a `bm25_score` score are derived from keyword search.

   The output looks similar to the following:

   ```
   {
     "text": "AutoML stands for Automated Machine Learning. It's a subfield of machine learning that focus
     "license": "Your use of this Llama model is subject to the Llama 3.2 Community License Agreement avai
     "citations": [
   ```

```
    {
      "segment": "\"segment\": \"| {   \\\"text\\\": \\\" AutoML is a subfield of machine learning t
      "distance": 0.0732,
      "document_name": "https://demo_namespace.objectstorage.demo_region.oci.customer-oci.com/p/demo
      "segment_number": 2278
    },
    {
      "segment": "}, {   \"segment\": \"| {   \\\"text\\\": \\\" AutoML is a subfield of machine lea
      "distance": 0.0738,
      "document_name": "https://demo_namespace.objectstorage.demo_region.oci.customer-oci.com/p/demo
      "segment_number": 2386
    },
    {
      "segment": "| {   \"text\": \" AutoML is a machine learning technique that automates the proce
      "distance": 0.0743,
      "document_name": "https://demo_namespace.objectstorage.demo_region.oci.customer-oci.com/p/demo
      "segment_number": 2583
    },
    {
      "segment": "7.2 HeatWave AutoML Monitoring You can monitor HeatWave AutoML status by querying
      "bm25_score": 15.8061,
      "document_name": "https://demo_namespace.objectstorage.demo_region.oci.customer-oci.com/p/demo
      "segment_number": 3285
    },
    {
      "segment": "| rapid_ml_status | ON             |\n+-----------------+---------------+\n• Quer
      "bm25_score": 15.3699,
      "document_name": "https://demo_namespace.objectstorage.demo_region.oci.customer-oci.com/p/demo
      "segment_number": 3288
    },
    {
      "segment": "3.1.3 HeatWave AutoML Workflow ..............................................
      "bm25_score": 15.0427,
      "document_name": "https://demo_namespace.objectstorage.demo_region.oci.customer-oci.com/p/demo
      "segment_number": 942
    }
  ],
  "vector_store": [
    "`demo_db`.`demo_embeddings`"
  ],
  "retrieval_info": null
} |
```


To continue running more queries in the same session, repeat steps 3 to 5.

Run Batch Queries

To run multiple RAG queries in parallel, use the `ML_RAG_TABLE` routine. This method is faster than running the `ML_RAG` routine multiple times.

In versions older than MySQL 9.2.1, to alter an existing table or create a new table, MySQL requires you to set the sql-require-primary-key system variable to `0`.

The `ML_RAG_TABLE` routine is available in MySQL 9.0.1-u1.

To run the steps in this section, create a new table `input_table` in `demo_db`:

```
mysql> USE demo_db;
mysql> CREATE TABLE input_table (id INT AUTO_INCREMENT, Input TEXT, primary key (id));
mysql> INSERT INTO input_table (Input) VALUES('What is HeatWave Lakehouse?');
mysql> INSERT INTO input_table (Input) VALUES('What is HeatWave AutoML?');
mysql> INSERT INTO input_table (Input) VALUES('What is HeatWave GenAI?');
```

To run batch queries using `ML_RAG_TABLE`, perform the following steps:

1. To specify the table for retrieving the vector embeddings to use as context, set the `@options` variable:

```
mysql> SET @options = JSON_OBJECT(
  "vector_store", JSON_ARRAY("DBName.VectorStoreTableName"),
```

```
  "model_options", JSON_OBJECT("language", "Language")
);
```

Replace the following:

- *DBName*: the name of the database that contains the vector store table.
- *VectorStoreTableName*: the name of the vector store table.
- *Language*: the two-letter ISO 639-1 code for the language you want to use. Default language is en, which is English. To view the list of supported languages, see Languages.

For example:

```
mysql> SET @options = JSON_OBJECT(
  "vector_store", JSON_ARRAY("demo_db.demo_embeddings"),
  "model_options", JSON_OBJECT("language", "en")
);
```

To learn more about the available routine options, see ML_RAG_TABLE Syntax.

2. In the ML_RAG_TABLE routine, specify the table columns containing the input queries and for storing the generated outputs:

```
mysql> CALL sys.ML_RAG_TABLE("InputDBName.InputTableName.InputColumn", "OutputDBName.OutputTableName.Ou
```

Replace the following:

- *InputDBName*: the name of the database that contains the table column where your input queries are stored.
- *InputTableName*: the name of the table that contains the column where your input queries are stored.
- *InputColumn*: the name of the column that contains input queries.
- *OutputDBName*: the name of the database that contains the table where you want to store the generated outputs. This can be the same as the input database.
- *OutputTableName*: the name of the table where you want to create a new column to store the generated outputs. This can be the same as the input table. If the specified table doesn't exist, a new table is created.
- *OutputColumn*: the name for the new column where you want to store the output generated for the input queries.

For example:

```
mysql> CALL sys.ML_RAG_TABLE("demo_db.input_table.Input", "demo_db.output_table.Output", @options);
```

3. View the contents of the output table:

```
mysql> SELECT * FROM output_table\G
*************************** 1. row ***************************
    id: 1
Output: {
    "text": "HeatWave Lakehouse is a feature of the HeatWave platform that enables query processing on
    "error": null,
    "license": "Your use of this Llama model is subject to the Llama 3.2 Community License Agreement ava
    "citations": [
        {
            "segment": "----------------------+ |  1 | {\"text\": \" HeatWave Lakehouse is a feature o
            "distance": 0.0828,
            "document_name": "https://demo_namespace.objectstorage.demo_region.oci.customer-oci.com/p/d
            "segment_number": 2308
        },
```

```
        {
            "segment": "----------------------------------------+ |  1 | {\"text\": \" HeatWave La
            "distance": 0.0863,
            "document_name": "https://demo_namespace.objectstorage.demo_region.oci.customer-oci.com/
            "segment_number": 2415
        },
        {
            "segment": "The Lakehouse feature of HeatWave enables query processing on data in Object
            "distance": 0.1028,
            "document_name": "https://demo_namespace.objectstorage.demo_region.oci.customer-oci.com/
            "segment_number": 2799
        }
    ],
    "vector_store": [
        "`demo_db`.`demo_embeddings`"
    ],
    "retrieval_info": {
        "method": "n_citations",
        "threshold": 0.1028
    }
}
*************************** 2. row ***************************
    id: 2
Output: {
    "text": "HeatWave AutoML is a feature of MySQL HeatWave that simplifies the use of machine learn
    "error": null,
    "citations": [
        {
            "segment": "|  HeatWave AutoML is a feature of MySQL HeatWave that makes it easy to use
            "distance": 0.0561,
            "document_name": "https://demo_namespace.objectstorage.demo_region.oci.customer-oci.com/
            "segment_number": 2069
        },
        {
            "segment": "HeatWave shapes and scaling, and all HeatWave AutoML makes it easy to use ma
            "distance": 0.0573,
            "document_name": "https://demo_namespace.objectstorage.demo_region.oci.customer-oci.com/
            "segment_number": 2420
        },
        {
            "segment": "HeatWave AutoML makes it easy to use machine learning, whether you are a nov
            "distance": 0.0598,
            "document_name": "https://demo_namespace.objectstorage.demo_region.oci.customer-oci.com/
            "segment_number": 967
        }
    ],
    "vector_store": [
        "`demo_db`.`demo_embeddings`"
    ],
    "retrieval_info": {
        "method": "n_citations",
        "threshold": 0.0598
    }
}
*************************** 3. row ***************************
    id: 3
Output: {
    "text": "HeatWave GenAI is a feature of HeatWave that allows communication with unstructured dat
    "error": null,
    "citations": [
        {
            "segment": "4.1 HeatWave GenAI Overview HeatWave GenAI is a feature of HeatWave that let
            "distance": 0.0521,
            "document_name": "https://demo_namespace.objectstorage.demo_region.oci.customer-oci.com/
            "segment_number": 2008
        },
        {
            "segment": "Chapter 3, HeatWave AutoML. 1.4 HeatWave GenAI The HeatWave GenAI feature of
            "distance": 0.0735,
            "document_name": "https://demo_namespace.objectstorage.demo_region.oci.customer-oci.com/
            "segment_number": 125
        },
```

```
        {
            "segment": "HeatWave Chat also provides a graphical interface integrated with the Visual St
            "distance": 0.0781,
            "document_name": "https://demo_namespace.objectstorage.demo_region.oci.customer-oci.com/p/d
            "segment_number": 2014
        }
    ],
    "vector_store": [
        "`demo_db`.`demo_embeddings`"
    ],
    "retrieval_info": {
        "method": "n_citations",
        "threshold": 0.0781
    }
}
```

As of MySQL 9.3.0, the output table generated using the `ML_RAG_TABLE` routine contains an additional details for error reporting. In case the routine fails to generate output for specific rows, details of the errors encountered and default values used are added for the rows in the output column.

Clean Up

If you created a new database for testing the steps in this topic, ensure that you delete the database to avoid being billed for it:

```
mysql> DROP DATABASE demo_db;
```

What's Next

- Learn how to Use Your Own Embeddings With Retrieval-Augmented Generation.
- Learn how to Start a Conversational Chat.

7.9.2 Use Your Own Embeddings

As of MySQL 9.2.1, MySQL HeatWave GenAI lets you use tables containing your own vector embedding to run retrieval-augemented generation (RAG) with vector search. The `ML_RAG` and `ML_RAG_TABLE` routines let you specify the table column names to use as filters for finding relevant tables for context retrieval.

In addition to the specified column names, the `ML_RAG` and `ML_RAG_TABLE` routines use the name of the embedding model used to embed the input query to find relevant embedding tables for context retrieval.

Following sections in this topic describe how you can use your own embedding table for context retrieval:

- Before You Begin
- Use Embeddings From an Embedding Model Available in MySQL HeatWave
- Use Embeddings From an Embedding Model Not Available in MySQL HeatWave
- Use Your Embedding Table With a Vector Store Table
- Run Batch Queries
- Clean Up
- What's Next

Before You Begin

- Review the MySQL HeatWave GenAI requirements and privileges.
- Connect to your MySQL HeatWave DB System.

- Create a table that satisfies the following criteria:
  - To qualify as a valid embedding table, the table must contain the following columns:
    - A string column containing the text segments.
    - A vector column containing the vector embeddings of the text segments.
  - A comment on the vector column must specify the name of the embedding model used to generate the vector embeddings.

  Following is an example of a valid embedding table that can be used for context retrieval:

```
mysql> CREATE TABLE demo_table (id INT AUTO_INCREMENT,
demo_text TEXT,
string_embedding TEXT,
demo_embedding VECTOR (3) COMMENT 'GENAI_OPTIONS=EMBED_MODEL_ID=demo_embedding_model',
primary key (id));
mysql> INSERT INTO demo_table (demo_text, string_embedding)
VALUES('MySQL is an open-source RDBMS that is widely used for its scalability, reliability, and secur
mysql> INSERT INTO demo_table (demo_text, string_embedding)
VALUES('AI refers to the development of machines that can think and act like humans.', '[0,0,1]');
mysql> INSERT INTO demo_table (demo_text, string_embedding)
VALUES('ML is a subset of AI that uses algorithms and statistical models to improve performance on ta
mysql> UPDATE demo_table SET demo_embedding=STRING_TO_VECTOR(string_embedding);
mysql> ALTER TABLE demo_table DROP COLUMN string_embedding;
```

  To learn how to generate vector embeddings and embedding tables using MySQL HeatWave GenAI, see Generating Vector Embeddings.

- If you want to use an inbuilt vector store table along with your own embedding table, complete the steps to set up the vector store.
- To Run Batch Queries, add the natural-language queries to a column in a new or existing table. To use the name of an embedding model that is not available in MySQL HeatWave for running RAG, also add the vector embeddings of the input queries to a column of the input table.
- To create and store the sample embedding tables required for running the steps in this topic, you can create and use a new database `demo_db`:

```
mysql> CREATE DATABASE demo_db;
mysql> USE demo_db;
```

Use Embeddings From an Embedding Model Available in MySQL HeatWave

To use an embedding table containing vector embeddings from an embedding model that is available in MySQL HeatWave, you can set the `vector_store_columns` parameter to specify the columns and column names used by the `ML_RAG` routine to filter tables for context retrieval. However, since the inbuilt vector store tables only use the predefined column names, if you change a column name used for filtering tables, the inbuilt vector store tables are filtered out and not used for context retrieval.

The example in this section uses the following table:

```
mysql> CREATE TABLE demo_minilm_table (id INT AUTO_INCREMENT, demo_text_column TEXT, primary key (id));
mysql> INSERT INTO demo_minilm_table (demo_text_column)
VALUES('MySQL is an open-source RDBMS that is widely used for its scalability, reliability, and securit
mysql> INSERT INTO demo_minilm_table (demo_text_column)
VALUES('AI refers to the development of machines that can think and act like humans.');
mysql> INSERT INTO demo_minilm_table (demo_text_column)
VALUES('ML is a subset of AI that uses algorithms and statistical models to improve performance on task
mysql> CALL sys.ML_EMBED_TABLE('demo_db.demo_minilm_table.demo_text_column', 'demo_db.demo_minilm_table
JSON_OBJECT('model_id', 'all_minilm_l12_v2'));
```

To run RAG, perform the following steps:

1. To change the column names to use to filter tables for context retrieval, then set the routine options as shown below:

```
mysql> SET @options = JSON_OBJECT(
  "vector_store_columns", JSON_OBJECT("segment", "TextSegmentColumnName", "segment_embedding", "VectorE
  "embed_model_id", "EmbeddingModelName",
  "model_options", JSON_OBJECT("language", "Language")
);
```

Replace the following:

- *`TextSegmentColumnName`*: the name of the embedding table column that contains the text segments in natural language. Default value is `segment`.
- *`VectorEmbeddingColumnName`*: the name of the embedding table column that contains vector embeddings of the natural-language text segments. Default value is `segment_embedding`.
- *`EmbeddingModelName`*: the name of the embedding model to use to generate the vector embeddings for the input query. The routine uses this embedding model name to find relevant tables for context retrieval. Default value is `minilm` if the output language is set to English and `multilingual-e5-small` if the output language is set to a language other than English.

  For possible values, to view the list of available embedding models, see MySQL HeatWave In-Database Embedding Models and OCI Generative AI Service Embedding Models.
- *`Language`*: the two-letter `ISO 639-1` code for the language you want to use for generating the output. The `model_option` option parameter `language` is required only if you want to use a language other than English. Default language is `en`, which is English. To view the list of supported languages, see Languages.

For example:

```
mysql> SET @options = JSON_OBJECT(
  "vector_store_columns", JSON_OBJECT("segment", "demo_text_column", "segment_embedding", "demo_embeddi
  "embed_model_id", "all_minilm_l12_v2", "model_options", JSON_OBJECT("language", "en")
);
```

In this example, all embedding tables containing a string column `demo_text_column` and a vector column `demo_embedding_column`, which contains vector embeddings from `all_minilm_l12_v2`, are used for context retrieval.

Similarly, you can use the `vector_store_columns` parameter to specify the following column names for the routine to filter relevant tables for context retrieval:

- `document_name`: name of a column containing the document names. This column can be of any data type supported by MySQL HeatWave. Default value is `document_name`.
- `document_id`: name of an integer column containing the document IDs. Default value is `document_id`.
- `metadata`: name of a JSON column containing additional table metadata. Default value is `metadata`.
- `segment_number`: name of an integer column containing the segment numbers. Default value is `segment_number`.

Since these are optional columns, if these column values are not set, then the routine does not use these columns to filter tables.

2. To define your natural-language query, set the `@query` variable:

```
SET @query="AddYourQuery";
```

Replace *`AddYourQuery`* with your natural-language query.

For example:

```
mysql> SET @query="What is AutoML?";
```

3. To retrieve the augmented prompt and generate the output, use the `ML_RAG` routine:

```
mysql> CALL sys.ML_RAG(@query,@output,@options);
```

4. Print the output:

```
mysql> SELECT JSON_PRETTY(@output);
```

The output is similar to the following:

```
| {
  "text": " AutoML is a machine learning technique that uses algorithms and
  models to automate the process of selecting the best features, tuning
  hyperparameters, and building predictive models. It allows users to easily
  create accurate machine learning models without requiring extensive expertise
  in data science or programming.",
  "license": "Your use of this Llama model is subject to the Llama 3.2 Community License Agreement a
  "citations": [
    {
      "segment": "AI refers to the development of machines that can think and act like humans.",
      "distance": 0.733,
      "document_name": ""
    },
    {
      "segment": "MySQL is an open-source RDBMS that is widely used for its scalability, reliability
      "distance": 0.8234,
      "document_name": ""
    },
    {
      "segment": "Machine learning is a subset of AI that uses algorithms and statistical models to
      "distance": 0.8282,
      "document_name": ""
    }
  ],
  "vector_store": [
    "`demo_db`.`demo_minilm_table`"
  ],
  "retrieval_info": {
    "method": "n_citations",
    "threshold": 0.8282
  }
} |
```

The `vector_store` section lists the name of the embedding table that is used to retrieve context for generating the output.

Use Embeddings From an Embedding Model Not Available in MySQL HeatWave

To use a table containing vector embeddings from an embedding model that is not available in MySQL HeatWave, the `ML_RAG` routine lets you provide the vector embedding of the input query and the name of the embedding model that you used to embed the input query as well as the vector embeddings stored in your embedding table. When you provide the vector embedding of the input query, the routine skips embedding the query and proceeds with the similarity search, context retrieval, and RAG. However, in this case, you cannot use the inbuilt vector store tables for context retrieval.

The example in this section uses the following table:

```
mysql> CREATE TABLE demo_table (id INT AUTO_INCREMENT,
demo_text TEXT,
```

```
string_embedding TEXT,
demo_embedding VECTOR (3) COMMENT 'GENAI_OPTIONS=EMBED_MODEL_ID=demo_embedding_model',
primary key (id));
mysql> INSERT INTO demo_table (demo_text, string_embedding)
VALUES('MySQL is an open-source RDBMS that is widely used for its scalability, reliability, and security.',
mysql> INSERT INTO demo_table (demo_text, string_embedding)
VALUES('AI refers to the development of machines that can think and act like humans.', '[0,0,1]');
mysql> INSERT INTO demo_table (demo_text, string_embedding)
VALUES('ML is a subset of AI that uses algorithms and statistical models to improve performance on tasks by
mysql> UPDATE demo_table SET demo_embedding=STRING_TO_VECTOR(string_embedding);
mysql> ALTER TABLE demo_table DROP COLUMN string_embedding;
```

To run RAG using a table that contains vector embeddings from an embedding model that is not available in MySQL HeatWave, perform the following steps:

1. Provide the vector embedding of the input query:

   ```
   SET @query_embedding = to_base64(string_to_vector('VectorEmbeddingOfTheQuery'));
   ```

   Replace *`VectorEmbeddingOfTheQuery`* with the vector embedding of your input query.

   For example:

   ```
   SET @query_embedding = to_base64(string_to_vector('[0,1,0]'));
   ```

2. To specify column names for the `ML_RAG` routine to find relevant tables for context retrieval, set the routine options:

   ```
   mysql> SET @options = JSON_OBJECT(
     "vector_store_columns", JSON_OBJECT("segment", "TextSegmentColumnName", "segment_embedding", "VectorE
     "embed_model_id", "EmbeddingModelName",
     "query_embedding", @query_embedding,
     "model_options", JSON_OBJECT("language", "Language")
   );
   ```

   Replace the following:

   - *`TextSegmentColumnName`*: the name of the embedding table column that contains the text segments in natural language.
   - *`VectorEmbeddingColumnName`*: the name of the embedding table column that contains vector embeddings of the natural-language text segments.
   - *`EmbeddingModelName`*: the name of the embedding model that you used to generate the vector embeddings for the input query and embedding tables.
   - *`Language`*: the two-letter `ISO 639-1` code for the language you want to use for generating the output. The `model_option` option parameter `language` is required only if you want to use a language other than English. Default language is `en`, which is English. To view the list of supported languages, see Languages.

   For example:

   ```
   mysql> SET @options = JSON_OBJECT(
     "vector_store_columns", JSON_OBJECT("segment", "demo_text", "segment_embedding", "demo_embedding"),
     "embed_model_id", "demo_embedding_model",
     "query_embedding", @query_embedding,
     "model_options", JSON_OBJECT("language", "en")
   );
   ```

   In this example, embedding tables containing a string column `demo_text` and a vector column `demo_embeddings` which contains vector embeddings from `demo_embedding_model` are used for context retrieval.

   Similarly, you can use the `vector_store_columns` parameter to specify the following column names for the routine to filter relevant tables for context retrieval:

- `document_name`: name of a column containing the document names. This column can be of any data type supported by MySQL HeatWave.
- `document_id`: name of an integer column containing the document IDs.
- `metadata`: name of a JSON column containing additional table metadata.
- `segment_number`: name of an integer column containing the segment numbers.

Since these are optional columns, if these column values are not set, then the routine does not use these columns to filter tables.

3. To define your natural-language query, set the `@query` variable:

```
SET @query="AddYourQuery";
```

Replace *`AddYourQuery`* with your natural-language query.

For example:

```
mysql> SET @query="What is AutoML?";
```

4. To retrieve the augmented prompt, use the `ML_RAG` routine:

```
mysql> CALL sys.ML_RAG(@query,@output,@options);
```

5. Print the output:

```
mysql> SELECT JSON_PRETTY(@output);
```

The output is similar to the following:

```
| {
  "text": " AutoML is a subfield of machine learning that focuses on automating
  the process of building and training machine learning models. It uses techniques
  such as algorithm selection, hyperparameter tuning, and model ensembling to
  automatically generate high-performing models with minimal human intervention.
  AutoML has the potential to democratize access to machine learning by making it easier
  for non-experts to build and deploy models.",
  "license": "Your use of this Llama model is subject to the Llama 3.2 Community License Agreement a
  "citations": [
    {
      "segment": "MySQL is an open-source RDBMS that is widely used for its scalability, reliability
      "distance": 0.0,
      "document_name": ""
    },
    {
      "segment": "Machine learning is a subset of AI that uses algorithms and statistical models to
      "distance": 0.2929,
      "document_name": ""
    },
    {
      "segment": "AI refers to the development of machines that can think and act like humans.",
      "distance": 1.0,
      "document_name": ""
    }
  ],
  "vector_store": [
    "`demo_db`.`demo_table`"
  ],
  "retrieval_info": {
    "method": "n_citations",
    "threshold": 1.0
  }
```

```
} |
```

The `vector_store` section lists the name of the embedding table that is used to retrieve context for generating the output.

Use Your Embedding Table With a Vector Store Table

By default, the `ML_RAG` routine uses all predefined columns and column names available in the inbuilt vector store table to filter tables for context retrieval. This means that if your embedding table does not contain all columns that are available in an inbuilt vector store table, then your embedding table is filtered out and is not used for context retrieval by the routine.

Therefore, if you want to use an inbuilt vector store table along with your own embedding table for context retrieval, your embedding table must satisfy the following additional requirements:

- Since the inbuilt vector store tables, which are set up using Asynchronous Load or Auto Parallel Load, use predefined column names, the column names in your embedding tables must match the predefined inbuilt vector store table column names as given below:
  - `segment`: name of the mandatory string column containing the text segments.
  - `segment_embedding`: name of the mandatory vector column containing the vector embeddings of the text segments.
  - `document_name`: name of the optional column containing the document names. This column can be of any data type supported by MySQL HeatWave.
  - `document_id`: name of the optional integer column containing the document IDs.
  - `metadata`: name of the optional JSON column containing metadata for the table.
  - `segment_number`: name of the optional integer column containing segment number.
- The vector embeddings in your embedding table must be from the same embedding model as the vector store table.

The example in this section uses the vector store table `demo_embeddings` created in the section Ingesting Files Using the URI with Asynchronous Load with the following table:

```
mysql> CREATE TABLE demo_e5_table (id INT AUTO_INCREMENT, segment TEXT, primary key (id));
mysql> INSERT INTO demo_e5_table (segment)
VALUES('MySQL is an open-source RDBMS that is widely used for its scalability, reliability, and security.')
mysql> INSERT INTO demo_e5_table (segment)
VALUES('AI refers to the development of machines that can think and act like humans.');
mysql> INSERT INTO demo_e5_table (segment)
VALUES('Machine learning is a subset of AI that uses algorithms and statistical models to improve performan
mysql> CALL sys.ML_EMBED_TABLE('demo_db.demo_e5_table.segment', 'demo_db.demo_e5_table.segment_embedding',
JSON_OBJECT('model_id', 'multilingual-e5-small'));
```

To run RAG using an inbuilt vector store table and your embedding table, perform the following steps:

1. Set the routine options:
   - If your embedding table contains all the mandatory and optional columns as the inbuilt vector store table, then set the routine options as shown below:

     ```
     mysql> SET @options = JSON_OBJECT(
       "embed_model_id", "EmbeddingModelName",
       "model_options", JSON_OBJECT("language", "Language"
       )
     );
     ```

     - *`EmbeddingModelName`*: the name of the embedding model to use to generate the vector embeddings for the input query. The routine uses this embedding model name to find relevant tables for context retrieval. As of MySQL 9.3.0, default value is `multilingual-e5-small`.

For possible values, to view the list of available embedding models, see MySQL HeatWave In-Database Embedding Models and OCI Generative AI Service Embedding Models.

- *Language*: the two-letter ISO 639-1 code for the language you want to use for generating the output. The model_option option parameter language is required only if you want to use a language other than English. Default language is en, which is English. To view the list of supported languages, see Languages.

For example:

```
mysql> SET @options = JSON_OBJECT("embed_model_id", "multilingual-e5-small", "model_options", JSON
```

- If your embedding table contains the same mandatory columns as that of an inbuilt vector store table, similar to demo_e5_table, which are:
  - A text column with the name segment.
  - A vector column segment_embedding.

  Then, set the routine options as shown below:

```
mysql> SET @options = JSON_OBJECT(
  "vector_store_columns", JSON_OBJECT("segment", "segment", "segment_embedding", "segment_embeddi
  "embed_model_id", "EmbeddingModelName",
  "model_options", JSON_OBJECT("language", "Language"
  ));
```

  For example:

```
mysql> SET @options = JSON_OBJECT(
  "vector_store_columns", JSON_OBJECT("segment", "segment", "segment_embedding", "segment_embeddi
  "embed_model_id", "multilingual-e5-small",
  "model_options", JSON_OBJECT("language", "en")
);
```

  In this example, both embedding tables and vector store tables that contain a string column segment and a vector column segment_embedding which contains vector embeddings from multilingual-e5-small are used for context retrieval.

2. To define your natural-language query, set the @query variable:

```
SET @query="AddYourQuery";
```

   Replace *AddYourQuery* with your natural-language query.

   For example:

```
mysql> SET @query="What is AutoML?";
```

3. To retrieve the augmented prompt and generate the output, use the ML_RAG routine:

```
mysql> CALL sys.ML_RAG(@query,@output,@options);
```

4. Print the output:

```
mysql> SELECT JSON_PRETTY(@output);
```

   The output is similar to the following:

```
| {
  "text": " AutoML is a machine learning technique that uses algorithms to
  automatically generate and optimize models for specific tasks, without
  the need for manual intervention. It combines the power of machine learning
  with the ease of use of traditional programming tools, allowing users to
  quickly and easily create accurate models for their data.",
```

```
  "license": "Your use of this Llama model is subject to the Llama 3.2 Community License Agreement avai
  "citations": [
    {
      "segment": "| {   \"text\": \" AutoML is a machine learning technique that uses algorithms to aut
      "distance": 0.0725,
      "document_name": ""
    },
    {
      "segment": "| {   \"text\": \" AutoML is a machine learning technique that automates the process
      "distance": 0.0743,
      "document_name": ""
    },
    {
      "segment": "| {   \"text\": \" AutoML is a subfield of machine learning that focuses on automatin
      "distance": 0.0762,
      "document_name": ""
    }
  ],
  "vector_store": [
    "`demo_db`.`demo_embeddings`",
    "`demo_db`.`demo_e5_table`"
  ],
  "retrieval_info": {
    "method": "n_citations",
    "threshold": 0.0762
  }
} |
```

The `vector_store` section lists the names of the vector store table, `demo_embeddings`, and embedding table, `demo_e5_table` that are used to retrieve context for generating the output.

Run Batch Queries

To run multiple RAG queries in parallel, use the `ML_RAG_TABLE` routine. This method is faster than running the `ML_RAG` routine multiple times.

To run the steps in this section, you can use the same sample table `demo_e5_table` as section Use Your Embedding Table With a Vector Store Table, and create the following table to store input queries for batch processing:

```
mysql> CREATE TABLE input_table (id INT AUTO_INCREMENT, Input TEXT, primary key (id));
mysql> INSERT INTO input_table (Input) VALUES('What is HeatWave Lakehouse?');
mysql> INSERT INTO input_table (Input) VALUES('What is HeatWave AutoML?');
mysql> INSERT INTO input_table (Input) VALUES('What is HeatWave GenAI?');
```

To run batch queries using `ML_RAG_TABLE`, perform the following steps:

1. To specify column names for the `ML_RAG_TABLE` routine to find relevant tables for context retrieval, set the routine options:

   ```
   mysql> SET @options = JSON_OBJECT(
     "vector_store_columns", JSON_OBJECT("segment", "TextSegmentColumnName", "segment_embedding", "VectorE
     "embed_model_id", "EmbeddingModelName",
     "model_options", JSON_OBJECT("language", "Language")
   );
   ```

   Replace the following:

   - *`TextSegmentColumnName`*: the name of the embedding table column that contains the text segments in natural language. If multiple tables contain a string column with the same name, they are all used for context retrieval. Default value is `segment`.
   - *`VectorEmbeddingColumnName`*: the name of the embedding table column that contains vector embeddings of the natural-language text segments. If multiple tables contain a vector column with the same name which contain embeddings from the specified embedding model, they are all used for context retrieval. Default value is `segment_embedding`.

- *EmbeddingModelName*: the name of the embedding model to use to generate the vector embeddings for the input query. The routine uses this embedding model name to find tables generated using the same model for context retrieval. Default value is `minilm` if the output language is set to English and `multilingual-e5-small` if the output language is set to a language other than English.

- *Language*: the two-letter `ISO 639-1` code for the language you want to use for generating the output. The `model_option` option parameter `language` is required only if you want to use a language other than English. Default language is `en`, which is English. To view the list of supported languages, see Languages.

For example:

```
mysql> SET @options = JSON_OBJECT(
  "vector_store_columns", JSON_OBJECT("segment", "segment", "segment_embedding", "segment_embedding"
  "embed_model_id", "multilingual-e5-small",
  "model_options", JSON_OBJECT("language", "en")
);
```

In this example, only embedding tables containing a string column `demo_text` and a vector column `demo_embeddings` which contains vector embeddings from `multilingual-e5-small` are used for context retrieval. Since the inbuilt vector store tables use predefined column names, if you change the column names to any value other than the default value, then the vector store tables are filtered out and are not used for context retrieval.

To learn more about the available routine options, see ML_RAG_TABLE Syntax.

Similarly, you can use the `vector_store_columns` parameter to specify the following column names for the routine to filter relevant tables for context retrieval:

- `document_name`: name of a column containing the document names. This column can be of data type supported by MySQL HeatWave. Default value is `document_name`.

- `document_id`: name of an integer column containing the document IDs. Default value is `document_id`.

- `metadata`: name of a JSON column containing additional table metadata. Default value is `metadata`.

- `segment_number`: name of an integer column containing the segment numbers. Default value is `segment_number`.

Since these are optional columns, if these column values are not set, then the routine does not use these columns to filter tables.

If you are using an embedding model that is not available in MySQL HeatWave, then you must also provide the vector embeddings of the input queries. You can specify name of the input table column that contains the vector embeddings of the input queries using the `embed_column` parameter. However, in this case, you cannot use the inbuilt vector store tables for context retrieval.

2. In the ML_RAG_TABLE routine, specify the table columns containing the input queries and for storing the generated outputs:

```
mysql> CALL sys.ML_RAG_TABLE("InputDBName.InputTableName.InputColumn", "OutputDBName.OutputTableName.Ou
```

Replace the following:

- *InputDBName*: the name of the database that contains the table column where your input queries are stored.
- *InputTableName*: the name of the table that contains the column where your input queries are stored.
- *InputColumn*: the name of the column that contains input queries.
- *OutputDBName*: the name of the database that contains the table where you want to store the generated outputs. This can be the same as the input database.
- *OutputTableName*: the name of the table where you want to create a new column to store the generated outputs. This can be the same as the input table. If the specified table doesn't exist, a new table is created.
- *OutputColumn*: the name for the new column where you want to store the output generated for the input queries.

For example:

```
mysql> CALL sys.ML_RAG_TABLE("demo_db.input_table.Input", "demo_db.output_table.Output", @options);
```

View the contents of the output table:

```
mysql> SELECT * FROM output_table\G
*************************** 1. row ***************************
    id: 1
Output: { "text": "HeatWave Lakehouse is a feature of the HeatWave platform that enables query processi
"error": null,
"license": "Your use of this Llama model is subject to the Llama 3.2 Community License Agreement availa
"citations": [
    {
        "segment": "See: Chapter 4, HeatWave GenAI.\n1.5 HeatWave Lakehouse The Lakehouse feature of He
        "distance": 0.1032,
        "document_name": ""
      },
    {
        "segment": "The Lakehouse Engine enables you to create tables which point to external data sour
        "distance": 0.106,
        "document_name": ""
      },
    {
        "segment": "The Lakehouse feature of HeatWave enables query processing on data resident in Obje
        "distance": 0.1063,
        "document_name": ""
      }
  ],
"vector_store": [
    "`demo_db`.`demo_embeddings`",
    "`demo_db`.`demo_e5_table`"
  ],
"retrieval_info": {
    "method": "n_citations",
    "threshold": 0.1063
  }
}
*************************** 2. row ***************************
    id: 2
Output: { "text": "HeatWave AutoML is a feature of MySQL HeatWave that makes it easy to use machine lea
"error": null,
"citations": [
```

```
    {
        "segment": "|  HeatWave AutoML is a feature of MySQL HeatWave that makes it easy to use mach
        "distance": 0.0561,
        "document_name": ""
      },
    {
        "segment": "HeatWave AutoML makes it easy to use machine learning, whether you are a novice
        "distance": 0.0598,
        "document_name": ""
      },
    {
        "segment": "2. HeatWave AutoML analyzes the training data, trains an optimized machine learn
        "distance": 0.0669,
        "document_name": ""
      }
  ],
"vector_store": [
    "`demo_db`.`demo_embeddings`",
    "`demo_db`.`demo_e5_table`"
  ],
"retrieval_info": {
    "method": "n_citations",
    "threshold": 0.0669
  }
}
*************************** 3. row ***************************
    id: 3
Output: { "text": "HeatWave GenAI is a feature of HeatWave that allows users to communicate with uns
"error": null,
"citations": [
    {
        "segment": "4.1 HeatWave GenAI Overview HeatWave GenAI is a feature of HeatWave that lets yo
        "distance": 0.0521,
        "document_name": ""
      },
    {
        "segment": "HeatWave Chat also provides a graphical interface integrated with the Visual Stu
        "distance": 0.0781,
        "document_name": ""
      },
    {
        "segment": "HeatWave GenAI retrieves content from the vector store and provides it as contex
        "distance": 0.0811,
        "document_name": ""
      }
  ],
"vector_store": [
    "`demo_db`.`demo_embeddings`",
    "`demo_db`.`demo_e5_table`"
  ],
"retrieval_info": {
    "method": "n_citations",
    "threshold": 0.0811
  }
}
```

As of MySQL 9.3.0, the output table generated using the `ML_RAG_TABLE` routine contains an additional details for error reporting. In case the routine fails to generate output for specific rows, details of the errors encountered and default values used are added for the rows in the output column.

Clean Up

If you created a new database for testing the steps in this topic, ensure that you delete the database to avoid being billed for it:

```
mysql> DROP DATABASE demo_db;
```

What's Next

Learn about Automatic Vector Index Creation.

7.9.3 Automatic Vector Index Creation

As of MySQL 9.5.0, MySQL HeatWave GenAI automatically creates vector indexes for frequenctly queried `VECTOR` columns in vector store and embedding tables using advanced index structures such as Hierarchical Navigable Small World (HNSW). This helps accelerate similarity search queries that use these columns.

Information about the created vector indexes are stored in the `rpd_vector_indexes` performance schema table.

Control Automatic Vector Index Creation and Usage

- To disable automatic index creation, use the `rapid_auto_vector_index_enabled` variable.
- To change the settings used for automatic vector index creation, use the following variables:
  - `rapid_auto_vector_index_interval_seconds`
  - `rapid_auto_vector_index_server_memory_quota_pct`
  - `rapid_auto_vector_index_skip_quiet_check`
  - `rapid_auto_vector_index_build_timeout_seconds`
- To change settings pertaining to usage of the created vector indexes, use the following variables:
  - `rapid_hnsw_ef_search`
  - `rapid_hnsw_use_adaptive_accuracy_estimation`

What's Next

Learn how to Start a Conversational Chat.

7.10 Start a Conversational Chat

You can use MySQL HeatWave Chat to simulate human-like conversations where you can get responses for multiple queries in the same session. MySQL HeatWave Chat is a conversational agent that utilizes large language models (LLMs) to understand inputs and responds in natural manner. It extends the text generation by using a chat history that lets you ask follow-up questions, and uses the vector search functionality to draw its knowledge from the inbuilt vector store. The responses generated by MySQL HeatWave Chat are quick and secure as all the communication and processing happens within the MySQL HeatWave service.

The sections in this topic describe how to run and manage MySQL HeatWave Chat.

7.10.1 Run MySQL HeatWave Chat

When you run MySQL HeatWave Chat, it automatically loads the default LLM.

By default, MySQL HeatWave Chat searches for an answer to a query across all ingested documents by automatically discovering available vector stores, and returns the answer along with relevant citations. You can limit the scope of search to specific document collections available in certain vector stores or specify documents to include in the search.

As of MySQL 9.2.1, MySQL HeatWave Chat lets you use your own embedding tables for context retrieval. And, it uses only the name of the embedding model used to embed the input query to find relevant tables. In earlier versions of MySQL, MySQL HeatWave Chat uses only inbuilt vector store

tables for context retrieval. And, it uses both the language specified for generating the output and the name of the embedding model used to embed the input query to find relevant tables containing information in the same language and vector embeddings from the sameembedding model.

If you do not have a vector store or an embedding table set up, then MySQL HeatWave Chat uses information available in public data sources to generate a response for your query.

This topic contains the following sections:

- Before You Begin
- Run the Chat
- What's Next

Before You Begin

- Review the MySQL HeatWave GenAI requirements.
- To extend the vector search functionality and ask specific questions about the information available in your proprietary documents stored in the vector store, complete the steps to set up a vector store.

  In this topic, the `HEATWAVE_CHAT` routine uses the vector store table `demo_embeddings` created in the section Ingesting Files Using the URI with Asynchronous Load for context retrieval.
- To use your own embedding table for context retrieval, create a table that satisfies the following criteria:
  - The table must contain the following columns:
    - A string column containing the text segments.
    - A vector column containing the vector embeddings of the text segments.
  - A comment on the vector column must specify the name of the embedding model used to generate the vector embeddings.
  - The vector embeddings in your embedding table must be from an embedding model supported by MySQL HeatWave. To view the list of available embedding models, see MySQL HeatWave In-Database Embedding Models and OCI Generative AI Service Embedding Models.

    Following is an example of a valid embedding table that can be used for context retrieval:

    ```
    mysql> CREATE TABLE demo_table (id INT AUTO_INCREMENT, demo_text TEXT, primary key (id));
    mysql> INSERT INTO demo_table (demo_text) VALUES('What is MySQL?');
    mysql> INSERT INTO demo_table (demo_text) VALUES('What is HeatWave?');
    mysql> INSERT INTO demo_table (demo_text) VALUES('What is HeatWave GenAI?');
    mysql> CALL sys.ML_EMBED_TABLE('demo_schema.demo_table.demo_text', 'demo_schema.demo_table.demo_emb
    JSON_OBJECT('model_id', 'all_minilm_l12_v2'));
    ```

    To learn how to generate vector embeddings and embedding tables using MySQL HeatWave GenAI, see Generating Vector Embeddings.

  If you want to use both inbuilt vector store tables and your own embedding tables for context retrieval, your embedding table must satisfy the following additional requirements:

  - Since the inbuilt vector store tables set up using Asynchronous Load or Auto Parallel Load use predefined column names, the column names in your embedding tables must match the predefined inbuilt vector store table column names as given below:
    - `segment`: name of the mandatory string column containing the text segments.
    - `segment_embedding`: name of the mandatory vector column containing the vector embeddings of the text segments.

- `document_name`: name of the optional column containing the document names. This column can be of any data type supported by MySQL HeatWave.
- `document_id`: name of the optional integer column containing the document IDs.
- `metadata`: name of the optional JSON column containing metadata for the table.
- `segment_number`: name of the optional integer column containing segment number.

- The vector embeddings in your embedding table must be from the same embedding model as the vector store table.

Run the Chat

To run MySQL HeatWave Chat, perform the following steps:

1. To delete previous chat output and state, if any, reset the `@chat_options` variable:

```
mysql> SET @chat_options=NULL;
```

Note

Ensure that you use the name `@chat_options` for the variable. The `HEATWAVE_CHAT` routine reserves this variable for specifying and saving various chat parameter settings.

2. Optionally, set the `@chat_options` variable in the following scenarios:

   - To use a language other than English, set the `language` model option:

```
mysql> SET @chat_options = JSON_OBJECT("model_options", JSON_OBJECT("language", "Language"));
```

     Replace *`Language`* with the two-letter `ISO 639-1` code for the language you want to use. Default language is `en`, which is English. To view the list of supported languages, see Languages.

     The `language` parameter is available as of MySQL 9.0.1-u1.

     For example, to use French set `language` to `fr`:

```
mysql> SET @chat_options = JSON_OBJECT("model_options", JSON_OBJECT("language", "fr"));
```

     This resets the `@chat_options` variable, and specifies the language for the chat.

   - To use your own embedding tables for context retrieval, change the column names used by the `HEATWAVE_CHAT` routine to filter tables by setting the `vector_store_columns` parameter:

```
mysql> SET @chat_options = JSON_OBJECT(
  "vector_store_columns", JSON_OBJECT("segment", "TextSegmentColumnName", "segment_embedding", "Vecto
  "embed_model_id", "EmbeddingModelName"
);
```

     Replace the following:

     - *`TextSegmentColumnName`*: the name of the embedding table column that contains the text segments in natural language. If multiple tables contain a string column with the same name, they are all used for context retrieval. Default value is `segment`.
     - *`VectorEmbeddingColumnName`*: the name of the embedding table column that contains vector embeddings of the natural-language text segments. If multiple tables contain a vector column with the same name which contain embeddings from the specified embedding model, they are all used for context retrieval. Default value is `segment_embedding`.

- *EmbeddingModelName*: the name of the embedding model to use to generate the vector embeddings for the input query. The routine uses this embedding model name to find tables generated using the same model for context retrieval. By default, the routine uses `minilm` if the output language is set to English and `multilingual-e5-small` if the output language is set to a language other than English.

By default, the routine uses all the predefined vector store column names to filter tables for context retrieval.

For example:

```
mysql> SET @chat_options = JSON_OBJECT(
  "vector_store_columns", JSON_OBJECT("segment", "demo_text", "segment_embedding", "demo_embedding
  "embed_model_id", "all_minilm_l12_v2"
);
```

This resets the `@chat_options` variable to specify the column names used for filtering tables for context retrieval. In this example, all embedding tables containing a string column `demo_text` and a vector column `demo_embeddings` which contains vector embeddings from `all_minilm_l12_v2` are used for context retrieval.

However, since the inbuilt vector store tables use predefined column names, if you change a column name used for filtering tables to any value other than the default value, the inbuilt vector store tables are filtered out and are not used for context retrieval.

- As of MySQL 9.3.0, to use vector store tables created in earlier versions of MySQL for context retrieval, set the embedding model name to `all_minilm_l12_v2`:

```
mysql> SET @chat_options = JSON_OBJECT("embed_model_id", "all_minilm_l12_v2");
```

This is because as of MySQL 9.3.0, MySQL HeatWave Chat uses vector store and embedding tables created using `multilingual-e5-small` for context retrieval by default.

3. Then, add your query to MySQL HeatWave Chat by using the `HEATWAVE_CHAT` routine:

```
mysql> CALL sys.HEATWAVE_CHAT("YourQuery");
```

For example:

```
mysql> CALL sys.HEATWAVE_CHAT("What is HeatWave AutoML?");
```

The output looks similar to the following:

```
+-----------------------------------------------------------------------------------------------------
| response
+-----------------------------------------------------------------------------------------------------
| HeatWave AutoML is a feature of MySQL HeatWave that makes it easy to use machine learning, allowin
+-----------------------------------------------------------------------------------------------------
```

Repeat this step to ask follow-up questions using the `HEATWAVE_CHAT` routine:

```
mysql> CALL sys.HEATWAVE_CHAT("What learning algorithms does it use?");
```

The output looks similar to the following:

```
+-----------------------------------------------------------------------------------------------------
| response
+-----------------------------------------------------------------------------------------------------
| HeatWave AutoML uses a variety of machine learning algorithms, including decision trees, random fo
+-----------------------------------------------------------------------------------------------------
```

What's Next

Learn how to View Chat Session Details.

7.10.2 View Chat Session Details

This topic describes how to view a chat session details. It contains the following sections:

- Before You Begin
- View Details
- What's Next

Before You Begin

- Complete the steps to run MySQL HeatWave Chat.

View Details

To view the chat session details, inspect the `@chat_options` variable:

```
mysql> SELECT JSON_PRETTY(@chat_options);
```

The output includes the following details about a chat session:

- *Vector store tables:* in the database which were referenced by MySQL HeatWave Chat.
- *Text segments:* that were retrieved from the vector store and used as context to prepare responses for your queries.
- *Chat history:* which includes both your queries and responses generated by MySQL HeatWave Chat.
- *LLM and Embedding Model IDs:* IDs of the models used by the routine.

The output looks similar to the following:

```
| {
  "tables": [
    {
      "table_name": "`demo_embeddings`",
      "schema_name": "`demo_db`"
    }
  ],
  "response": ""HeatWave AutoML uses a variety of machine learning algorithms, including
  decision trees, random forests, neural networks, and support vector machines (SVMs).
  The specific algorithm used depends on the characteristics of the data being analyzed
  and the goals of the model being created.,
  "documents": [
    {
      "id": "https://objectstorage.Region.oraclecloud.com/n/Namespace/b/BucketName/o/Path/heatwave-en.a4.pd
      "title": "heatwave-en.a4.pdf",
      "segment": "3.1 HeatWave AutoML Features HeatWave AutoML makes it easy to use
      machine learning, whether you are a novice user or an experienced ML practitioner.
      You provide the data, and HeatWave AutoML analyzes the characteristics of the
      data and creates an optimized machine learning model that you can use to
      generate predictions and explanations.",
      "distance": 0.1845
    },
    {
      "id": "https://objectstorage.Region.oraclecloud.com/n/Namespace/b/BucketName/o/Path/heatwave-en.a4.pd
      "title": "heatwave-en.a4.pdf",
      "segment": "The HeatWave AutoML ML_TRAIN routine leverages Oracle AutoML technology
      to automate the process of training a machine learning model. Oracle AutoML replaces
      the laborious and time consuming tasks of the data analyst whose workflow is as
      follows:\n1. Selecting a model from a large number of viable candidate models.\n2.
      For each model, tuning hyperparameters.\n3. Selecting only predictive features
      to speed up the pipeline and reduce over-fitting.\n99",
      "distance": 0.2268
    },
    {
      "id": "https://objectstorage.Region.oraclecloud.com/n/Namespace/b/BucketName/o/Path/heatwave-en.a4.pd
```

```
      "title": "heatwave-en.a4.pdf",
      "segment": "3.1.1 HeatWave AutoML Supervised Learning\nHeatWave AutoML supports supervised
      machine learning. That is, it creates a machine learning model by analyzing a labeled
      dataset to learn patterns that enable it to predict labels based on the features of the
      dataset. For example, this guide uses the Census Income Data Set in its examples, where
      features such as age, education, occupation, country, and so on, are used to predict the
      income of an individual (the label).",
      "distance": 0.2275
    }
  ],
  "chat_history": [
    {
      "user_message": "What is HeatWave AutoML?",
      "chat_query_id": "99471681-387f-11ef-96d7-020017331ed6",
      "chat_bot_message": "HeatWave AutoML is a feature of MySQL HeatWave that makes it easy
      to use machine learning, allowing users to create optimized machine learning models for
      predictions and explanations without having to leave the database."
    },
    {
      "user_message": "What learning algorithms does it use?",
      "chat_query_id": "c59140f5-387f-11ef-96d7-020017331ed6",
      "chat_bot_message": "HeatWave AutoML uses a variety of machine learning algorithms,
      including decision trees, random forests, neural networks, and support vector machines
      (SVMs). The specific algorithm used depends on the characteristics of the data being
      analyzed and the goals of the model being created"
    }
  ],
  "model_options": {
    "model_id": "llama3.2-3b-instruct-v1"
  },
  "embed_model_id": "multilingual-e5-small",
  "request_completed": true
} |
```


What's Next

- Learn how to generate SQL queries from natural-language statements.
- See Section 10.3, “MySQL HeatWave GenAI Routines”.

7.11 Generate SQL Queries From Natural-Language Statements

As of MySQL 9.4.1, MySQL HeatWave GenAI lets you generate SQL queries from natural-language statements, making it easier for you to interact with your databases. This feature collects information on the schemas, tables, and columns that you have access to, and then uses a Large Language Model (LLM) to generate an SQL query for the question pertaining to your data. It also lets you run the generated query and view the result set.

This topic describes how to use the `NL_SQL` routine to generate and run SQL queries from natural-language statements.

Note

This routine can generate and run `SELECT` statements only.

This topic contains the following sections:

- Before You Begin
- Generate and Run an SQL Query
- What's Next

Before You Begin

- Review the MySQL HeatWave GenAI requirements and privileges.

- If you want to use the default LLM, `meta.llama-3.3-70b-instruct`, or any other OCI Generative AI Service LLM, complete the steps to authenticate OCI Generative AI Service.
- Connect to your MySQL HeatWave DB System.
- Load structured data from the DB System or Object Storage.

  The examples in this topic uses a sample database, `airport`. To learn how to download and then load `airportdb` in MySQL HeatWave Cluster, see Section 3.2, “Quickstart: AirportDB Analytics”.

Generate and Run an SQL Query

Perform the following step:

```
mysql> CALL sys.NL_SQL("NaturalLanguageStatement",@output, JSON_OBJECT('schemas',JSON_ARRAY('DBName'),'mode
```

Replace the following:

- *`NaturalLanguageStatement`*: natural-language statement. It can be a question, statement, or query pertaining to your data available in MySQL HeatWave.
- *`DBName`*: database to consider for generating and running the SQL query.
- *`ModelID`*: LLM to use.

For example:

```
mysql> CALL sys.NL_SQL("How many flights are there in total?",@output, JSON_OBJECT('schemas',JSON_ARRAY('ai
```

The output is similar to the following:

```
+-----------------------------------------------------+
| Executing generated SQL statement...                |
+-----------------------------------------------------+
| SELECT COUNT(`flight_id`) FROM `airportdb`.`flight` |
+-----------------------------------------------------+
1 row in set (1 min 39.4060 sec)

+--------------------+
| COUNT(`flight_id`) |
+--------------------+
|             462553 |
+--------------------+
```

View the value stored in the variable `@output`:

```
mysql> SELECT JSON_PRETTY(@output);
+-----------------------------------------------------------------------------------------------------------------
| JSON_PRETTY(@output)
+-----------------------------------------------------------------------------------------------------------------
| {
  "tables": [
    "airportdb.airline",
    "airportdb.airplane",
    "airportdb.airplane_type",
    "airportdb.airport",
    "airportdb.airport_geo",
    "airportdb.airport_reachable",
    "airportdb.booking",
    "airportdb.employee",
    "airportdb.flight",
    "airportdb.flight_log",
    "airportdb.flightschedule",
    "airportdb.passenger",
    "airportdb.passengerdetails",
    "airportdb.weatherdata"
  ],
  "license": "Your use of this Llama model is subject to the Llama 3.1 Community License Agreement availabl
```

```
  "schemas": [
    "airportdb"
  ],
  "model_id": "llama3.1-8b-instruct-v1",
  "sql_query": "SELECT COUNT(`flight_id`) FROM `airportdb`.`flight`",
  "is_sql_valid": 1
} |
+-------------------------------------------------------------------------------------------------
```

The output includes the following details:

- List of tables the routine considered for generating and running the SQL query.
- List of databases the routine considered for generating and running the SQL query.
- Model ID of the LLM used to generate the SQL query.
- The generated SQL query.
- Whether the query is valid.

What's Next

Learn more about the `NL_SQL` routine.

Chapter 8 Performance and Usage

Table of Contents

8.1 MySQL HeatWave Performance Schema Tables 515
8.1.1 The rpd_column_id Table 515
8.1.2 The rpd_columns Table 516
8.1.3 The rpd_exec_stats Table 516
8.1.4 The rpd_mirror Table 517
8.1.5 The rpd_ml_stats Table 517
8.1.6 The rpd_nodes Table 519
8.1.7 The rpd_preload_stats Table 521
8.1.8 The rpd_query_stats Table 521
8.1.9 The rpd_table_id Table 523
8.1.10 The rpd_tables Table 523
8.1.11 The rpd_vector_indexes Table 528
8.2 Option Tracker 528

The MySQL Performance Schema collects statistics on the usage of MySQL HeatWave. Use SQL queries to access this data and check the system status and performance.

The Auto Shape Prediction feature in MySQL HeatWave Autopilot uses MySQL statistics for the workload to assess the suitability of the current shape. Auto Shape Prediction provides prompts to upsize the shape and improve system performance, or to downsize the shape if the system is under-utilized. This feature is available for MySQL HeatWave on AWS only.

Monitor MySQL HeatWave from the MySQL HeatWave Console:

- For MySQL HeatWave on OCI, the MySQL HeatWave Console uses Metrics. See Metrics in the *MySQL HeatWave on OCI Service Guide*.
- MySQL HeatWave on AWS users can monitor MySQL HeatWave on the **Performance** page in the MySQL HeatWave Console.
- For MySQL HeatWave for Azure, select **Metrics** on the details page for the MySQL HeatWave Cluster to access Microsoft Azure Application Insights.

8.1 MySQL HeatWave Performance Schema Tables

MySQL HeatWave Performance Schema tables provide information about MySQL HeatWave nodes, and about tables and columns that are currently loaded in MySQL HeatWave.

Information about MySQL HeatWave nodes is available only when `rapid_bootstrap` mode is `ON`. Information about tables and columns is available only after tables are loaded in the MySQL HeatWave Cluster. See Section 4.2, “Load Data from DB System into MySQL HeatWave Cluster”.

8.1.1 The rpd_column_id Table

The `rpd_column_id` table provides information about columns of tables that are loaded in MySQL HeatWave.

The `rpd_column_id` table has these columns:

- `ID`

  A unique identifier for the column.

- `TABLE_ID`

  The ID of the table to which the column belongs.

- `COLUMN_NAME`

  The column name.

The `rpd_column_id` table is read-only.

8.1.2 The rpd_columns Table

The `rpd_columns` table provides column encoding information for columns of tables loaded in MySQL HeatWave.

The `rpd_columns` table has these columns:

- `TABLE_ID`

  A unique identifier for the table.

- `COLUMN_ID`

  A unique identifier for the table column.

- `NDV`

  The number of distinct values in the column.

- `ENCODING`

  The type of encoding used.

- `DATA_PLACEMENT_INDEX`

  The data placement key index ID associated with the column. Index value ranges from 1 to 16. For information about data placement key index values, see Section 5.8.2, “Data Placement Keys”. NULL indicates that the column is not defined as a data placement key. For a `DATA_PLACEMENT_INDEX` query that identifies columns with data placement keys, see Section 4.5, “Retrieve MySQL HeatWave Metadata”.

- `DICT_SIZE_BYTES`

  The dictionary size per column, in bytes.

The `rpd_columns` table is read-only.

8.1.3 The rpd_exec_stats Table

The `rpd_exec_stats` table stores query execution statistics produced by MySQL HeatWave nodes in `JSON` format. One row of execution statistics is stored for each node that participates in the query. The table stores a maximum of 1000 queries.

For MySQL HeatWave AutoML routines that include multiple sub-queries, such as `ML_TRAIN`, a new record is used for each query.

The `rpd_exec_stats` table has these columns:

- `QUERY_ID`

  The query ID. The counter auto-increments for each MySQL HeatWave and MySQL HeatWave AutoML query.

- `NODE_ID`

  The MySQL HeatWave node ID.

- `EXEC_TEXT`

Query execution statistics. For MySQL HeatWave AutoML, this contains the MySQL HeatWave AutoML routine that the user runs.

8.1.4 The rpd_mirror Table

The `rpd_mirror` table keeps track of all existing tables in the DB System, whose engine is `InnoDB`.

The `rpd_mirror` table has these columns:

- `SCHEMA_NAME`

  The schema name.

- `TABLE_NAME`

  The table name.

- `MYSQL_ACCESS_COUNT`

  Number of times the table has been accessed by the DB System.

- `HEATWAVE_ACCESS_COUNT`

  Number of times the table has been accessed by the MySQL HeatWave Cluster.

- `IMPORTANCE`

  Indicates the importance of each table for automatic load (also known as self load). Tables with higher importance values are loaded to MySQL HeatWave first. For a given table, this value is computed based on its size, as well as the number and duration of queries accessing that table.

- `LAST_QUERIED`

  The timestamp of the last DB System query that referenced the table.

- `LAST_QUERIED_IN_HEATWAVE`

  The timestamp of the last MySQL HeatWave query that referenced the table.

- `STATE`

  The table state: loaded or not loaded.

- `RECOMMENDED_READ_THREADS`

  Value of the `innodb_parallel_read_threads` session variable with which the table was previously loaded.

- `QUERIED_PARTITIONS`

  For partitioned tables, lists the `InnoDB` partitions that have been queried.

8.1.5 The rpd_ml_stats Table

Note

The Performance Schema table described here is available as of MySQL 9.0.0. Support for MySQL HeatWave GenAI routines is available as of MySQL 9.0.1-u1.

The `rpd_ml_stats` table tracks the usage of successful MySQL HeatWave routines. These metrics reset whenever the respective DB system restarts.

The following MySQL HeatWave AutoML routines are tracked:

- `ML_TRAIN`
- `ML_EXPLAIN`
- `ML_PREDICT_ROW`
- `ML_PREDICT_TABLE`
- `ML_EXPLAIN_ROW`
- `ML_EXPLAIN_TABLE`

The following MySQL HeatWave GenAI routines are tracked:

- `ML_GENERATE`
- `ML_EMBED_ROW`

The `rpd_ml_stats` table has these columns:

- `STATUS_NAME`

  Identifies the type of meter tracking usage.

- `STATUS_VALUE`

  Displays metrics for metering. Content is displayed in JSON format.

Metrics in the table are entries as JSON values. The following metrics are used:

- `n_cells`

  The total number of table cells processed by the MySQL HeatWave AutoML routine for all invocations.

- `n_cells_user_excluded`

  The total number of table cells manually excluded for the MySQL HeatWave AutoML routine.

- `n_blob_cells`

  The total number of table BLOB cells processed by the MySQL HeatWave AutoML routine for all invocations.

- `table_size_bytes`

  The total number of bytes of data processed by the MySQL HeatWave AutoML routine for all invocations.

- `blob_size_bytes`

  The total number of bytes of BLOB/TEXT data processed by the MySQL HeatWave AutoML routine for all invocations.

- `model_size_bytes`

  The total number of bytes of data for the MySQL HeatWave AutoML model that is trained. This includes any explainer models. This metric only applies to the `ML_TRAIN` and `ML_EXPLAIN` MySQL HeatWave AutoML routines. All other routines will display NULL values.

- `input_size_bytes`

  The cumulative size in bytes of all input string/document invocations ingested by the MySQL HeatWave GenAI routine.

- `context_size_bytes`

  The size in bytes of the context string referenced when generating the response. This metric only applies to the `ML_GENERATE` MySQL HeatWave GenAI routine since the `ML_EMBED_ROW` routine does not have context. The metric will still appear for `ML_EMBED_ROW`, but will display a value of 0.

- `output_size_bytes`

  The cumulative size in bytes of responses generated by all invocations for the MySQL HeatWave GenAI routine.

- `n_invocations`

  The total number of times the routine has been successfully invoked on the MySQL HeatWave Cluster.

- `last_updated_timestamp`

  The POSIX timestamp of the last call.

8.1.6 The rpd_nodes Table

The `rpd_nodes` read-only table provides information about MySQL HeatWave nodes.

The table may not show the current status for a new node or newly configured node immediately. The `rpd_nodes` table is updated after the node has successfully joined the cluster.

If additional nodes fail while node recovery is in progress, the newly failed nodes are not detected and their status is not updated in the `performance_schema.rpd_nodes` table until after the current recovery operation finishes and the nodes that failed previously have rejoined the cluster.

The `rpd_nodes` table has the following columns:

- `ID`

  A unique identifier for the MySQL HeatWave node.

- `CORES`

  The number of cores used by the MySQL HeatWave node.

- `MEMORY_USAGE`

  Node memory usage in bytes. The value is refreshed every four seconds. If a query starts and finishes in the four seconds between refreshes, the memory used by the query is not accounted for in the reported value.

- `MEMORY_TOTAL`

  The total memory in bytes allocated to the MySQL HeatWave node.

- `BASEREL_MEMORY_USAGE`

  The base relation memory footprint per node.

- `ML_MEMORY_STATE`

  Available as of MySQL 9.4.0. The current `ML_MEMORY_STATE` of the node. The available statuses are `IDLE`, `IDLE_CACHED`, or `ACTIVE`. Updated about every five seconds, up to a maximum of every 11 seconds when `ML_MEMORY_STATE` is `ACTIVE`. Updated about every 60 seconds, up to a maximum of every 66 seconds when `ML_MEMORY_STATE` is `IDLE`, or `IDLE_CACHED`.

- `ML_TOTAL_MEMORY_USAGE`

  Available as of MySQL 9.4.0. The total amount of memory being used by the driver in bytes. Updated about every five seconds, up to a maximum of every 11 seconds when `ML_MEMORY_STATE` is `ACTIVE`. Updated about every 60 seconds, up to a maximum of every 66 seconds when `ML_MEMORY_STATE` is `IDLE`, or `IDLE_CACHED`.

- `ML_MEMORY_TOTAL`

  Available as of MySQL 9.4.0. The maximum amount of memory dedicated to the driver. Updated about every five seconds, up to a maximum of every 11 seconds when `ML_MEMORY_STATE` is `ACTIVE`. Updated about every 60 seconds, up to a maximum of every 66 seconds when `ML_MEMORY_STATE` is `IDLE`, or `IDLE_CACHED`.

- `ML_MODEL_MEMORY_USAGE`

  Available as of MySQL 9.4.0. The amount of memory used by the driver for all loaded models in bytes. Updated about every five seconds, up to a maximum of every 11 seconds when `ML_MEMORY_STATE` is `ACTIVE`. Updated about every 60 seconds, up to a maximum of every 66 seconds when `ML_MEMORY_STATE` is `IDLE`, or `IDLE_CACHED`.

- `ML_MODEL_LOADED_COUNT`

  Available as of MySQL 9.4.0. The number of currently loaded models. Updated about every five seconds, up to a maximum of every 11 seconds when `ML_MEMORY_STATE` is `ACTIVE`. Updated about every 60 seconds, up to a maximum of every 66 seconds when `ML_MEMORY_STATE` is `IDLE`, or `IDLE_CACHED`.

- `STATUS`

  The status of the MySQL HeatWave node. Possible statuses include:

  - `NOTAVAIL_RNSTATE`

    Not available.

  - `AVAIL_RNSTATE`

    Available.

  - `DOWN_RNSTATE`

    Down.

  - `SPARE_RNSTATE`

    Spare.

  - `DEAD_RNSTATE`

    The node is not operational.

- `IP`

  IP address of the MySQL HeatWave node.

- `PORT`

  The port on which the MySQL HeatWave node was started.

- `CLUSTER_EVENT_NUM`

  The number of cluster events such as node down, node up, and so on.

- `NUM_OBJSTORE_GETS`

  Number of `GET` requests from the node to the object store.

- `NUM_OBJSTORE_PUTS`

  The number of `PUT` requests from the node to the object store.

- `NUM_OBJSTORE_DELETES`

  The number of `DELETE` requests from the node to the object store.

- `ML_STATUS`

  MySQL HeatWave AutoML status. Possible status values include:

  - `UNAVAIL_MLSTATE`: MySQL HeatWave AutoML is not available.
  - `AVAIL_MLSTATE`: MySQL HeatWave AutoML is available.
  - `DOWN_MLSTATE`: MySQL HeatWave AutoML declares the node is down.

8.1.7 The rpd_preload_stats Table

The `rpd_preload_stats` table stores column level statistics collected from `InnoDB` tables. The statistics are used to estimate the number of MySQL HeatWave nodes required for a given dataset before loading the tables into the MySQL HeatWave Cluster.

The `rpd_preload_stats` table has these columns:

- `TABLE_SCHEMA`

  The schema name.

- `TABLE_NAME`

  The table name.

- `COLUMN_NAME`

  The column name.

- `AVG_BYTE_WIDTH_INC_NULL`

  The average byte width of the column. The average value includes NULL values.

The `rpd_preload_stats` table has the following characteristics:

- It is read-only. DDL, including `TRUNCATE TABLE`, is not permitted.
- It is truncated before new statistics are added.
- It cannot be dropped, truncated, or modified by DML statements directly.
- Statistics are not persisted when the server is shut down.
- It has a limit of 65536 rows.
- Statistics are approximate, based on an adaptive sample scan.
- Statistics are deterministic, provided that the data does not change.

8.1.8 The rpd_query_stats Table

The `rpd_query_stats` table stores query compilation and execution statistics produced by the MySQL HeatWave plugin in `JSON` format. One row of data is stored for each query. The table stores data for the last 1000 executed queries. Data is stored for successfully processed queries and failed queries.

For MySQL HeatWave AutoML routines that include multiple sub-queries, such as `ML_TRAIN`, a new record is used for each query.

The `rpd_query_stats` table has these columns:

- `CONNECTION_ID`

  The ID of the connection.

- `QUERY_ID`

  The query ID. The counter auto-increments for each MySQL HeatWave and MySQL HeatWave AutoML query.

- `MYSQL_QUERY_ID`

  The MySQL Shell query ID. The counter auto-increments for each query run using MySQL Shell. Available as of MySQL 9.4.0.

- `QUERY_TEXT`

  The `RAPID` engine query or MySQL HeatWave AutoML query run by the user.

- `QEXEC_TEXT`

  Query execution log. For MySQL HeatWave AutoML, this contains the arguments the user passed to the MySQL HeatWave AutoML routine call.

- `QKRN_TEXT`

  Logical query execution plan. This field is not used for MySQL HeatWave AutoML queries.

- `QEP_TEXT`

  Physical query execution plan. This field is not used for MySQL HeatWave AutoML queries.

  Includes `prepart` data, which can be queried to determine if a `JOIN` or `GROUP BY` query used data placement partitions. See Section 4.5, “Retrieve MySQL HeatWave Metadata”.

- `STATEMENT_ID`

  The global query ID.

- `INTERNAL_OFFLOAD_FAIL_TEXT`

  Internal offload failure message for the query failing the offload.

- `EXTERNAL_OFFLOAD_FAIL_TEXT`

  External offload failure message for the query failing the offload.

- `OFFLOAD_FAIL_QUERY_SUBSTR_START`

  Points to the start of the query substring that is responsible for the offload failure in the original query. Available as of MySQL 9.1.0.

- `OFFLOAD_FAIL_QUERY_SUBSTR_LEN`

Length of the query substring that is responsible for the offload failure in the original query from its start. Available as of MySQL 9.1.0.

- `EXPORT_FILE_PREFIX`

  Contains the export file prefix used in queries that export query results to Object Storage. Available as of MySQL 9.1.1.

8.1.9 The rpd_table_id Table

The `rpd_table_id` table provides the ID, name, and schema of the tables loaded in MySQL HeatWave.

The `rpd_table_id` table has these columns:

- `ID`

  A unique identifier for the table.

- `NAME`

  The full table name including the schema.

- `SCHEMA_NAME`

  The schema name.

- `TABLE_NAME`

  The table name.

- `MATERIALIZATION_QUERY`

  Available as of MySQL 9.3.0. The column contains the query used to create the MySQL HeatWave temporary table.

  As of MySQL 9.5.0, the column contains the query used to create the materialized view or the MySQL HeatWave temporary table.

The `rpd_table_id` table is read-only.

8.1.10 The rpd_tables Table

The `rpd_tables` table provides the system change number (SCN) and load pool type for tables loaded in MySQL HeatWave.

The `rpd_tables` table is read-only.

The `rpd_tables` table has these columns:

- `ID`

  A unique identifier for the table.

- `SNAPSHOT_SCN`

  The system change number (SCN) of the table snapshot. The SCN is an internal number that represents a point in time according to the system logical clock that the table snapshot was transactionally consistent with the source table.

- `PERSISTED_SCN`

  The SCN up to which changes are persisted.

- POOL_TYPE

  The load pool type of the table.

  - As of MySQL 9.5.0, the possible values are SNAPSHOT, TRANSACTIONAL, and VOLATILE.
  - As of MySQL 9.2.1, the possible values are SNAPSHOT and TRANSACTIONAL.
  - In previous versions of MySQL, the possible values are RAPID_LOAD_POOL_SNAPSHOT and RAPID_LOAD_POOL_TRANSACTIONAL.

- DATA_PLACEMENT_TYPE

  The data placement type.

- NROWS

  The number of rows that are loaded for the table. The value is set initially when the table is loaded, and updated as changes are propagated.

- LOAD_STATUS

  The load status of the table. Statuses include:

  - NOLOAD_RPDGSTABSTATE

    The table is not yet loaded.

  - LOADING_RPDGSTABSTATE

    The table is being loaded.

  - AVAIL_RPDGSTABSTATE

    The table is loaded and available for queries.

  - UNLOADING_RPDGSTABSTATE

    The table is being unloaded.

  - INRECOVERY_RPDGSTABSTATE

    The table is being recovered. After completion of the recovery operation, the table is placed back in the UNAVAIL_RPDGSTABSTATE state if there are pending recoveries.

  - STALE_RPDGSTABSTATE

    A failure during change propagation, and the table has become stale. See Section 4.2.7, “About Change Propagation”

    Note

    When TRUNCATE TABLE operation is performed, the table is marked for a full reload. However, the table is not reloaded immediately. The system reloads the table only after all currently undergoing incremental or scheduled reloads. hence, the table may appear as stale in the system (e.g., in RPDGSTABSTATE) until it is reloaded completely.

  - UNAVAIL_RPDGSTABSTATE

    The table is unavailable.

  - RECOVERYFAILED_RPDGSTABSTATE

This status is available as of MySQL 9.2.2. A Lakehouse table is marked as `RECOVERYFAILED_RPDGSTABSTATE` when an attempt to recover it fails. Tables in this state can be unloaded. Therefore, before using tables in this state, you need to manually reload them using `SECONDARY_UNLOAD` followed by `SECONDARY_LOAD`. When a MySQL HeatWave cluster restarts, there is no attempt to recover tables in `RECOVERYFAILED_RPDGSTABSTATE`. However, when a DB System restarts, an attempt is made to recover tables in this state.

- `LOAD_PROGRESS`

  The load progress of the table expressed as a percentage value.

  For Lakehouse, the following values are returned:

  - 10%: the initialization phase is complete.
  - 10-70%: the transformation to native MySQL HeatWave format is in progress.
  - 70% - 80%: the transformation to native MySQL HeatWave format is complete and the aggregation phase is in progress.
  - 80-99%: the recovery phase is in progress.
  - 100%: data load is complete.

- `SIZE_BYTES`

  The amount of data loaded for the table, in bytes.

- `TRANSFORMATION_BYTES`:

  The total size of raw Lakehouse data transformed, in bytes.

- `QUERY_COUNT`

  The number of queries that referenced the table.

- `LAST_QUERIED`

  The timestamp of the last query that referenced the table.

- `LOAD_START_TIMESTAMP`

  The load start timestamp for the table.

- `LOAD_END_TIMESTAMP`

  The load completion timestamp for the table.

- `RECOVERY_SOURCE`

  Indicates the source of the last successful recovery for a table. The values are `MySQL`, that is InnoDB, or `ObjectStorage`.

- `RECOVERY_START_TIMESTAMP`

  The timestamp when the latest successful recovery started.

- `RECOVERY_END_TIMESTAMP`

  The timestamp when the latest successful recovery ended.

- `LOAD_TYPE`

Specifies whether the table is automatically loaded by MySQL HeatWave (`SELF`) or loaded by the user (`USER`).

- `LOGICAL_PARTS_LOADED_AT_SCN`

  For partitioned tables, contains an array of JSON objects, each corresponding to a loaded partition. For each partition, the following values are stored:

  - `id`: the id of the partition in `InnoDB`.
  - `name`: the name of the partition.
  - `load_scn`: the system change number (SCN) at which the partition was loaded.
  - `load_type`: specifies whether the table is automatically loaded by MySQL HeatWave, or loaded by the user.

- `AUTO_ZMP_COLUMNS`

  Contains a list of IDs that correspond to the columns on which zone maps are automatically built.

- `ACE_MODEL`

  Available as of MySQL 9.3.0. Indicates whether an Advanced Cardinality Estimation (ACE) statistics model is currently associated with the given table in MySQL HeatWave.

- STALE_REASON

  Available as of MySQL 9.7.0. Identifies the reason that a table became stale.

  It includes the following reasons:

  - `OK`

    The table is not stale.

  - `ERROR_CLUSTER_OOM`

    The MySQL HeatWave Cluster ran out of memory.

  - `ERROR_PKEY_NOT_FOUND`

    The old version of an updated or deleted row was not found.

  - `ERROR_UU_OVERLAP`

    Update units with overlapping primary keys were found.

  - `ERROR_INVALID_CP_PACKET`

    An invalid change propagation packet.

  - `ERROR_UU_TOO_LARGE`

    An update unit that was larger than expected was found.

  - `DECOMPRESSION_TIMEOUT`

    The decompression of data took too long and timed out.

  - `ERROR_CLUSTER_OTHER`

    A non-specific error with the MySQL HeatWave Cluster.

  - `TABLE_TYPE_MISMATCH`

    An unexpected type of table was found (InnoDB versus Lakehouse).

  - `RPD_CP_RESET`

    Change propagation encountered a systemic error and had to reset.

  - `DROP_PARTITION_FAILED`

    An operation to drop a partition failed.

  - `ADD_PARTITION_FAILED`

    An operation to add a partition failed.

  - `ALTER_PARTITION_FAILED`

    An operation to alter a partition failed.

  - `PARTITION_UNLOAD_FAILED`

    An operation to unload a partition failed.

  - `PARTITION_LOAD_FAILED`

An operation to load a partition failed.

- `RELOAD_REQUIRED`

  The change propagation operation requires a reload.

- `RPD_PARSING_OOM`

  Binlog parsing ran out of memory.

- `RPD_PARSER_ERROR`

  Binlog parsing encountered a parsing error.

- `UNIDENTIFIED_ERROR`

  An unexpected error occurred during change propagation.

If tables become stale, you must unload them and load them again. To learn more, see the following:

- Unload Data from MySQL HeatWave Cluster
- Load Data Using Auto Parallel Load
- Load Data Manually

8.1.11 The rpd_vector_indexes Table

The `rpd_vector_indexes` table provides information about vector indexes created for different vector store or embedding tables.

The `rpd_vector_indexes` table has these columns:

- `TABLE_ID`: the table ID.
- `COLUMN_ID`: the column ID.
- `SPACE`: Vector distance metric, which can be one of: `DOT`, `EUCLIDEAN`, and `COSINE`. Queries using the same distance metric are accelerated by the index.
- `ALGORITHM`: vector indexing algorithm used, which is always `HNSW` as of MySQL 9.5.0.
- `TUNING_PARAMS`: the parameters used for tuning the vector index.
- `ORDER`: the order used for index creation: `ACS` or `DESC`.
- `SIZE_MB`: the size of the index per-rapid-node, in MB.
- `USAGE_COUNT`: the number of times the index has been used.
- `BUILD_START_TIME`: index build start time.
- `BUILD_END_TIME`: index build end time.
- `IS_USABLE`: whether the index is usable.

The `rpd_vector_indexes` table is read-only.

8.2 Option Tracker

The Option Tracker component provides usage information about different features and components of MySQL HeatWave.

For more information, see Option Tracker Component.

As of MySQL 9.2.1, the boolean flag `used` has been replaced with an integer flag `usedCounter` in the `option_tracker_usage_get()` and `option_tracker_usage_set()` functions. The counter is incremented in real-time and persisted to storage every hour.

- The `option_tracker_usage_get()` function returns a value similar to the following:

```
mysql> SELECT option_tracker_usage_get('Berry Picker');
+-----------------------------------------------------+
| option_tracker_usage_get('Berry Picker')            |
+-----------------------------------------------------+
| {"usedCounter": 30, "usedDate": "2025-14-16T09:14:41Z"} |
+-----------------------------------------------------+
```

- The `option_tracker_usage_set()` function accepts JSON-formatted string similar to the following for the `usage_data` argument:

```
{
  "usedCounter": "integer"
  "usedDate": "ISO8601 date"
}
```

Chapter 9 System and Status Variables

Table of Contents

9.1 System Variables 531
9.2 Status Variables 542

9.1 System Variables

MySQL HeatWave maintains several variables that configure its operation. Variables are set when the MySQL HeatWave Cluster is enabled. Most MySQL HeatWave variable settings are managed by OCI and cannot be modified directly.

- `bulk_loader.data_memory_size`

Command-Line Format	`--bulk_loader.data_memory_size=#`
System Variable	`bulk_loader.data_memory_size`
Scope	Global
Dynamic	Yes
`SET_VAR` Hint Applies	No
Type	Integer
Default Value	`(0.125) * #memory GB`
Minimum Value	`67108864`
Maximum Value	`1099511627776`

Specifies the amount of memory to use for `LOAD DATA` with `ALGORITHM=BULK`, in bytes. See: Section 4.1.2, “Bulk Ingest Data”.

- `bulk_loader.concurrency`

System Variable	`bulk_loader.concurrency`
Scope	Global
Dynamic	Yes
`SET_VAR` Hint Applies	No
Type	Integer
Default Value	`(1/2) * #vcpus`
Minimum Value	`1`
Maximum Value	`1024`

The maximum number of concurrent threads to use by one LOAD statement with `ALGORITHM=BULK`. See: Section 4.1.2, “Bulk Ingest Data”.

- `bulk_loader.chunk_size`

System Variable	`bulk_loader.chunk_size`
Scope	Global
Dynamic	Yes
`SET_VAR` Hint Applies	No
Type	Integer

Default Value	4194304
Minimum Value	1048576
Maximum Value	16777216

The queue chunk size in MiB used for reading data from CSV files.

- Each thread uses one queue with four elements. Another element is used for data conversion. Total memory requirement by the component is (4 + 1) x bulk_loader.chunk_size x bulk_loader.concurrency. The default value is 5 x 16 x 4 MiB = 320 MiB.
- The above calculation is for a single LOAD operation running with ALGORITHM = BULK. For concurrent execution, you need to multiply the value with the number of LOAD operations running with ALGORITHM=BULK. See: Section 4.1.2, “Bulk Ingest Data”.

- lakehouse_filter_warning_codes_list

Command-Line Format	--lakehouse_filter_warning_codes_list=codes
System Variable	lakehouse_filter_warning_codes_list
Scope	Session
Dynamic	Yes
SET_VAR Hint Applies	No
Type	String
Default Value	{6042,6044}

Specifies the warning codes for Lakehouse to filter.

- lakehouse_filter_warning_modes_list

Command-Line Format	--lakehouse_filter_warning_modes_list=modes
System Variable	lakehouse_filter_warning_modes_list
Scope	Session
Dynamic	Yes
SET_VAR Hint Applies	No
Type	String
Default Value	EMPTY_FILES,DUPLICATE_FILES,MISSING_FILES,NULL
Valid Values	AVRO_FILE_BLOCK_HEADER DUPLICATE_FILES EMPTY_FILES EMPTY_MISSING_COLUMNS INFER_SKIPPED MISSING_FILES NULL_COLUMNS NUMERIC_PARSING NUMERIC_TRUNCATION

	PARQUET_SCHEMA STRING_PARSING STRING_TRUNCATION TEMPORAL_PARSING TEMPORAL_TRUNCATION

Specifies the warning modes for Lakehouse to filter.

- rapid_compression

System Variable	rapid_compression
Scope	Session
Dynamic	Yes
SET_VAR Hint Applies	No
Type	Enumeration
Default Value	AUTO
Valid Values	ON OFF

Whether to enable or disable data compression before loading data into MySQL HeatWave. Data compression is enabled by default. The setting does not affect data that is already loaded. See Section 4.2.3, “Enable or Disable Data Compression”.

The default option is AUTO which automatically chooses the best compression algorithm for each column.

- rapid_bootstrap

Command-Line Format	--rapid-bootstrap[={OFF|ON|IDLE}]
System Variable	rapid_bootstrap
Scope	Global
Dynamic	Yes
SET_VAR Hint Applies	No
Type	Enumeration
Default Value	OFF
Valid Values	IDLE SUSPEND ON

The setting for this variable is managed by OCI and cannot be modified directly. Defines the MySQL HeatWave Cluster bootstrap state. States include:

- OFF

  The MySQL HeatWave Cluster is not bootstrapped (not initialized).

  - `IDLE`

    The MySQL HeatWave Cluster is idle (stopped).

  - `SUSPENDED`

    The MySQL HeatWave Cluster is suspended. The `SUSPENDED` state is a transition state between `IDLE` and `ON` that facilitates planned restarts of the MySQL HeatWave Cluster.

  - `ON`

    The MySQL HeatWave Cluster is bootstrapped (started).

- `rapid_dmem_size`

Command-Line Format	`--rapid-dmem-size=#`
System Variable	`rapid_dmem_size`
Scope	Global
Dynamic	Yes
`SET_VAR` Hint Applies	No
Type	Integer
Default Value	`2048`
Minimum Value	`512`
Maximum Value	`2097152`

  The setting for this variable is managed by OCI and cannot be modified directly. Specifies the amount of DMEM available on each core of each node, in bytes.

- `rapid_memory_heap_size`

Command-Line Format	`--rapid-memory-heap-size=#`
System Variable	`rapid_memory_heap_size`
Scope	Global
Dynamic	Yes
`SET_VAR` Hint Applies	No
Type	Integer
Default Value	`unlimited`
Minimum Value	`67108864`
Maximum Value	`unlimited`

  The setting for this variable is managed by OCI and cannot be modified directly. Defines the amount of memory available for the MySQL HeatWave plugin, in bytes. Ensures that MySQL HeatWave does not use more memory than is allocated to it.

- `rapid_execution_strategy`

Command-Line Format	`--rapid_execution_strategy[={MIN_RUNTIME|MIN_MEM_CONSUMPTION}]`
System Variable	`rapid_execution_strategy`
Scope	Session

Dynamic	No
SET_VAR Hint Applies	No
Type	Enumeration
Default Value	MIN_RUNTIME
Valid Values	MIN_RUNTIME MIN_MEM_CONSUMPTION

Specifies the query execution strategy to use. Minimum runtime (MIN_RUNTIME) or minimum memory consumption (MIN_MEM_CONSUMPTION).

MySQL HeatWave optimizes for network usage rather than memory. If you encounter out of memory errors when running a query, try running the query with the MIN_MEM_CONSUMPTION strategy by setting by setting rapid_execution_strategy prior to executing the query:

```
SET SESSION rapid_execution_strategy = MIN_MEM_CONSUMPTION;
```

See Section 11.1.1, “MySQL HeatWave Issues”.

- rapid_stats_cache_max_entries

Command-Line Format	--rapid-stats-cache-max-entries=#
System Variable	rapid_stats_cache_max_entries
Scope	Global
Dynamic	Yes
SET_VAR Hint Applies	No
Type	Integer
Default Value	65536
Minimum Value	0
Maximum Value	1048576

The setting for this variable is managed by OCI and cannot be modified directly. Specifies the maximum number of entries in the statistics cache.

The number of entries permitted in the statistics cache by default is 65536, which is enough to store statistics for 4000 to 5000 unique queries of medium complexity.

For more information, see Section 5.4.12, “Auto Query Plan Improvement”.

- rapid_ml_genai

Command-Line Format	--rapid-ml-genai=#
System Variable	rapid_ml_genai
Scope	Session
Dynamic	No
SET_VAR Hint Applies	No
Type	String
Default Value	Compartment ID of the DB system currently in use

The setting for this variable is managed by OCI and cannot be modified directly. To use the OCI Generative AI Service with MySQL HeatWave GenAI, MySQL HeatWave automatically sets this

system variable for resource principal authentication. The default value of the variable is the compartment ID of the database system currently in use.

- `rapid_ml_genai_session`

Command-Line Format	`--rapid-ml-genai-session=#`
System Variable	`rapid_ml_genai_session`
Scope	Session
Dynamic	Yes
`SET_VAR` Hint Applies	No
Type	String
Default Value	`Compartment ID of the DB system currently in use`

Specifies the ID of the compartment you want to use for accessing the OCI Generative AI service. If you do not want to use the compartment ID of the DB system currently in use for accessing the OCI Generative AI service, then you can set this system variable to specify the ID of the compartment you want to use.

- `rapid_enable_my_sc`

Command-Line Format	`--rapid-enable-my-sc=#`
System Variable	`rapid_enable_my_sc`
Scope	Session
Dynamic	Yes
`SET_VAR` Hint Applies	No
Type	Enumeration
Default Value	`ON`
Valid Values	`ON` `OFF`

As of MySQL 9.4.1, controls whether statistics feedback from the `InnoDB` execution engine is used to improve query plans, particularly for queries with two or more joins and high MySQL costs. By injecting actual statistics from previous query executions into subsequent optimizations, MySQL HeatWave can produce more accurate and efficient plans, leading to significant performance improvements.

- `rapid_ap_stats_cache_max_entries`

Command-Line Format	`--rapid-ap-stats-cache-max-entries=#`
System Variable	`rapid_ap_stats_cache_max_entries`
Scope	Global
Dynamic	Yes
`SET_VAR` Hint Applies	No
Type	Integer
Default Value	`65536`
Minimum Value	`0`

Maximum Value	1048576

As of MySQL 9.4.1, controls the size of the HashMap used to control statistics feedback. Setting to 0 disables statistics feedback. Lower values imply fewer statistics cached for feedback.

- rapid_hnsw_ef_search

System Variable	rapid_hnsw_ef_search
Scope	Global
Dynamic	Yes
SET_VAR Hint Applies	No
Type	Integer
Default Value	40
Minimum Value	0
Maximum Value	50000

As of MySQL 9.5.0, controls the accuracy of query results using the vector index. To disable the usage of HNSW indexes, you can set this session variable to 0.

- rapid_hnsw_use_adaptive_accuracy_estimation

System Variable	rapid_hnsw_use_adaptive_accuracy_estimation
Scope	Global
Dynamic	Yes
SET_VAR Hint Applies	No
Type	Enumeration
Default Value	ON
Valid Values	ON OFF

As of MySQL 9.5.0, controls adaptive accuracy estimation for similarity search queries that use the vector indexes.

- rapid_auto_vector_index_enabled

System Variable	rapid_auto_vector_index_enabled
Scope	Global
Dynamic	Yes
SET_VAR Hint Applies	No
Type	Enumeration
Default Value	ON
Valid Values	ON OFF

As of MySQL 9.5.0, controls automatic vector index creation for frequently queried vector store columns. If disabled, all automatically created indexes are removed and no more automatic index creation happens until this variable is enabled.

- `rapid_auto_vector_index_interval_seconds`

System Variable	`rapid_auto_vector_index_interval_seconds`
Scope	Global
Dynamic	Yes
`SET_VAR` Hint Applies	No
Type	Integer
Default Value	`86400`
Minimum Value	`30`
Maximum Value	`604800`

As of MySQL 9.5.0, specifies the time interval, in seconds, for automatically creating the vector indexes for frequently queried vector store columns.

- `rapid_auto_vector_index_server_memory_quota_pct`

System Variable	`rapid_auto_vector_index_server_memory_quota_pc`
Scope	Global
Dynamic	Yes
`SET_VAR` Hint Applies	No
Type	Integer
Default Value	`20`
Minimum Value	`0`
Maximum Value	`100`

As of MySQL 9.5.0, specifies the maximum memory percentage that can be used by created indexes. Less beneficial indexes are removed if memory limits are exceeded, making space for more beneficial indexes.

- `rapid_auto_vector_index_skip_quiet_check`

System Variable	`rapid_auto_vector_index_skip_quiet_check`
Scope	Global
Dynamic	Yes
`SET_VAR` Hint Applies	No
Type	Enumeration
Default Value	`OFF`
Valid Values	`ON` `OFF`

As of MySQL 9.5.0, controls whether MySQL HeatWave checks if the system is quiet before automatic vector index creation is started.

A quiet system is:

- When no tables have been queried in MySQL HeatWave within the past 5 minutes.
- When no ongoing table is loading into MySQL HeatWave.
- There is no lag in change propagation.

- If the system is not found to be quiet, MySQL HeatWave holds automatic index creation for 300 seconds and checks up to 10 times before skipping until the next interval.

This variable cannot be modified when RAPID is not bootstrapped.

- `rapid_auto_vector_index_build_timeout_seconds`

System Variable	`rapid_auto_vector_index_build_timeout_secon`
Scope	Global
Dynamic	Yes
`SET_VAR` Hint Applies	No
Type	Integer
Default Value	`3600`
Minimum Value	`1`
Maximum Value	`43200`

As of MySQL 9.5.0, specifies the timeout duration, in seconds, for automatic vector index creation.

This variable cannot be modified when RAPID is not bootstrapped.

- `secondary_engine_cost_threshold`

System Variable	`secondary_engine_cost_threshold`
Scope	Session
Dynamic	Yes
`SET_VAR` Hint Applies	Yes
Type	Numeric
Default Value	`100000.000000`
Minimum Value	`0`
Maximum Value	`DBL_MAX (maximum double value)`

Defines the cost estimate threshold for offloading queries to the secondary engine. A query with a cost estimate on the primary engine that is higher than the threshold value is considered for execution on the secondary engine.

- `show_create_table_skip_secondary_engine`

Command-Line Format	`--show-create-table-skip-secondary-engine[={OFF|ON}]`
System Variable	`show_create_table_skip_secondary_engine`
Scope	Session
Dynamic	Yes
`SET_VAR` Hint Applies	Yes
Type	Boolean

Default Value	OFF

Whether to exclude the SECONDARY ENGINE clause from SHOW CREATE TABLE output, and from CREATE TABLE statements dumped by the mysqldump utility.

mysqldump provides the --show-create-skip-secondary-engine option. When specified, it enables the show_create_table_skip_secondary_engine system variable for the duration of the dump operation.

- use_secondary_engine

System Variable	use_secondary_engine
Scope	Session
Dynamic	Yes
SET_VAR Hint Applies	Yes
Type	Enumeration
Default Value	ON
Valid Values	OFF ON FORCED

Whether to execute queries using the secondary engine. These values are permitted:

- OFF: Queries execute using the primary storage (InnoDB) on the MySQL DB System. Execution using the secondary engine (RAPID) is disabled.
- ON: Queries execute using the secondary engine (RAPID) when conditions warrant, falling back to the primary storage engine (InnoDB) otherwise. In the case of fallback to the primary engine, whenever that occurs during statement processing, the attempt to use the secondary engine is abandoned and execution is attempted using the primary engine.
- FORCED: Queries always execute using the secondary engine (RAPID) or fail if that is not possible. Under this mode, a query returns an error if it cannot be executed using the secondary engine, regardless of whether the tables that are accessed have a secondary engine defined.

- external_table_storage_engine

Command-Line Format	--external_table_storage_engine=text
System Variable	external_table_storage_engine
Scope	Global, Session
Dynamic	Yes
SET_VAR Hint Applies	No
Type	String
Default Value	LAKEHOUSE

Determines which engines are used for external tables.

- rapid_ml_concurrency

Command-Line Format	--rapid_ml_concurrency=#
System Variable	rapid_ml_concurrency
Scope	Global

Dynamic	Yes
SET_VAR Hint Applies	No
Type	Integer
Default Value	8
Minimum Value	1
Maximum Value	8

Allows concurrent query processing for MySQL HeatWave AutoML and MySQL HeatWave GenAI as of MySQL 9.4.1. It can have a value between 1 and 8. The default value is 8. A value over 1 allows for concurrent query processing.

We recommend not enabling concurrent query processing with the following DB System shapes:

- HeatWave.Free
- HeatWave.32GB

- rapid_ace_auto_train_enabled

System Variable	rapid_ace_auto_train_enabled
Scope	Global
Dynamic	Yes
SET_VAR Hint Applies	No
Type	Boolean
Default Value	ON

As of MySQL 9.4.2, enables or disables Advanced Cardinality Estimation (ACE) auto-training, which periodically runs in the background and manages rebuilding stale ACE models for loaded tables. See Analyze Tables.

- rapid_ace_auto_train_interval_seconds

System Variable	rapid_ace_auto_train_interval_seconds
Scope	Global
Dynamic	Yes
SET_VAR Hint Applies	No
Type	Integer
Default Value	3600
Minimum Value	60
Maximum Value	86400

As of MySQL 9.4.2, determines the time (in seconds) between runs of ACE auto-training. See Analyze Tables.

- rapid_ace_auto_train_max_execution_seconds

System Variable	rapid_ace_auto_train_max_execution_seconds
Scope	Global
Dynamic	Yes
SET_VAR Hint Applies	No
Type	Integer

Default Value	180
Minimum Value	30
Maximum Value	1800

As of MySQL 9.4.2, determines the maximum amount of time in seconds ACE auto-training runs before stopping and completing early. See Analyze Tables.

9.2 Status Variables

Several status variables provide operational information about MySQL HeatWave. You can retrieve status data using `SHOW STATUS` syntax. For example:

```
mysql> SHOW STATUS LIKE 'hw_data_scanned';
+-----------------+-------+
| Variable_name   | Value |
+-----------------+-------+
| hw_data_scanned | 0     |
+-----------------+-------+

mysql> SHOW STATUS LIKE 'secondary%';
+----------------------------------+-------+
| Variable_name                    | Value |
+----------------------------------+-------+
| Secondary_engine_execution_count | 0     |
+----------------------------------+-------+

mysql> SHOW STATUS LIKE 'rapid%';
+---------------------------------+------------+
| Variable_name                   | Value      |
+---------------------------------+------------+
| rapid_change_propagation_status | ON         |
| rapid_cluster_status            | ON         |
| rapid_core_count                | 64         |
| rapid_heap_usage                | 58720397   |
| rapid_load_progress             | 100.000000 |
| rapid_ml_operation_count        | 2          |
| rapid_ml_status                 | ON         |
| rapid_plugin_bootstrapped       | YES        |
| rapid_preload_stats_status      | Available  |
| rapid_query_offload_count       | 46         |
| rapid_service_status            | ONLINE     |
+---------------------------------+------------+
```

- `hw_data_scanned`

  Tracks the amount of data scanned by successfully executed MySQL HeatWave queries. Data is tracked in megabytes and is a cumulative total of data scanned since the MySQL HeatWave Cluster was last started. The counter is reset to 0 when the MySQL HeatWave Cluster is restarted (when the `rapid_bootstrap` state changes from `OFF` or `IDLE` to `ON`.)

- `rapid_change_propagation_status`

  The change propagation status.

  - `ON`: Indicates that change propagation is enabled globally, permitting changes to `InnoDB` tables on the MySQL DB System to be propagated to their counterpart tables in the MySQL HeatWave Cluster.
  - `OFF`: Change propagation is not enabled.

- `rapid_cluster_status`

  The MySQL HeatWave Cluster status.

  - `ON`: The MySQL HeatWave Cluster is enabled.

  - `OFF`: The MySQL HeatWave Cluster is not enabled.

- `rapid_core_count`

  The MySQL HeatWave node core count. The value remains at 0 until all MySQL HeatWave nodes are started.

- `rapid_heap_usage`

  MySQL DB System node heap usage.

- `rapid_lakehouse_automatic_incremental_load_health`

  Available as of MySQL 9.4.1. The value of this variable is a JSON object containing two string fields: `status` and `details`. You can review the `details` field for a description of the status, such as a reason for a `disabled` status.

  The following statuses are available:

  - `disabled`: The event-based incremental load feature is not available. Review the `details` field for more information on the reason for the status.
  - `initializing`: The event-based incremental load feature is waiting for the server to finish booting and verifying which external tables are set up for event-based incremental loading.
  - `online`: The event-based incremental load feature is available. The `details` field states the number of unique streams that are set up for a specific number of tables. For example: `Listening for` *`n`* `streams for` *`x`* `tables.`

- `rapid_lakehouse_health`

  - `OFFLINE`: Indicates an issue with the RAPID storage engine.
  - `RESOURCEPRINCIPALDOWN`: The resource principal token could not be retrieved.
  - `RESOURCEPRINCIPALNOTSET`: The resource principal endpoint was not set.
  - `LAKEHOUSEDISABLED`: Lakehouse is not currently enabled.
  - `ONLINE`: Lakehouse is enabled and running.

- `rapid_lakehouse_total_loaded_bytes`

  Total bytes of all external tables loaded in MySQL HeatWave.

- `rapid_load_progress`

  A percentage value indicating the status of a table load operation.

- `rapid_ml_operation_count`

  A count of the number of MySQL HeatWave AutoML operations that a dbsystem executes. An operation typically includes multiple queries, and it increases for both successful and failed queries. It resets when a MySQL HeatWave node restarts.

- `rapid_ml_status`

  The status of MySQL HeatWave AutoML.

  - `ON`: MySQL HeatWave AutoML is available to use.
  - `OFF`: MySQL HeatWave AutoML is not available to use.

- `rapid_n_external_tables`

  Returns the number of external tables. For example:

```
+-------------------------+-------+
| Variable_name           | Value |
+-------------------------+-------+
| rapid_n_external_tables | 2     |
+-------------------------+-------+
```

- `rapid_plugin_bootstrapped`

  The bootstrap mode for the MySQL HeatWave plugin.

  - `YES`: The MySQL HeatWave plugin is initialized and ready to use.
  - `NO`: The MySQL HeatWave plugin is installed, but it is not active.
  - `IDLE`: The MySQL HeatWave plugin is installed and the basic configuration is set. In this state, there is no connectivity to `RAPID` servers, MySQL HeatWave Cluster is not available, and change propagation is not running.
  - `SUSPENDED`: The MySQL HeatWave plugin is installed and the change propagation thread is started. The network is disabled and user operations cannot be offloaded to `RAPID`.

- `rapid_preload_stats_status`

  Reports the state of preload statistics collection. Column-level statistics are collected for tables on the MySQL DB System during a cluster size estimate. You can perform a cluster estimate when adding or editing a MySQL HeatWave Cluster. States include `Not started`, `In progress`, and `Statistics collected`.

  Preload statistics are cached in the `rpd_preload_stats` Performance Schema table. See Section 8.1.7, “The rpd_preload_stats Table”.

- `rapid_query_offload_count`

  The number of queries offloaded to MySQL HeatWave for processing.

- `rapid_service_status`

  Reports the status of the cluster as it is brought back online after a node failure.

- `Secondary_engine_execution_count`

  The number of queries executed by MySQL HeatWave. Execution occurs if query processing using the secondary engine advances past the preparation and optimization stages. The variable is incremented regardless of whether query execution is successful.

Chapter 10 Routines

Table of Contents

10.1 MySQL HeatWave Routines 545
10.1.1 HEATWAVE_LOAD 545
10.1.2 HEATWAVE_UNLOAD 550
10.1.3 HEATWAVE_RELOAD 553
10.1.4 Asynchronous Task Routines 553
10.1.5 HEATWAVE_ADVISOR 573
10.2 MySQL HeatWave AutoML Routines 575
10.2.1 ML_TRAIN 575
10.2.2 ML_EXPLAIN 589
10.2.3 ML_MODEL_EXPORT 591
10.2.4 ML_MODEL_IMPORT 593
10.2.5 ML_PREDICT_ROW 597
10.2.6 ML_PREDICT_TABLE 602
10.2.7 ML_EXPLAIN_ROW 608
10.2.8 ML_EXPLAIN_TABLE 612
10.2.9 ML_SCORE 614
10.2.10 ML_MODEL_LOAD 616
10.2.11 ML_MODEL_UNLOAD 617
10.2.12 ML_MODEL_ACTIVE 617
10.2.13 NL2ML 620
10.2.14 TRAIN_TEST_SPLIT 624
10.2.15 Model Types 626
10.2.16 Optimization and Scoring Metrics 628
10.3 MySQL HeatWave GenAI Routines 631
10.3.1 ML_GENERATE 631
10.3.2 ML_GENERATE_TABLE 635
10.3.3 VECTOR_STORE_LOAD 641
10.3.4 ML_RAG 645
10.3.5 ML_RAG_TABLE 650
10.3.6 HEATWAVE_CHAT 655
10.3.7 ML_EMBED_ROW 660
10.3.8 ML_EMBED_TABLE 662
10.3.9 NL_SQL 665
10.3.10 ML_RETRIEVE_SCHEMA_METADATA 668

The sections in this chapter list and describe various routines available in MySQL HeatWave.

10.1 MySQL HeatWave Routines

10.1.1 HEATWAVE_LOAD

MySQL 9.0.0 adds support for Lakehouse Incremental Load with the `refresh_external_tables` option, see: Section 4.3.14.2, “Refresh Data Using Incremental Load”.

MySQL 8.4.0 adds support for the following:

- An `input_list` JSON array replaces the `db_list` JSON array. This adds an include list to exactly specify the tables and columns to load for a set of queries. It is no longer necessary to include a complete schema, and exclude unnecessary tables and columns.

  `input_list` is backwards compatible with `db_list`.

- A `validation` mode for external files.

```
mysql> CALL sys.HEATWAVE_LOAD(input_list, options);

input_list: {JSON_ARRAY(input [,input] ...)}

options: {JSON_OBJECT(keyvalue[, keyvalue]...)|NULL}
keyvalue:
{
  "mode",{"normal"|"dryrun"|"validation"}
  |"output",{"normal"|"compact"|"silent"|"help"}
  |"sql_mode","sql_mode"
  |"policy",{"disable_unsupported_columns"|"not_disable_unsupported_columns"}
  |"set_load_parallelism",{true|false}
  |"auto_enc",JSON_OBJECT("mode",{"off"|"check"})
  |"refresh_external_tables",{true|false}
}

input: {'db_name' | db_object}

db_object: {JSON_OBJECT(keyvalue[, keyvalue]...)|NULL}
keyvalue:
{
  "db_name": "db_name"
  |"tables": JSON_ARRAY(table [, table] ...)
  |"exclude_tables": JSON_ARRAY(table [, table] ...)
}

table: {'table_name' | table_object}

table_object: {JSON_OBJECT("key","value"["key","value"] ...)}
keyvalue:
{
  "table_name": "table_name"
  |'engine_attribute': engine_attribute_object
  |'columns': JSON_ARRAY('column_name' [, 'column_name', ...])
  |'exclude_columns': JSON_ARRAY('column_name' [, 'column_name', ...])
}

engine_attribute_object: {JSON_OBJECT("key","value"["key","value"] ...)}
keyvalue:
{
  "sampling": true|false
  |"dialect": {dialect_section}
  |"file": JSON_ARRAY(file_section [, file_section]...)
}
```

Use `input_list` to define what to load. `input_list` is a JSON array and requires one or more valid `input` which can be either a valid schema name or a `db_object`. An empty array is permitted to view the Auto Parallel Load command-line help, see Use the Auto Parallel Load Command-Line Help. This is backwards compatible with `db_list`.

Use key-value pairs in `JSON` format to specify parameters. MySQL HeatWave uses the default setting if there is no option setting. Use `NULL` to specify no arguments.

For syntax examples, see Quickstart: Load Data Using Lakehouse Auto Parallel Load Examples.

Auto Parallel Load `options` is a JSON object literal that includes:

- `mode`: Defines the Auto Parallel Load operational mode. Permitted values are:
  - `normal`: The default. Generates and executes the load script.
  - `dryrun`: Generates a load script only. Auto Parallel Load executes in `dryrun` mode automatically if the MySQL HeatWave Cluster is not active.
  - `validation`: Only use with Lakehouse. `validation` performs the same checks as `dryrun` and also validates external files before loading. It follows all the `options` and the load configuration,

for example column information, `sql_mode`, `is_strict_mode` and `allow_missing_files`, but does not load any tables. It uses schema inference and might modify the schema, see: About Lakehouse Auto Parallel Load Schema Inference. `validation` is faster than a full load, particularly for large tables. The memory requirement is similar to running a full load.

Note

`validation` requires created tables.

- `output`: Defines how Auto Parallel Load produces output. Permitted values are:
  - `normal`: The default. Produces summarized output and sends it to `stdout` and to the `heatwave_autopilot_report` table. See Section 5.8.8, “Autopilot Report Table”.
  - `silent`: Sends output to the `heatwave_autopilot_report` table only. See Section 5.8.8, “Autopilot Report Table”. The `silent` output type is useful if human-readable output is not required; when the output is consumed by a script, for example. For an example of a stored procedure with an Auto Parallel Load call that uses the `silent` output type, see Quickstart: Load Data Using Lakehouse Auto Parallel Load Examples.
  - `compact`: Produces compact output.
  - `help`: Displays Auto Parallel Load command-line help. See Use the Auto Parallel Load Command-Line Help.
- `sql_mode`: Defines the SQL mode used while loading tables. Auto Parallel Load does not support the MySQL global or session `sql_mode` variable. To run Auto Parallel Load with a non-oci-default SQL mode configuration, specify the configuration using the Auto Parallel Load `sql_mode` option as a string value. If no SQL modes are specified, the default OCI SQL mode configuration is used.

  For information about SQL modes, see Server SQL Modes.
- `policy`: Defines the policy for handling of tables containing columns with unsupported data types. Permitted values are:
  - `disable_unsupported_columns`: The default. Disable columns with unsupported data types and include the table in the load script. Columns that are explicitly pre-defined as `NOT SECONDARY` are ignored (they are neither disabled or enabled).

    Auto Parallel Load does not generate statements to disable columns that are explicitly defined as `NOT SECONDARY`.
  - `not_disable_unsupported_columns`: Exclude the table from the load script if the table contains a column with an unsupported data type.

    A column with an unsupported data type that is explicitly defined as a `NOT SECONDARY` column does not cause the table to be excluded. For information about defining columns as `NOT SECONDARY`, see Section 4.2.6.1, “Exclude Table Columns”.
- `set_load_parallelism`: Enabled by default. Optimizes load parallelism based on machine-learning models by optimizing the `innodb_parallel_read_threads` variable setting before loading each table.
- `auto_enc`: Checks if there is enough memory for string column encoding. Settings include:
  - `mode`: Defines the `auto_enc` operational mode. Permitted values are:
    - `off`: Disables the `auto_enc` option. No memory checks are performed.
    - `check`: The default. Checks if there is enough memory on the MySQL node for dictionary-encoded columns and if there is enough root heap memory for variable-length column encoding

overhead. Dictionary-encoded columns require memory on the MySQL node for dictionaries. For each loaded table, 64KB of memory, the default heap segment size, must be allocated from the root heap for variable-length column encoding overhead. If there is not enough memory, Auto Parallel Load executes in `dryrun` mode and prints a warning about insufficient memory. The `auto_enc` option runs `check` mode if it is not specified explicitly and set to `off`. For more information, see Memory Estimation for String Column Encoding.

- `refresh_external_tables`: Only use with Lakehouse. Set to `true` to refresh external tables. See: Section 4.3.14.2, “Refresh Data Using Incremental Load”.

  Set to `false`, the default setting, to only load new tables, unloaded tables, and not refresh external tables.

  This option only works with `mode` set to `normal`. All other `mode` settings ignore this option.

The `db_object` is a JSON object literal that includes:

- `db_name`: The name of the database to load.
- Use one or other of the following, but not both. The use of both parameters will throw an error.
  - `tables`: An optional JSON array of `table` to include in the load.
  - `exclude_tables`: As of MySQL 8.4.0, an optional JSON array of `table` to exclude from the load.
- `table`: Either a valid table name or a `table_object`.
- As of MySQL 8.4.0, `table_object` is a JSON object literal that includes:
  - `table_name`: The name of the table to load.
  - `engine_attribute`: A JSON object literal that includes:
    - `sampling`: Only use with Lakehouse. If set to `true`, the default setting, Lakehouse Auto Parallel Load samples the data to infer the schema and collect statistics.

      If set to `false`, Lakehouse Auto Parallel Load performs a full scan to infer the schema and collect statistics. Depending on the size of the data, this can take a long time.

      Auto Parallel Load uses the inferred schema to generate `CREATE TABLE` statements. The statistics are used to estimate storage requirements and load times. See: About Lakehouse Auto Parallel Load Schema Inference.
    - For `dialect` and `file`, see: Section 4.3.6, “Lakehouse External Table Syntax”.
  - Use one or other of the following, but not both. The use of both parameters will throw an error.
    - `columns`: An optional JSON array of `column_name` to include in the load.
    - `exclude_columns`: An optional JSON array of `column_name` to exclude from the load.

Deprecated Auto Parallel Load Syntax

Before MySQL 8.4.0, the following syntax will be deprecated in a future release.

```
mysql> CALL sys.HEATWAVE_LOAD(db_list, options);

db_list: {JSON_ARRAY(["schema_name","schema_name"] ...)}

options: {JSON_OBJECT(keyvalue[, keyvalue]...)|NULL}
keyvalue:
{
```

```
  "mode",{"normal"|"dryrun"}
  |"output",{"normal"|"compact"|"silent"|"help"}
  |"sql_mode","sql_mode"
  |"policy",{"disable_unsupported_columns"|"not_disable_unsupported_columns"}
  |"exclude_list",JSON_ARRAY(schema_name_1, schema_name_2.table_name_1, schema_name_3.table_name_2.colu
  |"set_load_parallelism",{true|false}
  |"auto_enc",JSON_OBJECT("mode",{"off"|"check"})
  |"external_tables",JSON_ARRAY(db_object [, db_object]... )
}

db_object: {JSON_OBJECT("key","value"[,"key","value"] ...)}
keyvalue:
{
  "db_name": "name",
  |"tables": JSON_ARRAY(table [, table] ...)
}

table: {JSON_OBJECT("key","value"[,"key","value"] ...)}
keyvalue:
{
  "table_name": "name"
  |"sampling": {true|false}
  |"dialect": {dialect_section}
  |"file": JSON_ARRAY(file_section [, file_section]...)
}
```

`db_list` specifies the schemas to load. The list is a JSON array and requires one or more valid schema names. An empty array is permitted to view the Auto Parallel Load command-line help.

Auto Parallel Load `options` is a JSON object literal. The following options will be deprecated in a future release:

- `exclude_list`: Defines a list of schemas, tables, and columns to exclude from the load script. Names must be fully qualified without backticks.

  Use `db_object` with `tables`, `exclude_tables`, `columns` or `exclude_columns` instead.

  Auto Parallel Load automatically excludes database objects that cannot be offloaded, according to the default `policy` setting. These objects need not be specified explicitly in the exclude list. System schemas, non-`InnoDB` tables, tables that are already loaded in MySQL HeatWave, and columns explicitly defined as `NOT SECONDARY` are automatically excluded.

- `external_tables`: non-InnoDB tables which do not store any data, but refer to data stored externally. For the `external_tables` syntax, see: Section 4.3.8.5, “Use Lakehouse Auto Parallel Load with external_tables Option”.

  Use `db_object` with `tables` or `exclude_tables` instead.

Quickstart: Load Data Using Lakehouse Auto Parallel Load Examples

- Load a single schema with default options.

```
mysql> CALL sys.heatwave_load(JSON_ARRAY("tpch"),NULL);
```

- Load multiple schemas with default options.

```
mysql> CALL sys.heatwave_load(JSON_ARRAY("tpch","airportdb","employees","sakila"),NULL);
```

- Load multiple schemas with the `not_disable_unsupported_columns` policy, which causes tables with unsupported columns to be excluded from the load operation. Unsupported columns are those with unsupported data types.

```
mysql> CALL sys.heatwave_load(JSON_ARRAY("tpch","airportdb","employees","sakila"),
      JSON_OBJECT("policy","not_disable_unsupported_columns"));
```

- Load multiple schemas, excluding specified tables and a particular column:

```
mysql> CALL sys.heatwave_load(JSON_ARRAY("tpch","airportdb"),
      JSON_OBJECT("exclude_list",JSON_ARRAY("tpch.orders","airportdb.employee.salary")));
```

- Load tables that begin with an “hw” prefix from a schema named `schema_customer_1`.

```
mysql> SET @exc_list = (SELECT JSON_OBJECT('exclude_list',
            JSON_ARRAYAGG(CONCAT(table_schema,'.',table_name)))
            FROM information_schema.tables
            WHERE table_schema = 'schema_customer_1'
            AND table_name NOT LIKE 'hw%');
mysql> CALL sys.heatwave_load(JSON_ARRAY('schema_customer_1'), @exc_list);
```

- Load all schemas with tables that start with an “hw” prefix.

```
mysql> SET @db_list = (SELECT json_arrayagg(schema_name) FROM information_schema.schemata);
mysql> SET @exc_list = (SELECT JSON_OBJECT('exclude_list',
      JSON_ARRAYAGG(CONCAT(table_schema,'.',table_name)))
      FROM information_schema.tables
      WHERE table_schema NOT IN
      ('mysql','information_schema', 'performance_schema','sys')
      AND table_name NOT LIKE 'hw%');
mysql> CALL sys.heatwave_load(@db_list, @exc_list);
```

  You can check `db_list` and `exc_list` using `SELECT JSON_PRETTY(@db_list);` and `SELECT JSON_PRETTY(@exc_list);`

- Call Auto Parallel Load from a stored procedure:

```
DROP PROCEDURE IF EXISTS auto_load_wrapper;
DELIMITER //
CREATE PROCEDURE auto_load_wrapper()
BEGIN
-- AUTOMATED INPUT
SET @db_list = (SELECT JSON_ARRAYAGG(schema_name) FROM information_schema.schemata);
SET @exc_list = (SELECT JSON_ARRAYAGG(CONCAT(table_schema,'.',table_name))
FROM information_schema.tables WHERE table_schema = "db0");

CALL sys.heatwave_load(@db_list, JSON_OBJECT("output","silent","exclude_list",
CAST(@exc_list AS JSON)));

-- CUSTOM OUTPUT
SELECT log as 'Unsupported objects' FROM sys.heatwave_autopilot_report WHERE type="warn"
AND stage="VERIFICATION" and log like "%Unsupported%";
SELECT Count(*) AS "Total Load commands Generated"
FROM sys.heatwave_autopilot_report WHERE type = "sql" ORDER BY id;

END //
DELIMITER ;

CALL auto_load_wrapper();
```

Use the Auto Parallel Load Command-Line Help

To view Auto Parallel Load command-line help, issue the following statement:

```
mysql> CALL sys.heatwave_load(JSON_ARRAY(),JSON_OBJECT("output","help"));
```

The command-line help provides usage documentation for the Auto Parallel Load utility.

10.1.2 HEATWAVE_UNLOAD

Review the appropriate syntax windows to unload data automatically.

This topic contains the following sections:

- Auto Unload Syntax for MySQL 8.4.0 and Later
- Auto Unload Syntax to Unload All Tables

- Auto Unload Syntax for MySQL 8.3.0-u2 and Earlier
- Parameter Descriptions
- See Also

Auto Unload Syntax for MySQL 8.4.0 and Later

MySQL 8.4.0 adds an `input_list` JSON array that replaces the `db_list` JSON array. This adds an include list to exactly specify the tables to unload, which eliminates the need to include a database and exclude unnecessary tables. The `input_list` is backwards compatible with `db_list`.

```
mysql> CALL sys.HEATWAVE_UNLOAD (input_list, options);

input_list: {JSON_ARRAY(input [,input] ...)}

options: {JSON_OBJECT(keyvalue[, keyvalue]...)|NULL}
keyvalue:
{
  "mode",{"normal"|"dryrun"}
  |"output",{"normal"|"silent"|"help"}
}

input: {'db_name' | db_object}

db_object: {JSON_OBJECT("key","value"[,"key","value"] ...)}
keyvalue:
{
  "db_name": "db_name"
  |"tables": JSON_ARRAY(table [, table] ...)
  |"exclude_tables": JSON_ARRAY(table [, table] ...)
}

table: {'table_name'}
```

Auto Unload Syntax to Unload All Tables

The following syntax options allow you to automatically unload all tables.

```
mysql> CALL sys.HEATWAVE_UNLOAD_all (options);

options: {JSON_OBJECT(keyvalue[, keyvalue]...)|NULL}
keyvalue:
{
  "only_user_loaded_tables",{true|false}
  |"output",{"normal"|"silent"}
}
```

Auto Unload Syntax for MySQL 8.3.0-u2 and Earlier

The following syntax options only apply to MySQL 8.3.0-u2 and earlier.

```
mysql> CALL sys.HEATWAVE_UNLOAD (db_list, options);

db_list: {JSON_ARRAY(["schema_name","schema_name"] ...)}

options: {JSON_OBJECT(keyvalue[, keyvalue]...)|NULL}
keyvalue:
{
  "mode",{"normal"|"dryrun"}
  |"output",{"normal"|"silent"|"help"}
  |"exclude_list",JSON_ARRAY(schema_name_1, schema_name_2.table_name_1, ...)
}
```

Parameter Descriptions

As of MySQL 8.4.0 use `input_list` to define what to unload. `input_list` is a JSON array and requires one or more valid `input` which can be either a valid schema name or a `db_object`. An

empty array is permitted to view the Auto Unload command-line help. This is backwards compatible with `db_list`.

Before MySQL 8.4.0, `db_list` specifies the schemas to unload. The list is a JSON array and requires one or more valid schema names. An empty array is permitted to view the Auto Unload command-line help.

Use key-value pairs in `JSON` format to specify parameters. MySQL HeatWave uses the default setting if there is no option setting. Use `NULL` to specify no arguments.

For syntax examples, see Auto Unload Examples.

Auto Unload *`options`* include:

- `mode`: Defines the Auto Unload operational mode. Permitted values are:
  - `normal`: The default. Generates and executes the unload script.
  - `dryrun`: Generates an unload script only. Auto Unload executes in `dryrun` mode automatically if the MySQL HeatWave Cluster is not active.
- `output`: Defines how Auto Unload produces output. Permitted values are:
  - `normal`: The default. Produces summarized output and sends it to `stdout` and to the `heatwave_autopilot_report` table. See Section 5.8.8, “Autopilot Report Table”.
  - `silent`: Sends output to the `heatwave_autopilot_report` table only. See Section 5.8.8, “Autopilot Report Table”. The `silent` output type is useful if human-readable output is not required; when the output is consumed by a script, for example. For an example of a stored procedure with an Auto Unload call that uses the `silent` output type, see Auto Unload Examples.
  - `help`: Displays Auto Unload command-line help.
- `exclude_list`: Defines a list of schemas and tables to exclude from the unload script. Names must be fully qualified without backticks.

  Do not use as of MySQL 8.4.0. Use `db_object` with `tables` or `exclude_tables` instead. `exclude_list` will be deprecated in a future release.

  Auto Unload automatically excludes tables that are loading, unloading or in recovery. This is when the `load_status` is one of the following: `NOLOAD_RPDGSTABSTATE`, `LOADING_RPDGSTABSTATE`, `UNLOADING_RPDGSTABSTATE` or `INRECOVERY_RPDGSTABSTATE`.
- `only_user_loaded_tables`: Only available to unload all tables. It is enabled by default. The procedure only unloads user tables. If `false`, the procedure unloads user and system tables.

The `db_object` is a JSON object literal that includes:

- `db_name`: The name of the database to load.
- Use one or other of the following, but not both. The use of both parameters causes an error.
  - `tables`: An optional JSON array of `table` to include in the load.
  - `exclude_tables`: As of MySQL 8.4.0, an optional JSON array of `table` to exclude from the load.
- `table`: A valid table name.

See Also

- Learn How to Unload Data Automatically

10.1.3 HEATWAVE_RELOAD

Reloads all tables. Available as of MySQL 8.4.0.

```
mysql> CALL sys.HEATWAVE_RELOAD(options);

options: {JSON_OBJECT(keyvalue[, keyvalue]...)|NULL}
keyvalue:
{
  "only_user_loaded_tables",{true|false}
  |"output",{"normal"|"silent"}
}
```

Use key-value pairs in `JSON` format to specify *`options`*. MySQL HeatWave uses the default setting if there is no defined option. Use `NULL` to specify no arguments.

The *`options`* include:

- `only_user_loaded_tables`: Enabled by default. The procedure only reloads user tables. If `false`, the procedure reloads user and system tables.
- `output`: Defines where to send the output. Permitted values are:
  - `normal`: The default. Produces summarized output and sends it to `stdout` and to the `heatwave_autopilot_report` table. See Section 5.8.8, “Autopilot Report Table”.
  - `silent`: Sends output to the `heatwave_autopilot_report` table only. See Section 5.8.8, “Autopilot Report Table”. The `silent` output type is useful if human-readable output is not required; when the output is consumed by a script, for example.

Syntax Examples

- Reload tables with default routine option values:

```
mysql> CALL sys.HEATWAVE_RELOAD(NULL);
```

```
mysql> CALL sys.HEATWAVE_RELOAD(JSON_OBJECT());
```

- Reload all user-loaded and automatically-loaded tables:

```
mysql> CALL sys.HEATWAVE_RELOAD(JSON_OBJECT("only_user_loaded_tables",false));
```

- Reload user-loaded tables with the `output` option set to `silent`:

```
mysql> CALL sys.HEATWAVE_RELOAD(JSON_OBJECT("output","silent"));
```

- Reload all user-loaded and automatically-loaded tables with the `output` option set to `silent`:

```
mysql> CALL sys.HEATWAVE_RELOAD(JSON_OBJECT("only_user_loaded_tables",false,"output","silent"));
```

To learn more about user-loaded and automatically-loaded tables, see Automatic Loading and Unloading of DB System Tables and Partitions.

10.1.4 Asynchronous Task Routines

The sections in this topic describe the different routines that you can use to create, monitor, and terminate asynchronous tasks.

For applications that have multiple application users, and there is no direct mapping between MySQL users and application users, you can use the application-level (`app`) variants of the asynchronous task routines.

Asynchronous Task Routine routines reside in `mysql_tasks` schema.

10.1.4.1 execute_prepared_stmt_async

The `execute_prepared_stmt_async` routine creates a task that executes SQL statements asynchronously within an event stored in the `schema_name` schema.

The application-level variant for this routine is `execute_prepared_stmt_from_app_async`.

This topic contains the following sections:

- Syntax
- Syntax Example
- See Also

Syntax

```
mysql> CALL mysql_tasks.execute_prepared_stmt_async (
IN 'sql_statements' TEXT,
IN 'schema_name' VARCHAR(255),
IN 'task_name' VARCHAR(255),
IN 'task_data' JSON,
OUT @task_id VARCHAR(36));
```

Following are `execute_prepared_stmt_async` parameters:

- `sql_statements` (TEXT): specifies the SQL statements that you want to run asynchronously represented as a string value separated by a semi-colon (`;`).
- `schema_name` (VARCHAR(255)): specifies the schema to run the given SQL statements. If the schema is already set for the current session, you can set this parameter to `NULL` to use the current session schema.
- `task_name` (VARCHAR(255)): specifies a name for the asynchronous task. If set to `NULL`, `execute_prepared_stmt_async` is used as the default task name.
- `task_data` (JSON): specifies additional metadata related to the task, represented in JSON format. You can use this parameter to store metadata that you might find relevant for the task, such as parameter values that you passed to the task. It is optional to add a value for this parameter. If there is no metadata, set this parameter to `NULL`.
- `@task_id` (VARCHAR(36)): output parameter that returns the unique identifier (UUID) of the created task. Use this ID to monitor the task status or manage the task.

Syntax Example

```
mysql> CALL mysql_tasks.execute_prepared_stmt_async('DO SLEEP(5);
SELECT JSON_OBJECT("Async Task completed at", now()) into @task_result', NULL, 'DemoAsyncTask', NULL, @task
```

In this example, the first SQL statement `DO SLEEP(5)` simulates a time-consuming task. This can be any long-running task.

The second statement generates an output that is stored in the reserved session variable `@task_result`. You can optionally use the reserved session variable `@task_result` to add a custom task status message for the asynchronous task, or to store an output generated by an SQL statement that you are running asynchronously. The value that you want to store in `@task_result` must be in JSON format. Any value stored in `@task_result` is included in the asynchronous task status `data` field.

The task ID generated for the asynchronous task that is returned by the routine is stored in the session variable `@task_id`.

See Also

Running tasks Asynchronously.

10.1.4.2 execute_prepared_stmt_from_app_async

The `execute_prepared_stmt_from_app_async` routine creates an asynchronous task associated with a specific application user. Application-level asynchronous task lets you filter and track tasks for individual application users using an application-managed user ID. This routine includes options that define the structure and format used for storing task status and logs data.

This topic contains the following sections:

- Syntax
- Syntax Example
- See Also

Syntax

```
mysql> CALL mysql_tasks.execute_prepared_stmt_from_app_async (
IN 'sql_statements' TEXT,
IN 'app_user_id' VARCHAR(255),
IN 'schema_name' VARCHAR(255),
IN 'task_type' VARCHAR(80),
IN 'task_name' VARCHAR(255),
IN 'task_data' JSON,
IN 'data_json_schema' JSON,
IN 'log_data_json_schema' JSON,
IN 'progress_monitor_sql_statements' TEXT,
IN progress_monitor_refresh_period DECIMAL(5,2),
OUT @task_id VARCHAR(36));
```

Following are `execute_prepared_stmt_async` parameters:

- `sql_statements` (TEXT): specifies the SQL statements that you want to run asynchronously represented as a string value separated by a semi-colon (`;`).
- `app_user_id` (VARCHAR(255)): specifies the unique ID of the application user initiating the task.
- `schema_name` (VARCHAR(255)): specifies the schema to run the given SQL statements. If the schema is already set for the current session, you can set this parameter to `NULL` to use the current session schema.
- `task_type` (VARCHAR(80)): specifies a task type that you can later use to filter the tasks based on the specified task type. If the value is `NULL`, tasks of all types are included.
- `task_name` (VARCHAR(255)): specifies a name for the asynchronous task. If set to `NULL`, `execute_prepared_stmt_from_app_async` is used as the default task name.
- `task_data` (JSON): specifies additional metadata related to the task, represented in JSON format. You can use this parameter to store metadata that you might find relevant for the task, such as parameter values that you passed to the task. It is optional to add a value for this parameter. If there is no metadata, set this parameter to `NULL`.
- `data_json_schema` (JSON): specifies the JSON schema that defines the format and data types used for the `task_data` field included in the asynchronous task status. You can set this to `NULL` if no specific structure definition is required for the task data.
- `log_data_json_schema` (JSON): specifies the JSON schema that defines the format and data types used for the `data` field included in the task logs. You can set this parameter to `NULL` if no specific structure definition is required for the task log data.

- `progress_monitor_sql_statements` (TEXT): specifies an optional SQL statement to monitor the task progress. Set this parameter to `NULL` if progress monitoring is not required.
- `progress_monitor_refresh_period` (DECIMAL(5,2)): specifies the refresh interval in seconds. If set to `NULL`, the refresh period is set to 5 seconds by default.
- `@task_id` (VARCHAR(36)): output parameter that returns the unique identifier of the created task. Use this ID is to monitor the task status or manage the task.

Syntax Example

```
mysql> CALL mysql_tasks.execute_prepared_stmt_from_app_async(
'DO SLEEP(5); SELECT JSON_OBJECT("Task result", "Finished...") into @task_result',
'app_user_01',
'demo_db',
'demo task type',
'demo task name',
JSON_OBJECT('sleep', 5, 'refresh_period', 1),
'{
  "type": "object",
  "properties": {
      "sleep": {
          "type": "integer"
      },
      "refresh_period": {
          "type": "integer"
      }
  }
}',
'{
  "type": "object",
  "properties": {
    "iteration": {
      "type": "integer"
    },
    "Task result": {
      "type": "string"
    },
    "connection_id": {
      "type": "integer"
    }
  },
  "additionalProperties": false,
  "oneOf": [
    { "required": ["iteration"] },
    { "required": ["Task result"] },
    { "required": ["connection_id"] }
  ]
}',
@progress_monitor_sql_statements,
1,
@task_id);
```

Where:

```
mysql> SET @progress_monitor_sql_statements = '
  SET @my_counter = coalesce(@my_counter, 0) + 1;
  SET @my_progress = @my_counter * 100 / 5;
  CALL mysql_tasks.add_task_log(@task_id, "Progress update", JSON_OBJECT("iteration", @my_counter), @my_pro
```

In this example:

- The first SQL statement `DO SLEEP(5)` simulates a time-consuming task. The second statement `SELECT JSON_OBJECT("Task result", "Finished...") into @task_result` generates an output that is stored in the reserved session variable `@task_result`.

  You can optionally use the reserved session variable `@task_result` to add a custom task status message for the asynchronous task, or to store an output generated by an SQL statement that you

are running asynchronously. The value that you want to store in `@task_result` must be in JSON format. Any value stored in `@task_result` is included in the asynchronous task status `data` field. You can view the task status using the `app_task_status` routine.

- The next few parameter values specify the application-managed user ID `app_user_01`, database to use `demo_db`, task type `demo task type`, and task name `demo task name`.
- The JSON value `JSON_OBJECT('sleep', 5, 'refresh_period', 1)` specifies the additional metadata to include in the `task_data` field in the task status. This example stores different parameter values that were used during asynchronous task creation as additional task metadata, so that these values can be referred to at a later time. You can view the task status using the `app_task_status` routine.
- The following JSON schema defines the structure of the additional metadata to include in `task_data` field in the task status:

```
{
  "type": "object",
  "properties": {
      "sleep": {
          "type": "integer"
      },
      "refresh_period": {
          "type": "integer"
      }
  }
}
```

- The JSON following JSON schema defines the structure of the additional data to include in `data` field in the task logs:

```
{
  "type": "object",
  "properties": {
    "iteration": {
      "type": "integer"
    },
    "Task result": {
      "type": "string"
    },
    "connection_id": {
      "type": "integer"
    }
  },
  "additionalProperties": false,
  "oneOf": [
    { "required": ["iteration"] },
    { "required": ["Task result"] },
    { "required": ["connection_id"] }
  ]
}
```

This example specifies a strict schema for logging data. Any attempt to log a property that is not included in the specified schema would cause the asynchronous task to fail. Also, in this example, all specified properties are added to the task logs, one property per log. You can view the task logs using the `app_task_logs` routine.

- The `@progress_monitor_sql_statements` session variable specifies the SQL statements to use for monitoring the task progress. The following progress monitoring statements are used in this example:

```
  SET @my_counter = coalesce(@my_counter, 0) + 1;
  SET @my_progress = @my_counter * 100 / 5;
  CALL mysql_tasks.add_task_log(@task_id, "Progress update", JSON_OBJECT("iteration", @my_counter), @
```

  - The first two statements increase the value of a counter stored in the session variable `@my_counter` which is then used to calculate the progress percentage stored in the

`@my_progress` session variable every second considering that the task takes 5 seconds to complete.

- The last statements in this example logs the progress of the task using the `add_task_log` routine.

  In this example wherein the task runs for 5 seconds, progress monitoring statements are used to track and log the task progress after every 1 second. However, progress tracking, data logging, and refresh period would vary depending on the duration and complexity of task you are running.

- The refresh period is set to `1`. Therefore, the specified progress monitoring statements run after every 1 second.
- Finally, the task ID generated for the asynchronous task that is returned by the routine is stored in the session variable `@task_id`.

See Also

Running tasks Asynchronously.

10.1.4.3 add_task_log

The `add_task_log` routine add a log entry for the task with the specified ID. You can use this routine with an application-level task as well and it does not require an application-managed user ID.

Syntax

```
mysql> CALL mysql_tasks.add_task_log(
  IN 'task_id' VARCHAR(36),
  IN 'message' VARCHAR(2000),
  IN 'data' JSON,
  IN progress SMALLINT,
  IN 'status' ENUM
  );
```

Following are `add_task_log` parameters:

- `task_id` VARCHAR(36): specifies the unique task ID (`UUID`) of the task for which you want to add logs.
- `message` VARCHAR(2000): specifies the log message to add.
- `data` JSON: specifies additional log data. If there is no additional data, set to `NULL`.
- `progress` SMALLINT: specifies the task progress percentage from `0` to `100`.
- `status` ENUM: specifies the current task status. Possible values are: `SCHEDULED`, `RUNNING`, `COMPLETED`, `ERROR`, or `CANCELLED`.

Syntax Example

See the `execute_prepared_stmt_from_app_async` routine example.

See Also

Running tasks Asynchronously.

10.1.4.4 task

The `task` routine returns information about a single task.

The application-level variant for this routine is `app_task`.

This topic contains the following sections:

- Syntax
- Syntax Example
- See Also

Syntax

```
mysql> SELECT mysql_tasks.task(IN 'task_id' VARCHAR(36));
```

The `task_id` (VARCHAR(36)) parameter specifies the unique task ID (`UUID`) of the task whose information is required.

Syntax Example

```
mysql> SELECT JSON_PRETTY(mysql_tasks.task(@task_id));
```

This example retrieves all the task information associated with the task ID that is stored in the session variable `@task_id`. The output is similar to the following:

```
| {
  "id": "17efb56b-52a1-11f0-bca4-020017205654",
  "data": {
    "mysqlMetadata": {
      "autoGc": true,
      "events": [
        "`demo_db`.`17ef9cf8-52a1-11f0-bca4-020017205654`"
      ]
    }
  },
  "name": "DemoAsyncTask",
  "alias": "Thu-7",
  "task_type": "Async_SQL",
  "connection_id": 65220,
  "data_json_schema": {
    "type": "object",
    "required": [
      "mysqlMetadata"
    ],
    "properties": {
      "mysqlMetadata": {
        "type": "object",
        "required": [
          "events",
          "autoGc"
        ],
        "properties": {
          "autoGc": {
            "type": "boolean"
          },
          "events": {
            "type": "array",
            "items": {
              "type": "string"
            },
            "minItems": 1,
            "uniqueItems": true
          }
        }
      }
    }
  },
  "log_data_json_schema": null
} |
```

See Also

Running tasks Asynchronously.

10.1.4.5 app_task

The `app_task` routine returns information about a single application task.

This topic contains the following sections:

- Syntax
- Syntax Example
- See Also

Syntax

```
mysql> SELECT mysql_tasks.app_task(
  IN 'app_user_id' VARCHAR(255),
  IN 'task_id' VARCHAR(36));
```

Following are `app_task` parameters:

- `app_user_id` (VARCHAR(255)): specifies the unique ID of the application user who owns the task.
- `task_id` (VARCHAR(36)): specifies the unique task ID (`UUID`) of the application task whose information is required.

Syntax Example

```
mysql> SELECT JSON_PRETTY(mysql_tasks.app_task('app_user_01',@task_id));
```

The output is similar to the following:

```
| {
  "id": "a516b352-574c-11f0-9604-020017205654",
  "data": {
    "sleep": 5,
    "mysqlMetadata": {
      "autoGc": true,
      "events": [
        "`demo_db`.`a516a336-574c-11f0-9604-020017205654`",
        "`demo_db`.`a516a3f8-574c-11f0-9604-020017205654`"
      ]
    },
    "refresh_period": 1
  },
  "name": "demo task name",
  "alias": "Wed-3",
  "task_type": "demo task type",
  "connection_id": 1650,
  "data_json_schema": {
    "type": "object",
    "required": null,
    "properties": {
      "sleep": {
        "type": "integer"
      },
      "mysqlMetadata": {
        "type": "object",
        "required": [
          "events",
          "autoGc"
        ],
        "properties": {
          "autoGc": {
            "type": "boolean"
          },
          "events": {
            "type": "array",
            "items": {
              "type": "string"
```

```
            },
            "minItems": 1,
            "uniqueItems": true
          }
        }
      },
      "refresh_period": {
        "type": "integer"
      }
    }
  },
  "log_data_json_schema": {
    "type": "object",
    "oneOf": [
      {
        "required": [
          "iteration"
        ]
      },
      {
        "required": [
          "Task result"
        ]
      },
      {
        "required": [
          "connection_id"
        ]
      }
    ],
    "properties": {
      "iteration": {
        "type": "integer"
      },
      "Task result": {
        "type": "string"
      },
      "connection_id": {
        "type": "integer"
      }
    },
    "additionalProperties": false
  }
} |
```

This example retrieves all the information associated with the specified application-managed user ID and task ID stored in the session variable `@task_id`.

See Also

Running tasks Asynchronously.

10.1.4.6 task_status

The `task_status` routine returns the task status.

The application-level variant for this routine is `app_task_status`.

This topic contains the following sections:

- Syntax
- Syntax Example
- See Also

Syntax

```
mysql> SELECT mysql_tasks.task_status(IN 'task_id' VARCHAR(36));
```

The `task_id` (VARCHAR 36) parameter specifies the unique task ID (`UUID`) of the task whose information is required.

Syntax Example

```
mysql> SELECT JSON_PRETTY(mysql_tasks.task_status(@task_id));
```

This example returns the status of the asynchronous task with task ID stored in the session variable `@task_id`. The output is similar to the following:

```
  | {
  "id": "17efb56b-52a1-11f0-bca4-020017205654",
  "data": {
    "Async Task completed at": "2025-06-26 15:20:34.000000"
  },
  "name": "DemoAsyncTask",
  "alias": "Thu-7",
  "status": "COMPLETED",
  "message": "Execution finished.",
  "progress": 100,
  "row_hash": "fa701b650afd8638049f8fd436a500f5",
  "task_data": {
    "mysqlMetadata": {
      "autoGc": true,
      "events": [
        "`demo_db`.`17ef9cf8-52a1-11f0-bca4-020017205654`"
      ]
    }
  },
  "task_type": "Async_SQL",
  "server_uuid": "5172a252-8a06-11ef-b3bb-02001727ecef",
  "progress_bar": "##########",
  "connection_id": 65220,
  "starting_time": "2025-06-26 15:20:29.000000",
  "scheduled_time": "2025-06-26 15:20:29.000000",
  "estimated_remaining_time": 0.00000,
  "estimated_completion_time": "2025-06-26 15:20:34.000000"
} |
```

See Also

Running tasks Asynchronously.

10.1.4.7 app_task_status

The `app_task_status` routine returns information about a single task.

This topic contains the following sections:

- Syntax
- Syntax Example
- See Also

Syntax

```
mysql> SELECT mysql_tasks.app_task_status(
  IN 'app_user_id' VARCHAR(255),
  IN 'task_id' VARCHAR (36);
```

Following are `app_task_status` parameters:

- `app_user_id` (VARCHAR(255)): specifies the the application-managed user ID of the user who created the task.

- `task_id` (VARCHAR (36)): specifies the unique task ID (`UUID`) of the task for which the status information is required.

Syntax Example

```
mysql> SELECT JSON_PRETTY(mysql_tasks.app_task_status('app_user_01', @task_id));
```

The output is similar to the following:

```
| {
  "id": "a516b352-574c-11f0-9604-020017205654",
  "data": {
    "Task result": "Finished..."
  },
  "name": "demo task name",
  "alias": "Wed-3",
  "status": "COMPLETED",
  "message": "Execution finished.",
  "progress": 100,
  "row_hash": "de3195fc0e2427d9b91cf03b920b96f2",
  "task_data": {
    "sleep": 5,
    "mysqlMetadata": {
      "autoGc": true,
      "events": [
        "`demo_db`.`a516a336-574c-11f0-9604-020017205654`",
        "`demo_db`.`a516a3f8-574c-11f0-9604-020017205654`"
      ]
    },
    "refresh_period": 1
  },
  "task_type": "demo task type",
  "server_uuid": "5172a252-8a06-11ef-b3bb-02001727ecef",
  "progress_bar": "##########",
  "connection_id": 1650,
  "starting_time": "2025-07-02 13:58:35.000000",
  "scheduled_time": "2025-07-02 13:58:35.000000",
  "estimated_remaining_time": 0.00000,
  "estimated_completion_time": "2025-07-02 13:58:40.000000"
} |
```

This example returns the current status of the asynchronous task with the specified application-managed user ID and task ID stored in the session variable `@task_id`.

See Also

Running tasks Asynchronously.

10.1.4.8 task_list

The `task_list` routine returns a paginated list of asynchronous tasks.

The application-level variant for this routine is `app_task_list`.

This topic contains the following sections:

- Syntax
- Syntax Example
- See Also

Syntax

```
mysql> SELECT mysql_tasks.task_list(
  IN 'task_type' VARCHAR(80),
  IN offset INT UNSIGNED,
  IN limit INT UNSIGNED);
```

Following are `task_list` parameters:

- `task_type` (VARCHAR(80)): filters the tasks based on a specific task type. If the value is `NULL`, tasks of all types are included. Task type for tasks created using the `execute_prepared_stmt_async` procedure is `Async_SQL`. For asynchronous `VECTOR_STORE_LOAD` tasks, the task type is `GenAI_Load`.
- `offset` (INT UNSIGNED): the number of rows to skip. The offset value is used for pagination. If the value is `NULL`, the offset defaults to `0`, and the result is displayed starting from the first row.
- `limit` (INT UNSIGNED): the maximum number of task records included in the list of tasks. If the value is `NULL`, the limit defaults to `20`, and up to 20 rows are displayed.

Syntax Example

```
mysql> JSON_PRETTY(mysql_tasks.task_list('Async_SQL', 2, 2));
```

The output is similar to the following:

```
| [
  {
    "id": "4605cf38-527b-11f0-bca4-020017205654",
    "data": {
      "mysqlMetadata": {
        "autoGc": true,
        "events": [
          "`demo_db`.`4605c15a-527b-11f0-bca4-020017205654`"
        ]
      }
    },
    "name": "DemoAsyncTask",
    "task_type": "Async_SQL",
    "connection_id": 63582
  },
  {
    "id": "18398c40-527b-11f0-bca4-020017205654",
    "data": {
      "mysqlMetadata": {
        "autoGc": true,
        "events": [
          "`demo_db`.`18397be6-527b-11f0-bca4-020017205654`"
        ]
      }
    },
    "name": "DemoAsyncTask",
    "task_type": "Async_SQL",
    "connection_id": 63576
  }
] |
```

This example lists two tasks of type `Async_SQL`.

See Also

Running tasks Asynchronously.

10.1.4.9 app_task_list

The `app_task_list` routine returns a paginated list of asynchronous tasks associated with an application user.

This topic contains the following sections:

- Syntax
- Syntax Example

- See Also

Syntax

```
mysql> SELECT mysql_tasks.app_task_list(
  IN 'app_user_id' VARCHAR(255),
  IN 'task_type' VARCHAR(80),
  IN offset IN UNSIGNED,
  IN limit INT UNSIGNED);
```

Following are `app_task_list` parameters:

- `app_user_id` (VARCHAR(255)): specifies the application-managed user ID to use for filtering the tasks.
- `task_type` (VARCHAR(80)): specifies the task type to use for filtering the tasks. If the value is `NULL`, tasks of all types are listed.
- `offset` (INT UNSIGNED): specifies the initial number of tasks to skip. The offset value is used for pagination. If the value is `NULL`, the offset defaults to `0`, and the result is displayed starting from the first row.
- `limit` (INT UNSIGNED): specifies the maximum number of task records to include. If the value is `NULL`, the limit defaults to `20`, and up to 20 rows are displayed.

Syntax Example

```
mysql> SELECT JSON_PRETTY(mysql_tasks.app_task_list('app_user_01','demo task type',0,1));
```

The output is similar to the following:

```
| [
  {
    "id": "a516b352-574c-11f0-9604-020017205654",
    "data": {
      "sleep": 5,
      "mysqlMetadata": {
        "autoGc": true,
        "events": [
          "`demo_db`.`a516a336-574c-11f0-9604-020017205654`",
          "`demo_db`.`a516a3f8-574c-11f0-9604-020017205654`"
        ]
      },
      "refresh_period": 1
    },
    "name": "demo task name",
    "task_type": "demo task type",
    "connection_id": 1650
  }
] |
```

This example lists one application task of the specified task type.

See Also

Running tasks Asynchronously.

10.1.4.10 task_logs

The `task_logs` routine returns a list of logs associated with a task.

The application-level variant for this routine is `app_task_logs`.

This topic contains the following sections:

- Syntax

- Syntax Example
- See Also

Syntax

```
mysql> SELECT mysql_tasks.task_logs(
  IN 'task_id' VARCHAR(36),
  IN 'newer_than_log_time' TIMESTAMP(6));
```

Following are `task_logs` parameters:

- `task_id` (VARCHAR(36)): specifies the unique task ID (`UUID`) of the task whose log entries are required.
- `newer_than_log_time` (TIMESTAMP(6)): all the log entries created after the given timestamp are included. If the value is `NULL`, all the log entries for a given task ID are displayed.

Syntax Example

```
mysql> SELECT JSON_PRETTY(mysql_tasks.task_logs(@task_id, '2025-06-26 10:00:00'));
```

The output is similar to the following:

```
| [
  {
    "id": "4d7dcd7a-5298-11f0-bca4-020017205654",
    "data": null,
    "status": "SCHEDULED",
    "message": "Task created by user.",
    "task_id": "4d7bf6ec-5298-11f0-bca4-020017205654",
    "log_time": "2025-06-26 14:17:33.708391",
    "progress": 0
  },
  {
    "id": "4d7e5319-5298-11f0-bca4-020017205654",
    "data": null,
    "status": "RUNNING",
    "message": "Event execution started...",
    "task_id": "4d7bf6ec-5298-11f0-bca4-020017205654",
    "log_time": "2025-06-26 14:17:33.711716",
    "progress": 0
  },
  {
    "id": "507a49ad-5298-11f0-bca4-020017205654",
    "data": {
      "Async Task completed at": "2025-06-26 14:17:38.000000"
    },
    "status": "COMPLETED",
    "message": "Execution finished.",
    "task_id": "4d7bf6ec-5298-11f0-bca4-020017205654",
    "log_time": "2025-06-26 14:17:38.718451",
    "progress": 100
  }
] |
```

This example retrieves the logs added for the task ID stored in the session variable `@task_id` after `2025-06-26 10:00:00`.

See Also

Running tasks Asynchronously.

10.1.4.11 app_task_logs

The `app_task_logs` routine returns a list of logs associated with an application-level task.

This topic contains the following sections:

- Syntax
- Syntax Example
- See Also

Syntax

```
mysql> SELECT mysql_tasks.app_task_logs(
  IN 'app_user_id' VARCHAR(255),
  IN 'task_id' VARCHAR(36),
  IN 'newer_than_log_time' TIMESTAMP(6));
```

Following are `app_task_logs` parameters:

- `app_user_id` VARCHAR(255): specifies the application-managed user ID of the user who owns the task.
- `task_id` (VARCHAR(36)): specifies the unique task ID (`UUID`) of the task whose log entries are required.
- `newer_than_log_time` TIMESTAMP(6): all the log entries created after the specified timestamp are included. If the value is `NULL`, all the log entries for the specified task ID are displayed.

Syntax Example

```
mysql> SELECT JSON_PRETTY(mysql_tasks.app_task_logs('app_user_01',@task_id,'2025-07-02 10:00:00'));
```

The output is similar to the following:

```
| [
  {
    "id": "a51885db-574c-11f0-9604-020017205654",
    "data": null,
    "status": "SCHEDULED",
    "message": "Task created by user.",
    "task_id": "a516b352-574c-11f0-9604-020017205654",
    "log_time": "2025-07-02 13:58:34.746672",
    "progress": 0
  },
  {
    "id": "a5198943-574c-11f0-9604-020017205654",
    "data": null,
    "status": "RUNNING",
    "message": "Event execution started...",
    "task_id": "a516b352-574c-11f0-9604-020017205654",
    "log_time": "2025-07-02 13:58:34.753164",
    "progress": 0
  },
  {
    "id": "a51a219f-574c-11f0-9604-020017205654",
    "data": {
      "connection_id": 1651
    },
    "status": "RUNNING",
    "message": "Progress monitor started.",
    "task_id": "a516b352-574c-11f0-9604-020017205654",
    "log_time": "2025-07-02 13:58:34.757277",
    "progress": 0
  },
  {
    "id": "a51aad03-574c-11f0-9604-020017205654",
    "data": {
      "iteration": 1
    },
    "status": "RUNNING",
    "message": "Progress update",
    "task_id": "a516b352-574c-11f0-9604-020017205654",
    "log_time": "2025-07-02 13:58:34.760739",
    "progress": 20
```

```
  },
  {
    "id": "a5b3d539-574c-11f0-9604-020017205654",
    "data": {
      "iteration": 2
    },
    "status": "RUNNING",
    "message": "Progress update",
    "task_id": "a516b352-574c-11f0-9604-020017205654",
    "log_time": "2025-07-02 13:58:35.764511",
    "progress": 40
  },
  {
    "id": "a64cf56c-574c-11f0-9604-020017205654",
    "data": {
      "iteration": 3
    },
    "status": "RUNNING",
    "message": "Progress update",
    "task_id": "a516b352-574c-11f0-9604-020017205654",
    "log_time": "2025-07-02 13:58:36.767959",
    "progress": 60
  },
  {
    "id": "a6e61404-574c-11f0-9604-020017205654",
    "data": {
      "iteration": 4
    },
    "status": "RUNNING",
    "message": "Progress update",
    "task_id": "a516b352-574c-11f0-9604-020017205654",
    "log_time": "2025-07-02 13:58:37.771507",
    "progress": 80
  },
  {
    "id": "a77f3649-574c-11f0-9604-020017205654",
    "data": {
      "iteration": 5
    },
    "status": "RUNNING",
    "message": "Progress update",
    "task_id": "a516b352-574c-11f0-9604-020017205654",
    "log_time": "2025-07-02 13:58:38.775084",
    "progress": 100
  },
  {
    "id": "a815b093-574c-11f0-9604-020017205654",
    "data": {
      "Task result": "Finished..."
    },
    "status": "COMPLETED",
    "message": "Execution finished.",
    "task_id": "a516b352-574c-11f0-9604-020017205654",
    "log_time": "2025-07-02 13:58:39.761243",
    "progress": 100
  }
] |
```


This example lists all the log entries for the task associated with the specified application-managed user ID, task ID stored in the session variable `@task_id`, and created after the specified date and time.

See Also

Running tasks Asynchronously.

10.1.4.12 task_status_brief

The `task_status_brief` routine returns the brief status of a task, including a subset of information about the task and its last log entry.

The application-level variant for this routine is `app_task_status_brief`.

This topic contains the following sections:

- Syntax
- Syntax Example
- See Also

Syntax

```
mysql> SELECT mysql_tasks.task_status_brief(IN 'task_id' VARCHAR(36));
```

The `task_id` (VARCHAR(36)) parameter specifies the unique task ID (`UUID`) of the task whose log entries are required.

Syntax Example

```
mysql> SELECT JSON_PRETTY(mysql_tasks.task_status_brief(@task_id));
```

The output is similar to the following:

```
  | {
  "data": {
    "Async Task completed at": "2025-06-26 14:17:38.000000"
  },
  "status": "COMPLETED",
  "message": "Execution finished.",
  "progress": 100
} |
```

This example returns basic information related to the current status of the asynchronous task with task ID stored in the session variable `@task_id`.

See Also

Running tasks Asynchronously.

10.1.4.13 app_task_status_brief

The `app_task_status_brief` routine returns the brief status of an application task, including a subset of information about the task and its last log entry.

This topic contains the following sections:

- Syntax
- Syntax Example
- See Also

Syntax

```
mysql> SELECT mysql_tasks.app_task_status_brief(
  IN 'app_user_id' VARCHAR(255),
  IN 'task_id' VARCHAR(36));
```

Following are `app_task_status_brief` parameters:

- `app_user_id` (VARCHAR(255)): specifies the unique ID of the application user who owns the task.
- `task_id` (VARCHAR(36)): specifies the unique task ID (`UUID`) of the task for which the current task status is required.

Syntax Example

```
mysql> SELECT JSON_PRETTY(mysql_tasks.app_task_status_brief('app_user_01', @task_id));
```

The output is similar to the following:

```
+-----------------------------------------------------------------------------------------------------------------------
| JSON_PRETTY(mysql_tasks.app_task_status_brief('app_user_01', @task_id))
+-----------------------------------------------------------------------------------------------------------------------
| {
  "data": {
    "Task result": "Finished..."
  },
  "status": "COMPLETED",
  "message": "Execution finished.",
  "progress": 100
} |
+-----------------------------------------------------------------------------------------------------------------------
```

This example returns basic information related to the current status of the asynchronous task associated with the specified application-managed user ID and task ID stored in the session variable `@task_id`.

See Also

Running tasks Asynchronously.

10.1.4.14 task_status_list

The `task_status_list` routine returns a paginated list of task statuses.

The application-level variant for this routine is `app_task_status_list`.

This topic contains the following sections:

- Syntax
- Syntax Example
- See Also

Syntax

```
mysql> SELECT mysql_tasks.task_status_list(
  IN 'task_type' VARCHAR(80),
  IN offset INT UNSIGNED,
  IN limit INT UNSIGNGED);
```

Following are `task_status_list` parameters:

- `task_type` (VARCHAR(80)): filters the tasks based on a specific task type. If the value is `NULL`, tasks of all types are included. Task type for tasks created using the `execute_prepared_stmt_async` procedure is `Async_SQL`. For asynchronous `VECTOR_STORE_LOAD` tasks, the task type is `GenAI_Load`.
- `offset` (INT UNSIGNED): specifies the initial number of rows to skip. The offset value is used for pagination.
- `limit` (INT UNSIGNED): specifies the maximum number of task records to include.

Syntax Example

```
mysql> SELECT JSON_PRETTY(mysql_tasks.task_status_list('Async_SQL', 0, 1));
```

The output is similar to the following:

```
| [
  {
    "id": "17efb56b-52a1-11f0-bca4-020017205654",
    "data": {
      "Async Task completed at": "2025-06-26 15:20:34.000000"
    },
    "name": "DemoAsyncTask",
    "alias": "Thu-7",
    "status": "COMPLETED",
    "message": "Execution finished.",
    "progress": 100,
    "row_hash": "fa701b650afd8638049f8fd436a500f5",
    "task_data": {
      "mysqlMetadata": {
        "autoGc": true,
        "events": [
          "`demo_db`.`17ef9cf8-52a1-11f0-bca4-020017205654`"
        ]
      }
    },
    "task_type": "Async_SQL",
    "server_uuid": "5172a252-8a06-11ef-b3bb-02001727ecef",
    "progress_bar": "##########",
    "starting_time": "2025-06-26 15:20:29.000000",
    "scheduled_time": "2025-06-26 15:20:29.000000",
    "estimated_remaining_time": 0.00000,
    "estimated_completion_time": "2025-06-26 15:20:34.000000"
  }
] |
```

This example returns task status information one asynchronous task with task type `Async_SQL`.

See Also

Running tasks Asynchronously.

10.1.4.15 app_task_status_list

The `app_task_status_list` routine returns a paginated list of application task statuses.

This topic contains the following sections:

- Syntax
- Syntax Example
- See Also

Syntax

```
mysql> SELECT mysql_tasks.app_task_status_list(
  IN 'app_user_id' VARCHAR(255),
  IN 'task_type' VARCHAR(90),
  IN offset INT UNSIGNED,
  IN limit INT UNSIGNED);
```

Following are `task_status_list` parameters:

- `app_user_id` (VARCHAR(255)): specifies the unique ID of the application user who owns the task.
- `task_type` (VARCHAR(80)): specifies the task type for filtering the task statuses. If the value is `NULL`, status information for all tasks is included.
- `offset` (INT UNSIGNED): specifies the initial number of rows to skip. The offset value is used for pagination. If the value is `NULL`, the offset defaults to `0`, and the result is displayed starting from the first row.

- *limit* (INT UNSIGNED): specifies the maximum number of task records to include in the list of tasks. If the value is NULL, the limit defaults to 20, and up to 20 rows are displayed.

Syntax Example

```
mysql> SELECT JSON_PRETTY(mysql_tasks.app_task_status_list('app_user_01','demo task type', 0, 1));
```

The output is similar to the following:

```
| [
  {
    "id": "a516b352-574c-11f0-9604-020017205654",
    "data": {
      "Task result": "Finished..."
    },
    "name": "demo task name",
    "alias": "Wed-3",
    "status": "COMPLETED",
    "message": "Execution finished.",
    "progress": 100,
    "row_hash": "de3195fc0e2427d9b91cf03b920b96f2",
    "task_data": {
      "sleep": 5,
      "mysqlMetadata": {
        "autoGc": true,
        "events": [
          "`demo_db`.`a516a336-574c-11f0-9604-020017205654`",
          "`demo_db`.`a516a3f8-574c-11f0-9604-020017205654`"
        ]
      },
      "refresh_period": 1
    },
    "task_type": "demo task type",
    "server_uuid": "5172a252-8a06-11ef-b3bb-02001727ecef",
    "progress_bar": "##########",
    "starting_time": "2025-07-02 13:58:35.000000",
    "scheduled_time": "2025-07-02 13:58:35.000000",
    "estimated_remaining_time": 0.00000,
    "estimated_completion_time": "2025-07-02 13:58:40.000000"
  }
] |
```

This example returns task status information one asynchronous task associated with the specified application-managed user ID and task type.

See Also

Running tasks Asynchronously.

10.1.4.16 kill_task

The `kill_task` routine terminates an active or scheduled task using its task ID.

The application-level variant for this routine is `kill_app_task`.

This topic contains the following sections:

- Syntax
- Syntax Example
- See Also

Syntax

```
mysql> CALL mysql_tasks.kill_task(IN 'task_id' VARCHAR(36));
```

The `task_id` (VARCHAR(36)) parameter specifies the unique task ID (`UUID`) of the task to be terminated.

Syntax Example

```
mysql> CALL mysql_tasks.kill_task('17efb56b-52a1-11f0-bca4-020017205654');
```

This example terminates the task with the specified task ID.

See Also

Running tasks Asynchronously.

10.1.4.17 kill_app_task

The `kill_app_task` routine kills an active or scheduled task using its task ID. Users can only terminate tasks they own.

Syntax

```
mysql> SELECT mysql_tasks.kill_app_task(
  IN 'app_user_id' VARCHAR(255),
  IN 'task_id' VARCHAR(36);
```

Following are `kill_app_task` parameters:

- `app_user_id` (VARCHAR(255)): specifies the unique ID of the application user who owns the task.
- `task_id` (VARCHAR (36)): specifies the unique task ID (`UUID`) of the task to be terminated.

Syntax Example

```
mysql> SELECT mysql_tasks.kill_app_task('app_user_01','698968b7-3b96-11f0-a5dc-02001712c792');
```

This example terminates the task created by the specified application user and with the specified task ID.

See Also

Running tasks Asynchronously.

10.1.5 HEATWAVE_ADVISOR

```
mysql> CALL sys.HEATWAVE_ADVISOR (options);

options: NULL
       | JSON_OBJECT(keyvaluepair[, keyvaluepair] ...)
keyvaluepair: 'output', {'normal'|'silent'|'help'}
            | 'target_schema', JSON_ARRAY('db_name'[, 'db_name']...)
            | 'exclude_query', JSON_ARRAY('query_id'[, 'query_id']...)
            | 'query_session_id', JSON_ARRAY('query_session_id'[, 'query_session_id']...)
            | 'query_insights', {true|false}
            | 'auto_enc', JSON_OBJECT(auto_enc_option)
            | 'auto_dp', JSON_OBJECT(auto_dp_option[, auto_dp_option])
            | 'auto_unload', JSON_OBJECT(auto_unload_option[, auto_unload_option]...)

auto_enc_option: 'mode', {'off'|'recommend'} [, 'fixed_enc', JSON_OBJECT(col_enc[, col_enc]...)]
col_enc: 'db_name.tbl_name.col_name', {'varlen'|'dictionary'}

auto_dp_option: 'benefit_threshold',N
              | 'max_combinations',N

auto_unload_option: 'mode', {'off'|'recommend'}
                  | 'exclude_list', JSON_ARRAY('db_obj'[, 'db_obj']...)
```

```
                  | 'last_queried_hours', N
                  | 'memory_gain_ascending', {true|false}
                  | 'limit_tables', N
db_obj: db_name
       db_name.tbl_name
```

Advisor *`options`* are specified as key-value pairs in `JSON` format. Options include:

- `output`: Defines how Advisor produces output. Permitted values are:
  - `normal`: The default. Produces summarized output and sends it to `stdout` and to the `heatwave_autopilot_report` table. See Section 5.8.8, “Autopilot Report Table”.
  - `silent`: Sends output to the `heatwave_autopilot_report` table only. See Section 5.8.8, “Autopilot Report Table”. The `silent` output type is useful if human-readable output is not required; when the output is consumed by a script, for example.
  - `help`: Displays Advisor command-line help.
- `target_schema`: Defines one or more schemas for Advisor to analyze. The list is specified as a `JSON` array. If a target schema is not specified, Advisor analyzes all schemas in the MySQL HeatWave Cluster. When a target schema is specified, Advisor generates recommendations for tables belonging to the target schema. For the most accurate recommendations, specify one schema at a time. Only run Advisor on multiple schemas if the queries access tables in multiple schemas.
- `exclude_query`: Defines the IDs of queries to exclude when Advisor analyzes query statistics. To identify query IDs, query the `performance_schema.rpd_query_stats` table. For a query example, see Auto Data Placement Examples.
- `query_session_id`: Defines session IDs for filtering queries by session ID. To identify session IDs, query the `performance_schema.rpd_query_stats` table. For a query example, see Auto Query Time Estimation Examples.
- `query_insights`: Provides runtime information for successfully executed queries and runtime estimates for `EXPLAIN` queries, queries cancelled using `Ctrl+C`, and queries that fail due to an out-of-memory error. See: Section 5.8.6, “Auto Query Time Estimation”. The default setting is `false`.

`auto_enc`: Defines settings for Auto Encoding, see Section 5.8.4, “Auto Encoding”. Options include:

- `mode`: Defines the operational mode. Permitted values are:
  - `off`: The default. Disables Auto Encoding.
  - `recommend`: Enables Auto Encoding.
- `fixed_enc`: Defines an encoding type for specified columns. Use this option if you know the encoding you want for a specific column and you are not interested in an encoding recommendation for that column. Only applicable in `recommend` mode. Columns with a fixed encoding type are excluded from encoding recommendations. The `fixed_enc` key is a fully qualified column name without backticks in the following format: *`schema_name.tbl_name.col_name`*. The value is the encoding type; either `varlen` or `dictionary`. Multiple key-value pairs can be specified in a comma-separated list.

`auto_dp`: Defines settings for Data Placement, which recommends data placement keys. See: Section 5.8.5, “Auto Data Placement”. Options include:

- `benefit_threshold`: The minimum query performance improvement expressed as a percentage value. Advisor only suggests data placement keys estimated to meet or exceed the `benefit_threshold`. The default value is 0.01 (1%). Query performance is a combined measure of all analyzed queries.

- `max_combinations`: The maximum number of data placement key combinations Advisor considers before making recommendations. The default is 10000. The supported range is 1 to 100000. Specifying fewer combinations generates recommendations more quickly but recommendations may not be optimal.

`auto_unload`: Defines settings for Unload Advisor, which recommends tables to unload. See: Section 5.8.7, “Unload Advisor”. Options include:

- `mode`: Defines the operational mode. Permitted values are:
  - `off`: The default. Disables Unload Advisor.
  - `recommend`: Enables Unload Advisor.
- `exclude_list`: Defines a list of schemas and tables to exclude from Unload Advisor. Names must be fully qualified without backticks.
- `last_queried_hours`: Recommend unloading tables that were not queried in the past `last_queried_hours` hours. Minimum: `1`, maximum `744`, default: `24`.
- `memory_gain_ascending`: Whether to rank the unload table suggestions in ascending or descending order based on the table size. Default: `false`.
- `limit_tables`: A limit to the number of unload table suggestions, based on the order imposed by `memory_gain_ascending`. The default is `10`.

Advisor command-line help

To access the Advisor command-line help:

```
mysql> CALL sys.HEATWAVE_ADVISOR(JSON_OBJECT('output','help'));
```

10.2 MySQL HeatWave AutoML Routines

MySQL HeatWave AutoML routines reside in the MySQL `sys` schema.

10.2.1 ML_TRAIN

Run the `ML_TRAIN` routine on a training dataset to produce a trained machine learning model.

MySQL 9.0.0 introduces changes to how MySQL HeatWave AutoML stores models. See The Model Object Catalog Table. `ML_TRAIN` upgrades older models.

Before training models, make sure to review the following:

- Additional MySQL HeatWave AutoML Requirements
- MySQL HeatWave AutoML Privileges
- Supported Data Types for MySQL HeatWave AutoML
- Train a Model
- Machine Learning Use Cases

This topic has the following sections. Refer to the appropriate sections depending on the type of machine learning model you would like to train.

- ML_TRAIN Syntax
- Required ML_TRAIN Parameters

- Common ML_TRAIN Options
- Parameters to Train a Classification Model
- Syntax Examples for Classification Training
- Parameters to Train a Regression Model
- Syntax Examples for Regression Training
- Parameters to Train a Forecasting Model
- Syntax Examples for Forecast Training
- Parameters to Train an Anomaly Detection Model
- Syntax Examples for Anomaly Detection Training
- Parameters to Train a Recommendation Model
- Syntax Examples for Recommendation Training
- Parameters to Train a Model with Topic Modeling
- Syntax Examples for Topic Modeling Training
- ML_TRAIN and ML_EXPLAIN
- Additional Syntax Examples
- See Also

ML_TRAIN Syntax

```
mysql> CALL sys.ML_TRAIN ('table_name', 'target_column_name', options, model_handle);

options: {JSON_OBJECT(keyvalue[, keyvalue]...)|NULL}
keyvalue:
{
  'task', {'classification'|'regression'|'forecasting'|'anomaly_detection'|'log_anomaly_detection'|'recomme
  |'datetime_index', 'column'
  |'endogenous_variables', JSON_ARRAY('column'[,'column'] ...)
  |'exogenous_variables', JSON_ARRAY('column'[,'column'] ...)
  |'model_list', JSON_ARRAY('model'[,'model'] ...)
  |'exclude_model_list', JSON_ARRAY('model'[,'model'] ...)
  |'optimization_metric', 'metric'
  |'include_column_list', JSON_ARRAY('column'[,'column'] ...)
  |'exclude_column_list', JSON_ARRAY('column'[,'column'] ...)
  |'contamination',  'contamination factor'
  |'supervised_submodel_options', {'n_neighbors', 'N', 'min_labels', 'N'}
  |'experimental', {'semisupervised'}
  |'ensemble_score', 'ensemble metric'
  |'users', 'users_column'
  |'items', 'items_column'
  |'notes', 'notes_text'
  |'feedback', {'explicit'|'implicit'}
  |'feedback_threshold', 'threshold'
  |'item_metadata', JSON_OBJECT('table_name'[,'database_name.table_name'] ...)
  |'user_metadata', JSON_OBJECT('table_name'[,'database_name.table_name'] ...)
  |'document_column', 'column_name'
  |'logad_options', JSON_OBJECT(("key","value"[,"key","value"] ...)
}

logad_options: JSON_OBJECT(keyvalue[, keyvalue]...)
keyvalue:
{
  'additional_masking_regex', JSON_ARRAY('regular_expression'[,'regular_expression', ...])
  |'window_size', 'N'
  |'window_stride', 'N'
```

```
  |'log_source_column', 'column'
  |'embedding_model', 'model'
  |'keyword_model', 'model'
}
```

Required ML_TRAIN Parameters

Set the following parameters to train all machine learning models.

- `table_name`: The name of the table that contains the labeled training dataset. The table name must be valid and fully qualified, so it must include the database name, *`database_name.table_name`*. The table cannot exceed 10 GB, 100 million rows, or 1017 columns.
- `target_column_name`: The name of the target column containing ground truth values.

  MySQL HeatWave AutoML does not support a text target column.

  If training an unsupervised Anomaly detection model (unlabeled data), set `target_column_name` to `NULL`.

  Forecasting does not require `target_column_name`, and it can be set to `NULL`.
- `model_handle`: A user-defined session variable that stores the machine learning model handle for the duration of the connection. User variables are written as *`@var_name`*. Any valid name for a user-defined variable is permitted. For example, `@my_model`.

  If you set a value to the `model_handle` variable before calling `ML_TRAIN`, that model handle is used for the model. A model handle must be unique in the model catalog. We recommend this method.

  If you don't set a value to the `model_handle` variable, MySQL HeatWave AutoML generates one. When `ML_TRAIN` finishes executing, retrieve the generated model handle by querying the session variable. See Model Handles to learn more.

Common ML_TRAIN Options

The following optional parameters apply to more than one type of machine learning task. They are specified as key-value pairs in `JSON` format. If an option is not specified, the default setting is used. If no options are specified, you can specify `NULL` in place of the `JSON` argument.

- `task`: Specifies the machine learning task.
  - `classification`: The default value if a task is not set. Use this task type to assign items to defined categories.
  - `regression`: Use this task type if the target column is a continuous numerical value. This task generates predictions based on the relationship between a dependent variable and one or more independent variables.
  - `forecasting`: Use this task type if you have a date-time column that requires a timeseries forecast. To use this task, you must set a target column, the date-time column (`datetime_index`), and endogenous variables (`endogenous_variables`).
  - `anomaly_detection`: Use this task type to detect unusual patterns in data.
  - `log_anomaly_detection`: Use this task to detect unusual patterns in log data (As of MySQL 9.2.2).
  - `recommendation`: Use this task type for generate recommendations for users and items.
  - `topic_modeling`: Use this task to cluster word groups and similar expressions that best characterize the documents (As of MySQL 9.0.1-u1).

- `model_list`: The type of model to be trained. If more than one model is specified, the best model type is selected from the list. See Model Types.

  This option cannot be used together with the `exclude_model_list` option.

- `exclude_model_list`: Model types that should not be trained. Specified model types are excluded from consideration during model selection. See Model Types.

  This option cannot be specified together with the `model_list` option.

- `optimization_metric`: The scoring metric to optimize for when training a machine learning model. The metric must be compatible with the `task` type and the target data. See Section 10.2.16, "Optimization and Scoring Metrics".

  This is not supported for `anomaly_detection` tasks. Instead, metrics for anomaly detection can only be used with the `ML_SCORE` routine.

- `include_column_list`: `ML_TRAIN` must include this list of columns.

  For `classification`, `regression`, `anomaly_detection` and `recommendation` tasks, `include_column_list` ensures that `ML_TRAIN` will not drop these columns.

  For `forecasting` tasks, `include_column_list` can only include `exogenous_variables`. If `include_column_list` is included in the `ML_TRAIN` options for a `forecasting` task with at least one `exogenous_variables`, this forces `ML_TRAIN` to only consider those models that support `exogenous_variables`.

  All columns in `include_column_list` must be included in the training table.

- `exclude_column_list`: Feature columns of the training dataset to exclude from consideration when training a model. Columns that are excluded using `exclude_column_list` do not also need to be excluded from the dataset used for predictions.

  The `exclude_column_list` cannot contain any columns provided in `endogenous_variables`, `exogenous_variables`, and `include_column_list`.

- `notes`: Add notes to the `model_metadata` for your own reference.

Refer to the following model-specific parameters to train different types of machine learning models.

Parameters to Train a Classification Model

To train a classification model, set the `task` to `classification`.

If the `task` is set to `NULL`, or if all training options is set to `NULL`, a classification model is trained by default.

Syntax Examples for Classification Training

- The following example sets the model handle before training, which is good practice. See Define Model Handle. The `task` is set to `classification`.

```
mysql> SET @census_model = 'census_manual';
mysql> CALL sys.ML_TRAIN('census_data.census_train', 'revenue', JSON_OBJECT('task', 'classification'), @c
```

- The following example sets all options to `NULL`, so `ML_TRAIN` runs the `classification` task option by default.

```
mysql> CALL sys.ML_TRAIN('census_data.census_train', 'revenue', NULL, @census_model);
```

Parameters to Train a Regression Model

To train a regression model, set the `task` to `regression`.

Syntax Examples for Regression Training

- The following example specifies the `regression` task type.

```
mysql> CALL sys.ML_TRAIN('nyc_taxi.nyc_taxi_train', 'tip_amount', JSON_OBJECT('task', 'regression'),
```

Parameters to Train a Forecasting Model

See the following to learn more about forecasting models:

- Forecasting Task Types
- Prediction Intervals
- Train a Forecasting Model

To train a forecasting model, set the `task` to `forecasting` and set the following required parameters.

- `datetime_index`: The column name for a datetime column that acts as an index for the forecast variable. The column can be one of the supported datetime column types, `DATETIME`, `TIMESTAMP`, `DATE`, `TIME`, and `YEAR`, or an auto-incrementing index.

  The forecast models `SARIMAXForecaster`, `VARMAXForecaster`, and `DynFactorForecaster` cannot back test, that is forecast into training data, when using `exogenous_variables`. Therefore, the predict table must not overlap the `datetime_index` with the training table. The start date in the predict table must be a date immediately following the last date in the training table when `exogenous_variables` are used. For example, the predict table has to start with year 2024 if the training table with `YEAR` data type `datetime_index` ends with year 2023.

  The `datetime_index` for the predict table must not have missing dates after the last date in the training table. For example, the predict table has to start with year 2024 if the training table with `YEAR` data type `datetime_index` ends with year 2023. The predict table cannot start with year, for example, 2025 or 2030, because that would miss out 1 and 6 years, respectively.

  When *`options`* do not include `exogenous_variables` , the predict table can overlap the `datetime_index` with the training table. This supports back testing.

  The valid range of years for `datetime_index` dates must be between 1678 and 2261. It will cause an error if any part of the training table or predict table has dates outside this range. The last date in the training table plus the predict table length must still be inside the valid year range. For example, if the `datetime_index` in the training table has `YEAR` data type, and the last date is year 2023, the predict table length must be less than 238 rows: 2261 minus 2023 equals 238 rows.

- `endogenous_variables`: The column or columns to be forecast.

  Univariate forecasting models support a single numeric column, specified as a `JSON_ARRAY`. This column must also be specified as the `target_column_name`, because that field is required, but it is not used in that location.

  Multivariate forecasting models support multiple numeric columns, specified as a `JSON_ARRAY`. One of these columns must also be specified as the `target_column_name`.

  `endogenous_variables` cannot be text.

Set the following forecasting options as required to train forecasting models.

- `exogenous_variables`: For forecasting tasks, the column or columns of independent, non-forecast, predictive variables, specified as a `JSON_ARRAY`. These optional variables are not forecast, but help to predict the future values of the forecast variables. These variables affect a model without being affected by it. For example, for sales forecasting these variables might be advertising expenditure, occurrence of promotional events, weather, or holidays.

`ML_TRAIN` will consider all supported models during the algorithm selection stage if *`options`* includes `exogenous_variables`, including models that do not support `exogenous_variables`.

For example, if *`options`* includes univariate `endogenous_variables` with `exogenous_variables`, then `ML_TRAIN` will consider `NaiveForecaster`, `ThetaForecaster`, `ExpSmoothForecaster`, `ETSForecaster`, `STLwESForecaster`, `STLwARIMAForecaster`, and `SARIMAXForecaster`. `ML_TRAIN` will ignore `exogenous_variables` if the model does not support them.

Similarly, if *`options`* includes multivariate `endogenous_variables` with `exogenous_variables`, then `ML_TRAIN` will consider `VARMAXForecaster` and `DynFactorForecaster`.

If *`options`* also includes `include_column_list`, this will force `ML_TRAIN` to only consider those models that support `exogenous_variables`.

- `include_column_list`: Can only include `exogenous_variables`. If `include_column_list` contains at least one `exogenous_variables`, this will force `ML_TRAIN` to only consider those models that support `exogenous_variables`.

Syntax Examples for Forecast Training

- The following example specifies the `forecasting` task type, and the additional required parameters, `datetime_index` and `endogenous_variables`.

```
mysql> CALL sys.ML_TRAIN('air_data.air_train', 'passengers',
                         JSON_OBJECT('task', 'forecasting',
                                     'datetime_index', 'Period',
                                     'endogenous_variables', JSON_ARRAY('passengers')), @model);
```

- The following example specifies the `OrbitForecaster` forecasting model with exogenous variables.

```
mysql> CALL sys.ML_TRAIN('mlcorpus.opsd_germany_daily_train', NULL,
                         JSON_OBJECT('task', 'forecasting',
                                     'datetime_index', 'ddate',
                                     'endogenous_variables', JSON_ARRAY('consumption'),
                                     'exogenous_variables', JSON_ARRAY('wind', 'solar', 'wind_solar'),
                                     'model_list', JSON_ARRAY('OrbitForecaster')), @model);
```

- The following example specifies the `OrbitForecaster` forecasting model without exogenous variables.

```
mysql> CALL sys.ML_TRAIN('air_data.air_train', 'passengers',
                         JSON_OBJECT('task', 'forecasting',
                                     'datetime_index', 'Period',
                                     'endogenous_variables', JSON_ARRAY('passengers'),
                                     'model_list', JSON_ARRAY('OrbitForecaster')), @model);
```

Parameters to Train an Anomaly Detection Model

See the following to learn more about anomaly detection models:

- Anomaly Detection Model Types
- Anomaly Detection Learning Types
- Anomaly Detection for Logs

To train an anomaly detection model, set the appropriate required parameters depending on the type of anomaly detection model to train.

- Set the `task` parameter to `anomaly_detection` for running anomaly detection on table data, or `log_anomaly_detection` for running anomaly detection on log data (MySQL 9.2.2 and later).

- If running an unsupervised model, the `target_column_name` parameter must be set to `NULL`.
- If running a semi-supervised model:
  - The `target_column_name` parameter must specify a column whose only allowed values are 0 (normal), 1 (anomalous), and NULL (unlabeled). All rows will be used to train the unsupervised component, while the rows with a value different than NULL will be used to train the supervised component.
  - The `experimental` option must be set to `semisupervised`.
- If running anomaly detection on log data (MySQL 9.2.2 and later), the input table can only have the following columns:
  - The column containing the logs.
  - If including logs from different sources, a column containing the source of each log. Identify this column with the `log_source_column` option.
  - If including labeled data, a column identifying the labeled log lines. See Semi-supervised Anomaly Detection to learn more.
  - As of MySQL 9.4.1, at least one column must act as the primary key to establish the temporal order of logs. If the primary key column (or columns) is not one of the previous required columns (log data, source of log, or label), then you must use the `exclude_column_list` option when running `ML_TRAIN` to exclude all primary key columns that don't include required data. See Syntax Examples for Anomaly Detection Training to review relevant examples.
  - If the input table has additional columns to the ones permitted, you must use the `exclude_column_list` option to exclude irrelevant columns.

Set the following options as needed for anomaly detection models:

- `contamination`: Represents an estimate of the percentage of outliers in the training table.
  - The contamination factor is calculated as: estimated number of rows with anomalies/total number of rows in the training table.
  - The contamination value must be greater than 0 and less than 0.5. The default value is 0.01.
- `model_list`: You can select the Principal Component Analysis (PCA), Generalized Local Outlier Factor (GLOF), or Generalized kth Nearest Neighbors (GkNN) model. If no option is specified, the default model is GkNN. Selecting more than one model or an unsupported model produces an error.

To train a semi-supervised anomaly detection model (MySQL 9.0.1-u1 and later), set the following options:

- `supervised_submodel_options`: Allows you to set optional override parameters for the supervised model component. The only model supported is `DistanceWeightedKNNClassifier`. The following parameters are supported:
  - `n_neighbors`: Sets the desired k value that checks the k closest neighbors for each unclassified point. The default value is 5 and the value must be an integer greater than 0.
  - `min_labels`: Sets the minimum number of labeled data points required to train the supervised component. If fewer labeled data points are provided during training of the model, `ML_TRAIN` fails. The default value is 20 and the value must be an integer greater than 0.
- `ensemble_score`: This option specifies the metric to use to score the ensemble of unsupervised and supervised components. It identifies the optimal weight between the two components based on the metric. The supported metrics are `accuracy`, `precision`, `recall`, and `f1`. The default metric is `f1`.

To train a model for anomaly detection on log data (MySQL 9.2.2 and later), set the following options:

- `logad_options`: A `JSON_OBJECT` that allows you to configure the following options.
  - `additional_masking_regex`: Allows you to mask log data in a `JSON_ARRAY`. By default, the following parameters are automatically masked during training.
    - IP
    - DATETIME
    - TIME
    - HEX
    - IPPORT
    - OCID
  - `window_size`: Specifies the maximum number of log lines to be grouped for anomaly detection. The default value is 10.
  - `window_stride`: Specifies the stride value to use for segmenting log lines. For example, there is log A, B, C, D, and E. The `window_size` is 3, and the `window_stride` is 2. The first row has log A, B, and C. The second row has log C, D, and E. If this value is equal to `window_size`, there is no overlapping of log segments. The default value is 3.
  - `log_source_column`: Specifies the column name that contains the source identifier of the respective log lines. Log lines are grouped according to their respective source (for example, logs from multiple MySQL databases that are in the same table). By default, all log lines are assumed to be from the same source.
  - `embedding_model`: Available as of MySQL 9.5.0. The embedding model used to extract semantic features from log data. To review supported embedding models in MySQL HeatWave, run the following query: `SELECT sys.ML_LIST_LLMS();` and see models that have `capabilities` with `TEXT_EMBEDDINGS`. The default value is `multilingual-e5-small`. Using an embedding model causes higher memory usage. If you set this to `NULL`, then you cannot also set `keyword_model` to `NULL`.
  - `keyword_model`: Available as of MySQL 9.5.0. The keyword feature extractor used to extract keyword features from log data. The available options are `tf-idf` and `NULL`. The default value is `tf-idf`. If you set this to `NULL`, then you cannot also set `embedding_model` to `NULL`.

Anomaly detection models don't support the following options during training:

- `exclude_model_list`
- `optimization_metric`

Syntax Examples for Anomaly Detection Training

- The following example specifies the `anomaly_detection` task type and excludes the `target` column for training. After training, query the model type to confirm the default GKNN model is selected.

```
mysql> CALL sys.ML_TRAIN('volcano_data.volcano_data_train', NULL,
                         JSON_OBJECT('task', 'anomaly_detection',
                                     'exclude_column_list', JSON_ARRAY('target')), @anomaly_model);
Query OK, 0 rows affected (10.1872 sec)
mysql> SELECT model_handle, model_type FROM ML_SCHEMA_admin.MODEL_CATALOG WHERE model_handle=@model;
+---------------+------------+
| model_handle  | model_type |
+---------------+------------+
```

```
| anomaly_model | GKNN       |
+---------------+------------+
1 row in set (0.0428 sec)
```

- The following example specifies the `anomaly_detection` task with a `contamination` option. After training, query the model catalog metadata to check the value of the `contamination` option.

```
mysql> CALL sys.ML_TRAIN('volcano_data.volcano_data_train', NULL,
                         JSON_OBJECT('task', 'anomaly_detection',
                                     'contamination', 0.013,
                                     'exclude_column_list', JSON_ARRAY('target')), @model2);
Query OK, 0 rows affected (11.6307 sec)

mysql> SELECT JSON_EXTRACT(model_metadata, '$.contamination')
          FROM ML_SCHEMA_user1.MODEL_CATALOG
          WHERE model_handle = @model2;
+-----------------------------------------------+
| JSON_EXTRACT(model_metadata, '$.contamination') |
+-----------------------------------------------+
| 0.0130000002682209                            |
+-----------------------------------------------+
1 row in set (0.0717 sec)
```

- The following example enables semi-supervised learning using all defaults. The *`target_column_name`* is set to `target`. The `experimental` option is set to `semisupervised`.

```
mysql> CALL sys.ML_TRAIN('mlcorpus.anomaly_train_with_partial_target', "target",
                         CAST('{"task": "anomaly_detection", "experimental": {"semisupervised": {}}}
                         @semisupervised_model);
```

- The following example enables semi-supervised learning with additional options.

```
mysql> CALL sys.ML_TRAIN('mlcorpus.anomaly_train_with_partial_target', "target",
                         CAST('{"task": "anomaly_detection",
                                "experimental": {"semisupervised": {"supervised_submodel_options": {
                                "ensemble_score": "recall"}}}' as JSON), @semisupervised_model_optio
```

  Where:

  - The `supervised_submodel_options` parameter `min_labels` is set to 10.
  - The `supervised_submodel_options` parameter `n_neighbors` is set to 3.
  - The `ensemble_score` option is set to the `recall` metric.

- The following example selects the PCA (Principal Component Analysis) anomaly detection model. After training, query the model handle and model type to confirm the selected model.

```
mysql> CALL sys.ML_TRAIN('volcano_data.volcano_data_train', NULL,
                         JSON_OBJECT('task', 'anomaly_detection',
                                     'exclude_column_list', JSON_ARRAY('target'),
                                     'model_list', JSON_ARRAY('PCA')), @model);
Query OK, 0 rows affected (4.8730 sec)
mysql> SELECT model_handle, model_type FROM ML_SCHEMA_admin.MODEL_CATALOG WHERE model_handle=@model;
+--------------+------------+
| model_handle | model_type |
+--------------+------------+
| anomaly_pca  | PCA        |
+--------------+------------+
1 row in set (0.0416 sec)
```

- The following example runs the `log_anomaly_detection` task with available default values.

```
mysql> CALL sys.ML_TRAIN('mlcorpus.`log_anomaly_just_patterns`', NULL, JSON_OBJECT('task', 'log_anoma
```

- The following example runs the `log_anomaly_detection` task with the PCA anomaly detection model.

```
mysql> CALL sys.ML_TRAIN('mlcorpus.`log_anomaly_just_patterns`', NULL,
```

```
                          JSON_OBJECT('task', 'log_anomaly_detection',
                                      'model_list', JSON_ARRAY('PCA')), @logad_model);
```

- An `ML_TRAIN` example that excludes two primary key columns: `primary_key_column1` and `primary_key_column2`. These columns must be excluded because they do not have one of the required items of data for training: the log data, the source of the log, or the label.

```
mysql> CALL sys.ML_TRAIN('mlcorpus.log_anomaly_two_primary', NULL,
                          JSON_OBJECT('task', 'log_anomaly_detection',
                                      'logad_options', JSON_OBJECT('window_size', 2, 'window_stride', 1),
                                      'exclude_column_list', JSON_ARRAY('primary_key_column1', 'primary_k
                                      @log_anomaly_us );
```

- The following example runs the `log_anomaly_detection` task and masks log data with the `additional_masking_regex` option. In addition to the default parameters that are automatically masked, email addresses from Yahoo, Hotmail, and Gmail are also masked. The `log_source_column` option is also included, which specifies the column that identifies the respective source of the log line.

```
mysql> CALL sys.ML_TRAIN('mlcorpus.`log_anomaly_sourced`', NULL,
                          JSON_OBJECT('task', 'log_anomaly_detection',
                                      'logad_options',
                                      JSON_OBJECT('additional_masking_regex',
                                      JSON_ARRAY('(\W|^)[\w.\-]{0,25}@(yahoo|hotmail|gmail)\.com(\W|$)'),
                                      'log_source_column', 'source')), @log_anomaly_us);
```

- The following example sets semi-supervised learning for training log data for anomaly detection. The window size is also set to a value of 4, and the window stride is set to 1.

```
mysql> CALL sys.ML_TRAIN('mlcorpus.`log_anomaly_semi`', "label",
                          JSON_OBJECT('task', 'log_anomaly_detection',
                                      'logad_options',
                                      JSON_OBJECT('window_size', 4, 'window_stride', 1),
                                      "experimental",
                                      JSON_OBJECT("semisupervised",
                                      JSON_OBJECT("supervised_submodel_options",
                                      JSON_OBJECT("min_labels", 10)))), @log_anomaly_us);
```

- The following example sets unsupervised learning for training log data for anomaly detection. A query reviews supported embedding models. The `all_minilm_l12_v2` embedding model and `tf-idf` keyword model are selected for training.

```
mysql> SELECT sys.ML_LIST_LLMS();
+-----------------------------------------------------------------------------------------------------------------
| sys.ML_LIST_LLMS()
+-----------------------------------------------------------------------------------------------------------------
| [{"model_id": "llama3.1-8b-instruct-v1", "provider": "HeatWave", "capabilities": ["GENERATION"], "defau
| {"model_id": "llama3.2-1b-instruct-v1", "provider": "HeatWave", "capabilities": ["GENERATION"], "defaul
| {"model_id": "llama3.2-3b-instruct-v1", "provider": "HeatWave", "capabilities": ["GENERATION"], "defaul
| {"model_id": "mistral-7b-instruct-v3", "provider": "HeatWave", "capabilities": ["GENERATION"], "default
| {"model_id": "all_minilm_l12_v2", "provider": "HeatWave", "capabilities": ["TEXT_EMBEDDINGS"], "default
| {"model_id": "multilingual-e5-small", "provider": "HeatWave", "capabilities": ["TEXT_EMBEDDINGS"], "def
+-----------------------------------------------------------------------------------------------------------------
mysql> SET @model='log_embedding_model';
Query OK, 0 rows affected (0.0490 sec)
mysql> CALL sys.ML_TRAIN('anomaly_log_embedding.training_data', NULL,
                          JSON_OBJECT('task', 'log_anomaly_detection',
                                      'exclude_column_list',
                                      JSON_ARRAY('log_id', 'timestamp', 'target'),
                                      'logad_options',
                                      JSON_OBJECT('embedding_model', 'all_minilm_l12_v2', 'keyword_model'
Query OK, 0 rows affected (27.0830 sec)
```

Parameters to Train a Recommendation Model

See Recommendation Task Types to learn more about recommendation models.

To train a recommendation model, set the `task` to `recommendation` and set the following required parameters.

- `users`: Specifies the column name corresponding to the user ids. Values in this column must be in a `STRING` data type, otherwise an error will be generated during training.

  This must be a valid column name, and it must be different from the `items` column name.

- `items`: Specifies the column name corresponding to the item ids. Values in this column must be in a `STRING` data type, otherwise an error will be generated during training.

  This must be a valid column name, and it must be different from the `users` column name.

To train a recommendation model with explicit feedback, set `feedback` to `explicit`. If `feedback` is not set, the default value is `explicit`.

To train a recommendation model with implicit feedback, set `feedback` to `implicit` and set the following option as needed:

- `feedback_threshold`: The feedback threshold for a recommendation model that uses implicit feedback. It represents the threshold required to be considered positive feedback. For example, if numerical data records the number of times users interact with an item, you might set a threshold with a value of 3. This means users would need to interact with an item more than three times to be considered positive feedback.

To train a content-based recommendation model, set `feedback` to `implicit` and set the following required parameters:

- `item_metadata`: As of MySQL 9.5.0, it defines the table that has item descriptions. It is a JSON object that has the `table_name` option as a key, which specifies the table that has item descriptions. One column must be the same as the `item_id` in the input table.
- `user_metadata`: As of MySQL 9.5.0, it defines the table that has user descriptions. It is a JSON object that has the `table_name` option as a key, which specifies the table that has user descriptions. One column must be the same as the `user_id` in the input table.
  - `table_name`: To be used with the `item_metadata` and `user_metadata` options. It specifies the table name that has item or user descriptions. It must be a string in a fully qualified format (database_name.table_name) that specifies the table name.

Syntax Examples for Recommendation Training

- The following example defines a model handle, trains a recommendation model with no specified model type, and then queries the automatically selected `model_type` for the trained model from the model catalog. As of MySQL 9.5.0, the default model is `TwoTower`.

```
mysql> SET @rec_model = 'rec_model';
mysql> CALL sys.ML_TRAIN('movielens_data.movielens_train', 'rating',
                         JSON_OBJECT('task', 'recommendation',
                                     'users', 'user_id',
                                     'items', 'item_id'), @rec_model);
Query OK, 0 rows affected (14.4091 sec)

mysql> SELECT model_handle, model_type FROM ML_SCHEMA_admin.MODEL_CATALOG WHERE model_handle='rec_mod
+--------------+------------+
| model_handle | model_type |
+--------------+------------+
| rec_model    | TwoTower   |
+--------------+------------+
1 row in set (0.0395 sec)
```

- The following example specifies the `SVD` recommendation model type, and then queries the `model_type` for the trained model from the model catalog to confirm the selected model.

```
mysql> CALL sys.ML_TRAIN('movielens_data.movielens_train', 'rating',
                         JSON_OBJECT('task', 'recommendation',
                                     'users', 'user_id',
                                     'items', 'item_id',
```

```
                                    'model_list', JSON_ARRAY('SVD')), @rec_model);
Query OK, 0 rows affected (9.4139 sec)

mysql> SELECT model_type FROM ML_SCHEMA_admin.MODEL_CATALOG WHERE model_handle=@rec_model;
+------------+
| model_type |
+------------+
| SVD        |
+------------+
1 row in set (0.0485 sec)
```

- The following example specifies three models for the `model_list` option. From those three recommendation models, the `SVDpp` model is automatically selected for training.

```
mysql> CALL sys.ML_TRAIN('movielens_data.movielens_train', 'rating',
                         JSON_OBJECT('task', 'recommendation',
                                     'users', 'user_id',
                                     'items', 'item_id',
                                     'model_list', JSON_ARRAY('SVD', 'SVDpp', 'NMF')), @rec_model);

Query OK, 0 rows affected (13.8714 sec)

mysql> SELECT model_type FROM ML_SCHEMA_admin.MODEL_CATALOG WHERE model_handle=@rec_model;
+------------+
| model_type |
+------------+
| SVDpp      |
+------------+
1 row in set (0.0403 sec)
```

- The following example specifies the `recommendation` task with implicit feedback.

```
mysql> CALL sys.ML_TRAIN('mlcorpus.training_table', 'rating',
                         JSON_OBJECT('task', 'recommendation',
                                     'users', 'user_id',
                                     'items', 'item_id',
                                     'feedback', 'implicit'), @model);
Query OK, 0 rows affected (2 min 13.6415 sec)
```

- The following example trains a content-based recommendation model by specifying a table with item descriptions (`mlcorpus_recsys.\`citeulike_items_sample`). The optimization metric `hit_ratio_at_k` is used. The model must use implicit feedback.

```
mysql> CALL sys.ML_TRAIN('amazon_data.`amazon_train`', 'rating',
                         JSON_OBJECT('task', 'recommendation',
                                     'model_list', JSON_ARRAY('CTR'),
                                     'users', 'user_id',
                                     'items', 'item_id',
                                     'feedback', 'implicit',
                                     'optimization_metric', 'hit_ratio_at_k',
                                     'item_metadata', JSON_OBJECT('table_name', 'amazon_data.`amazon_ite
```

Parameters to Train a Model with Topic Modeling

To train a machine learning model with topic modeling, set the `task` to `topic_modeling` and set the following required parameter:

- `document_column`: Specify the column name that contains the text to train.

The following parameters are not supported for training machine learning models with topic modeling:

- `model_list`
- `optimization_metric`
- `exclude_model_list`
- `exclude_column_list`
- `include_column_list`

Syntax Examples for Topic Modeling Training

The following example runs the `topic_modeling` task with the required defined parameters.

```
mysql> CALL sys.ML_TRAIN('topic_modeling_data.text_types_train', NULL,
                         JSON_OBJECT('task', 'topic_modeling',
                                     'document_column', 'D0'), @topic_model);
```

ML_TRAIN and ML_EXPLAIN

The `ML_TRAIN` routine also runs the `ML_EXPLAIN` routine with the default Permutation Importance model for prediction explainers and model explainers. See Generate Model Explanations. To train other prediction explainers and model explainers use the `ML_EXPLAIN` routine with the preferred explainer after `ML_TRAIN`.

`ML_EXPLAIN` does not support the `anomaly_detection` and `recommendation` tasks, and `ML_TRAIN` does not run `ML_EXPLAIN`.

Additional Syntax Examples

- The `model_list` option permits specifying the type of model to be trained. If more than one type of model specified, the best model type is selected from the list. For a list of supported model types, see Model Types. This option cannot be used together with the `exclude_model_list` option.

  The following example trains either an `XGBClassifier` or `LGBMClassifier` model.

  ```
  mysql> CALL sys.ML_TRAIN('ml_data.iris_train', 'class',
                           JSON_OBJECT('task','classification',
                                       'model_list', JSON_ARRAY('XGBClassifier', 'LGBMClassifier')), @
  ```

- The `exclude_model_list` option specifies types of models that should not be trained. Specified model types are excluded from consideration. For a list of model types you can specify, see Model Types. This option cannot be used together with the `model_list` option.

  The following example excludes the `LogisticRegression` and `GaussianNB` models.

  ```
  mysql> CALL sys.ML_TRAIN('ml_data.iris_train', 'class',
                           JSON_OBJECT('task','classification',
                                       'exclude_model_list', JSON_ARRAY('LogisticRegression', 'Gaussia
  ```

- The `optimization_metric` option specifies a scoring metric to optimize for. See: Optimization and Scoring Metrics.

  The following example optimizes for the `neg_log_loss` metric.

  ```
  mysql> CALL sys.ML_TRAIN('heatwaveml_bench.census_train', 'revenue',
                           JSON_OBJECT('task','classification',
                                       'optimization_metric', 'neg_log_loss'), @census_model);
  ```

- The `exclude_column_list` option specifies feature columns to exclude from consideration when training a model.

  The following example excludes the `'age'` column from consideration when training a model for the `census` dataset.

  ```
  mysql> CALL sys.ML_TRAIN('heatwaveml_bench.census_train', 'revenue',
                           JSON_OBJECT('task','classification',
                                       'exclude_column_list', JSON_ARRAY('age')), @census_model);
  ```

- The `include_column_list` option specifies feature columns that must be considered for training and should not be dropped.

  The following example specifies to consider the `'job'` column when training a model for the `census` dataset.

```
mysql> CALL sys.ML_TRAIN('heatwaveml_bench.census_train', 'revenue',
                         JSON_OBJECT('task','classification',
                                     'include_column_list', JSON_ARRAY('job')), @census_model);
```

- The following example adds `notes` to the `model_metadata`. After training, query the model metadata to confirm the notes were successfully added. Optionally, use `JSON_PRETTY` to view the output in an easily readable format.

```
mysql> CALL sys.ML_TRAIN('bank_test.bank_train', 'y', JSON_OBJECT('task', 'classification', 'notes', 'ban
Query OK, 0 rows affected (1 min 34.7958 sec)

mysql> SELECT JSON_PRETTY(model_metadata) FROM ML_SCHEMA_admin.MODEL_CATALOG WHERE model_handle=@model;
+-----------------------------------------------------------------------------------------------------------
| JSON_PRETTY(model_metadata)
+-----------------------------------------------------------------------------------------------------------
| {
  "task": "classification",
  "notes": "bank marketing model",
  "chunks": 1,
  "format": "HWMLv2.0",
  "n_rows": 300,
  "status": "Ready",
  "options": {
    "task": "classification",
    "notes": "bank marketing model",
    "model_explainer": "permutation_importance",
    "prediction_explainer": "permutation_importance"
  },
  "n_columns": 16,
  "column_names": [
    "age",
    "job",
    "marital",
    "education",
    "default",
    "balance",
    "housing",
    "loan",
    "contact",
    "day",
    "month",
    "duration",
    "campaign",
    "pdays",
    "previous",
    "poutcome"
  ],
  "contamination": null,
  "model_quality": "high",
  "training_time": 70.57345581054688,
  "algorithm_name": "XGBClassifier",
  "training_score": -0.2614343762397766,
  "build_timestamp": 1746109286,
  "n_selected_rows": 240,
  "training_params": {
    "recommend": "ratings",
    "force_use_X": false,
    "recommend_k": 3,
    "remove_seen": true,
    "ranking_topk": 10,
    "lsa_components": 100,
    "ranking_threshold": 1,
    "feedback_threshold": 1
  },
  "train_table_name": "bank_test.bank_train",
  "model_explanation": {
    "permutation_importance": {
      "age": 0.0,
      "day": 0.0,
      "job": 0.0,
      "loan": 0.0,
```

```
      "month": 0.0,
      "pdays": 0.0,
      "balance": 0.0,
      "contact": 0.0175,
      "default": 0.0,
      "housing": 0.0,
      "marital": 0.0,
      "campaign": 0.0,
      "duration": 0.1524,
      "poutcome": 0.0278,
      "previous": 0.0,
      "education": 0.0
    }
  },
  "n_selected_columns": 3,
  "target_column_name": "y",
  "optimization_metric": "neg_log_loss",
  "selected_column_names": [
    "contact",
    "duration",
    "poutcome"
  ],
  "training_drift_metric": {
    "mean": 0.2961,
    "variance": 0.3273
  }
} |
+-----------------------------------------------------------------------------------------------------------------
1 row in set (0.0403 sec)
```

See Also

- Train a Model
- The Model Catalog
- Track Progress for MySQL HeatWave AutoML Routines

10.2.2 ML_EXPLAIN

Running the `ML_EXPLAIN` routine on a model and dataset trains a prediction explainer and model explainer, and adds a model explanation to the model catalog. See Generate Model Explanations and Generate Prediction Explanations to learn more.

MySQL 9.0.0 introduces support for large models that changes how MySQL HeatWave AutoML stores models. See The Model Object Catalog Table. `ML_EXPLAIN` upgrades older models.

`ML_EXPLAIN` does not support the following model types:

- Forecasting
- Recommendation
- Anomaly detection
- Anomaly detection for logs
- Topic Modeling

ML_EXPLAIN Syntax

```
mysql> CALL sys.ML_EXPLAIN ('table_name', 'target_column_name', model_handle, options);

options: {JSON_OBJECT(keyvalue[, keyvalue]...)|NULL}
keyvalue:
{
  'model_explainer', {'permutation_importance'|'partial_dependence'|'shap'|'fast_shap'|NULL}
  |'prediction_explainer', {'permutation_importance'|'shap'}
```

```
  |'columns_to_explain', JSON_ARRAY('column'[,'column'] ...)
  |'target_value', 'target_class'
}
```

When the ML_TRAIN routine runs, ML_EXPLAIN also runs with the Permutation Importance model explainer and prediction explainer. To run ML_EXPLAIN_ROW and ML_EXPLAIN_TABLE with a different explainer, you must first run ML_EXPLAIN with the same explainer. See Generate Model Explanations and Generate Prediction Explanations to learn more.

Required ML_EXPLAIN Parameters

Set the following required parameters:

- table_name: You must define the table that you previously trained. The table name must be valid and fully qualified, so it must include the database name (*database_name*.*table_name*).
- target_column_name: Define the name of the target column in the training dataset that contains ground truth values.
- model_handle: Enter the model handle for the trained model. The model explanation is stored in this model metadata. The model must be loaded first. For example:

  ```
  mysql> CALL sys.ML_MODEL_LOAD('ml_data.iris_train_user1_1636729526', NULL);
  ```

  See Load a Model and Work with Model Handles to learn more.

  If you run ML_EXPLAIN again with the same model handle and model explainer, the model explanation field is overwritten with the new result.

ML_EXPLAIN Options

Optional parameters are specified as key-value pairs in JSON format. If an option is not specified, the default setting is used. If you specify NULL in place of the JSON argument, the default Permutation Importance model explainer is trained, and no prediction explainer is trained.

Set the following options as needed:

- model_explainer: Specifies one of the following model explainers:
  - permutation_importance: The default model explainer.
  - shap: The SHAP model explainer, which produces feature importance values based on Shapley values.
  - fast_shap: The Fast SHAP model explainer, which is a subsampling version of the SHAP model explainer. It usually has a faster runtime.
  - partial_dependence: Explains how changing the values in one or more columns will change the value predicted by the model. The following additional arguments are required:
    - columns_to_explain: A JSON array of one or more column names in the table specified by *table_name*. The model explainer explains how changing the value in this column or columns affects the model.
    - target_value: A valid value that the target column containing ground truth values, as specified by target_column_name, can take.
- prediction_explainer: Specifies one of the following prediction explainers:
  - permutation_importance: The default prediction explainer.
  - shap: The SHAP prediction explainer, which produces feature importance values based on Shapley values.

Syntax Examples

Before running these examples, you must train and load the model first. See Train a Model and Load a Model.

- The following example sets `NULL` for the options, which trains the default Permutation Importance model explainer and no prediction explainer.

```
mysql> CALL sys.ML_EXPLAIN('bank_marketing_test.bank_train', 'y', @bank_test, NULL);
```

- The following example trains the Fast SHAP model explainer and SHAP prediction explainer.

```
mysql> CALL sys.ML_EXPLAIN('bank_marketing_test.bank_train', 'y', @bank_test,
                           JSON_OBJECT('model_explainer', 'fast_shap',
                                       'prediction_explainer', 'shap'));
```

- The following example trains the Partial Dependence model explainer (which requires extra options) and the SHAP prediction explainer. In this example, `sepal width` is the column to explain and the target value to include in `Iris_setosa`.

```
mysql> CALL sys.ML_EXPLAIN('ml_data.iris_train', 'class', @iris_model,
                           JSON_OBJECT('columns_to_explain', JSON_ARRAY('sepal width'),
                                       'target_value', 'Iris-setosa',
                                       'model_explainer', 'partial_dependence',
                                       'prediction_explainer', 'shap'));
```

- You can query the model explanation from the model catalog. The `JSON_PRETTY` parameter displays the output in an easily readable format. See View Model Explanations.

```
mysql> SELECT JSON_PRETTY(model_explanation) FROM ML_SCHEMA_user1.MODEL_CATALOG WHERE model_handle=@c
+-----------------------------------------------------------------------------------------------------------
| JSON_PRETTY(model_explanation)
+-----------------------------------------------------------------------------------------------------------
| {
  "permutation_importance": {
    "age": 0.0292,
    "sex": 0.0023,
    "race": 0.0019,
    "fnlwgt": 0.0038,
    "education": 0.0008,
    "workclass": 0.0068,
    "occupation": 0.0223,
    "capital-gain": 0.0479,
    "capital-loss": 0.0117,
    "relationship": 0.0234,
    "education-num": 0.0352,
    "hours-per-week": 0.0148,
    "marital-status": 0.024,
    "native-country": 0.0
  }
} |
+-----------------------------------------------------------------------------------------------------------
1 row in set (0.0427 sec)
```

See Also

- Generate Model Explanations
- Generate Prediction Explanations
- Track Progress for MySQL HeatWave AutoML Routines

10.2.3 ML_MODEL_EXPORT

MySQL 9.0.0 introduces the `ML_MODEL_EXPORT` routine. Use this routine to export a model from the model catalog to a user defined table.

MySQL 9.0.0 also introduces support for large models that changes how MySQL HeatWave AutoML stores models. See: The Model Object Catalog Table. `ML_MODEL_EXPORT` upgrades older models.

To learn how to use `ML_MODEL_EXPORT` to share models, see Share a Model.

ML_MODEL_EXPORT Overview

After you run `ML_MODEL_EXPORT`, the output table has these columns and formats:

- `chunk_id`:

  `INT AUTO_INCREMENT PRIMARY KEY`

- `model_object`:

  `LONGTEXT DEFAULT NULL`

- `model_metadata`:

  `JSON`

  See Model Metadata.

`ML_MODEL_EXPORT` should work regardless of `model_metadata.status`:

- If there is no corresponding row in the `model_object_catalog` for an existing `model_handle` in the `MODEL_CATALOG`:

  There should be only one row in the output table with `chunk_id` = 0, `model_object` = NULL and `model_metadata` = `MODEL_CATALOG.model_metadata`.

- If there is at least one row in the `model_object_catalog` for an existing `model_handle` in the `MODEL_CATALOG`:

  - There should be N rows in the output table with `chunk_id` being 1 to N.
  - `ML_MODEL_EXPORT` copies the `model_object` from `model_object_catalog` to the output table.
  - `model_metadata` in the row with `chunk_id` = 1 should be the same as in the `MODEL_CATALOG`.

If you use `ML_MODEL_IMPORT` to import an exported model into a different DB System, the results depend on the MySQL version:

- If the DB System has MySQL 9.0.0 or greater, the import should work.
- If the DB System has a MySQL version before MySQL 9.0.0:
  - If the model format is HWMLv2.0 or ONNXv2.0, then the import fails.
  - If the model format is HWMLv1.0, ONNXv1.0 or ONNX, make the following changes:
    - Drop `model_metadata.chunks`.
    - Change the model format to ONNX if the model format is ONNXv1.0.

    The import should then succeed.

ML_MODEL_EXPORT Syntax

```
mysql> CALL sys.ML_MODEL_EXPORT (model_handle, output_table_name);
```

`ML_MODEL_EXPORT` parameters:

- `model_handle`: The model handle for the model. See Work with Model Handles.

- `output_table_name`: The name for the output table.

Syntax Examples

- An example that exports a MySQL HeatWave AutoML model with metadata to the model catalog (ML_SCHEMA_user1.model_export). The output table name is `model_export`. You can then use `SHOW_CREATE_TABLE` to view information on the table for the exported model.

```
mysql> CALL sys.ML_MODEL_EXPORT(@iris_model, 'ML_SCHEMA_user1.model_export');
Query OK, 0 rows affected (0.06 sec)

mysql> SHOW CREATE TABLE ML_SCHEMA_user1.model_export;
+--------------+-----------------------------------------------------------------------+
| Table        | Create Table                                                          |
+--------------+-----------------------------------------------------------------------+
| model_export | CREATE TABLE `model_export` (
`chunk_id` int NOT NULL AUTO_INCREMENT,
`model_object` longtext,
`model_metadata` json DEFAULT NULL,
PRIMARY KEY (`chunk_id`)
) ENGINE=InnoDB AUTO_INCREMENT=2 DEFAULT CHARSET=utf8mb4 COLLATE=utf8mb4_0900_ai_ci |
+--------------+-----------------------------------------------------------------------+
1 row in set (0.00 sec)
```

See Also

- Share a Model
- Manage External ONNX Models

10.2.4 ML_MODEL_IMPORT

Use the `ML_MODEL_IMPORT` routine to import a pre-trained model into your model catalog.

To learn how to use `ML_MODEL_IMPORT` to share models, see Share a Model.

This topic has the following sections.

- ML_MODEL_IMPORT Overview
- ML_MODEL_IMPORT Syntax
- ML_MODEL_IMPORT Parameters
- Syntax Examples
- See Also

ML_MODEL_IMPORT Overview

MySQL HeatWave AutoML supports the import of MySQL HeatWave AutoML and Open Neural Network Exchange (ONNX) format models. After import, all the MySQL HeatWave AutoML routines can be used with an ONNX model.

Models in ONNX format (`.onnx`) cannot be loaded directly into a MySQL table. They require string serialization and conversion to Base64 binary encoding. Before running `ML_MODEL_IMPORT`, follow the instructions in Import an External ONNX Model to pre-process and then load the model into a temporary table for import to MySQL HeatWave.

MySQL 9.0.0 introduces support for large models that changes how MySQL HeatWave AutoML stores models. See The Model Object Catalog Table. `ML_MODEL_IMPORT` upgrades older models.

MySQL 9.0.0 also supports model import from a table. The supported model import formats are ONNX, and all the formats supported by the model catalog. The default import format is ONNX. MySQL HeatWave AutoML verifies models with the ONNX format, and stores them as ONNXv2.0.

The table to import should have the following columns, and their recommended parameters:

- `chunk_id`:

  `INT AUTO_INCREMENT PRIMARY KEY`

- `model_object`:

  `LONGTEXT NOT NULL`

- `model_metadata`:

  `JSON DEFAULT NULL`

  See Model Metadata.

The table must meet the following criteria:

- There must be only one row in the table with `chunk_id` = 1.
- The `model_metadata` corresponding to `chunk_id` = 1 must have the correct JSON key-value pair for the model format.

`ML_MODEL_IMPORT` stores the `model_metadata` corresponding to `chunk_id` = 1 in the model catalog, and ignores the `model_metadata` from other rows.

If `chunks` in the `model_metadata` corresponding to `chunk_id` = 1 is not set, it is set to the number of rows in the input table.

If `ML_MODEL_IMPORT` fails or is cancelled, there is no change to the `MODEL_CATALOG` and to the `model_object_catalog`.

ML_MODEL_IMPORT Syntax

- As of MySQL 9.0.0, `ML_MODEL_IMPORT` can import a model from a table as well as a pre-processed model object. This uses an alternative syntax for `model_metadata`:
- Before MySQL 9.0.0, `ML_MODEL_IMPORT` can only import a pre-processed model object.

```
mysql> CALL sys.ML_MODEL_IMPORT (model_object, model_metadata, model_handle);

model_metadata (model from a table): {JSON_OBJECT(keyvalue[, keyvalue]...)}
keyvalue:
{
  'schema', 'database'
  |'table', 'table'
}

model_metadata (pre-processed model object): {JSON_OBJECT(keyvalue[, keyvalue]...)}
keyvalue:
{
  'task', {'classification'|'regression'|'forecasting'|'anomaly_detection'|'recommendation'|NULL}
  |'build_timestamp', 'timestamp'
  |'target_column_name', 'column'
  |'train_table_name', 'table'
  |'column_names', JSON_ARRAY('column'['column'] ...)
  |'model_explanation', ml_explain_options
  |'notes', 'notes'
  |'format', 'format'
  |'status', {'creating'|'ready'|'error'|NULL}
  |'model_quality', 'quality'
  |'training_time', 'time'
  |'algorithm_name', 'algorithm'
  |'training_score', 'score'
  |'n_rows', 'rows'
  |'n_columns', 'columns'
  |'n_selected_rows', 'rows'
  |'n_selected_columns', 'columns'
```

```
  |'optimization_metric', 'metric'
  |'selected_column_names', JSON_ARRAY('column'[,'column'] ...)
  |'contamination', 'contamination'
  |'options', ml_train_options
  |'training_params', ml_train_params
  |'onnx_inputs_info', data_types_map
  |'onnx_outputs_info', labels_map
  |'training_drift_metric', JSON_OBJECT('mean', 'value', 'variance', 'value')
  |'chunks', 'chunks'
}
```

ML_MODEL_IMPORT Parameters

Set the following parameters:

- `model_object`:
  - To import a model from a table: Set to `NULL`.
  - To import a model object: Define the pre-processed model object.
- `model_metadata`:
  - To import a model from a table:
    - `database`: The name of the database.
    - `table`: The name of the table.
  - To import a model object: An optional JSON object literal that contains key-value pairs with model metadata. See Model Metadata.
- `model_handle`: The model handle for the model. The model is stored in the model catalog under this name and accessed using it. Specify a model handle that does not already exist in the model catalog. Set to `NULL` to generate a unique model handle. See Work with Model Handles.

Syntax Examples

- An example that exports a model to a table, switches users, and then imports the model from that table. To learn more, see Share a Model.

```
mysql> CALL sys.ML_MODEL_EXPORT(@iris_model, 'ML_SCHEMA_user1.model_export');
Query OK, 0 rows affected (0.06 sec)

mysql> SHOW CREATE TABLE ML_SCHEMA_user1.model_export;
+--------------+-------------------------------------------------------------------------+
| Table        | Create Table                                                            |
+--------------+-------------------------------------------------------------------------+
| model_export | CREATE TABLE `model_export` (
`chunk_id` int NOT NULL AUTO_INCREMENT,
`model_object` longtext,
`model_metadata` json DEFAULT NULL,
PRIMARY KEY (`chunk_id`)
) ENGINE=InnoDB AUTO_INCREMENT=2 DEFAULT CHARSET=utf8mb4 COLLATE=utf8mb4_0900_ai_ci |
+--------------+-------------------------------------------------------------------------+
1 row in set (0.00 sec)

# switch to user2

mysql> CALL sys.ML_MODEL_IMPORT(NULL, JSON_OBJECT('schema', 'ML_SCHEMA_user1', 'table', 'model_export
Query OK, 0 rows affected (0.19 sec)

mysql> CALL sys.ML_MODEL_LOAD(@iris_export, NULL);
Query OK, 0 rows affected (0.63 sec)

mysql> SELECT model_object, model_object_size FROM ML_SCHEMA_user2.MODEL_CATALOG WHERE model_handle=@
+--------------+-------------------+
| model_object | model_object_size |
```

```
+--------------+-------------------+
| NULL         |            348954 |
+--------------+-------------------+
1 row in set (0.00 sec)


mysql> SELECT chunk_id, LENGTH(model_object) FROM ML_SCHEMA_user2.model_object_catalog WHERE model_handle
+----------+----------------------+
| chunk_id | LENGTH(model_object) |
+----------+----------------------+
|        1 |               348954 |
+----------+----------------------+
1 row in set (0.00 sec)
```

- An example that imports a model in ONNX format from a table. To learn more, see Import an External ONNX Model.

```
mysql> DROP TABLE IF EXISTS model_table;

mysql> CREATE TABLE model_table (
        chunk_id INT AUTO_INCREMENT PRIMARY KEY,
        model_object LONGTEXT NOT NULL,
        model_metadata JSON DEFAULT NULL);

mysql> LOAD DATA INFILE '/onnx_examples/x00'
        INTO TABLE model_table
        CHARACTER SET binary
        FIELDS TERMINATED BY '\t'
        LINES TERMINATED BY '\r'
        (model_object);
Query OK, 1 row affected (34.96 sec)
Records: 1  Deleted: 0  Skipped: 0  Warnings: 0

mysql> LOAD DATA INFILE '/onnx_examples/x01'
        INTO TABLE model_table
        CHARACTER SET binary
        FIELDS TERMINATED BY '\t'
        LINES TERMINATED BY '\r'
        (model_object);
Query OK, 1 row affected (32.74 sec)
Records: 1  Deleted: 0  Skipped: 0  Warnings: 0

mysql> LOAD DATA INFILE '/onnx_examples/x02'
        INTO TABLE model_table
        CHARACTER SET binary
        FIELDS TERMINATED BY '\t'
        LINES TERMINATED BY '\r'
        (model_object);
Query OK, 1 row affected (11.90 sec)
Records: 1  Deleted: 0  Skipped: 0  Warnings: 0

mysql> SET @model_metadata = JSON_OBJECT('task','classification',
                                         'onnx_outputs_info', JSON_OBJECT('predictions_name','label',
                                                                          'prediction_probabilities_name
                                         'target_column_name','target');

mysql> UPDATE mlcorpus.model_table SET model_metadata=@model_metadata WHERE chunk_id=1;

mysql> CALL sys.ML_MODEL_IMPORT(NULL, JSON_OBJECT('schema', 'mlcorpus', 'table', 'model_table'), @onnx_mo
Query OK, 0 rows affected (18 min 7.29 sec)

mysql> CALL sys.ML_MODEL_LOAD(@onnx_model, NULL);
Query OK, 0 rows affected (6 min 51.37 sec)

mysql> SELECT COUNT(*) FROM ML_SCHEMA_root.model_object_catalog WHERE model_handle=@onnx_model;
+----------+
| COUNT(*) |
+----------+
|        3 |
+----------+
1 row in set (0.01 sec)
```

```
mysql> SELECT SUM(LENGTH(model_object)) FROM ML_SCHEMA_root.model_object_catalog WHERE model_handle=@
+--------------------------+
| SUM(LENGTH(model_object)) |
+--------------------------+
|                2148494845 |
+--------------------------+
1 row in set (57.36 sec)
```

See Also

- Share a Model
- Import an External ONNX Model

10.2.5 ML_PREDICT_ROW

`ML_PREDICT_ROW` generates predictions for one or more rows of unlabeled data specified in `JSON` format. Invoke `ML_PREDICT_ROW` with a `SELECT` statement.

A call to `ML_PREDICT_ROW` can include columns that were not present during `ML_TRAIN`. A table can include extra columns, and still use the MySQL HeatWave AutoML model. This allows side by side comparisons of target column labels, ground truth, and predictions in the same table. `ML_PREDICT_ROW` ignores any extra columns, and appends them to the results.

`ML_PREDICT_ROW` does not support the following model types:

- Forecasting
- Anomaly detection for logs
- Recommendation models trained with the `TwoTower` mdoel.

This topic has the following sections.

- ML_PREDICT_ROW Syntax
- Required ML_PREDICT_ROW Parameters
- ML_PREDICT ROW Option for Data Drift Detection
- ML_PREDICT_ROW Options for Recommendation Models
- Options for Anomaly Detection Models
- Syntax Examples
- See Also

ML_PREDICT_ROW Syntax

```
mysql> SELECT sys.ML_PREDICT_ROW(input_data, model_handle), options);

options: {JSON_OBJECT(keyvalue[, keyvalue]...)|NULL}
keyvalue:
{
  'threshold', 'N'
  |'topk', 'N'
  |'recommend', {'ratings'|'items'|'users'|'users_to_items'|'items_to_users'|'items_to_items'|'users_to
  |'remove_seen', {'true'|'false'}
  |'additional_details', {'true'|'false'}
}
```

Required ML_PREDICT_ROW Parameters

Set the following required parameters:

- `input_data`: Define the data to generate predictions for. The column names must match the feature column names in the table used to train the model. You can define the input data in the following ways:

  Specify a single row of data in `JSON` format.

  ```
  mysql> SELECT sys.ML_PREDICT_ROW(JSON_OBJECT("column_name", value, "column_name", value, ...), model_hand
  ```

  Run `ML_PREDICT_ROW` on multiple rows of data by specifying the columns as key-value pairs in `JSON` format and select from a table.

  ```
  mysql> SELECT sys.ML_PREDICT_ROW(JSON_OBJECT("output_col_name", schema.`input_col_name`, "output_col_name
         model_handle, options) FROM input_table_name LIMIT N;
  ```

- `model_handle`: Define the model handle or a session variable that contains the model handle. See Work with Model Handles.

Review the following options in `JSON` format.

ML_PREDICT ROW Option for Data Drift Detection

To view data drift detection values for classification, regression, and anomaly detection (as of MySQL 9.3.2) models, set the `additional_details` option to `true`. The `ml_results` includes the `drift` JSON object literal. See Analyze Data Drift.

ML_PREDICT_ROW Options for Recommendation Models

Set the following options as needed for Recommendation models.

- `topk`: Specify the number of recommendations to provide as a positive integer. The default is `3`.
- `recommend`: Specify what to recommend.
  - `ratings`: Use this option to predict ratings. This is the default value.

    The target column is `prediction`, and the values are `float`.

    The input table must contain at least two columns with the same names as the user column and item column from the training model.
  - `items`: Use this option to recommend items for users.

    The target column is `item_recommendation`, and the values are:

    ```
    JSON_OBJECT("column_item_id_name", JSON_ARRAY("item_1", ... , "item_k"), "column_rating_name" , JSON_AR
    ```

    The input table must contain at least one column with the same name as the user column from the training model.
  - `users`: Use this option to recommend users for items.

    The target column is `user_recommendation`, and the values are:

    ```
    JSON_OBJECT("column_user_id_name", JSON_ARRAY("user_1", ... , "user_k"), "column_rating_name" , JSON_AR
    ```

    The input table must contain at least one column with the same name as the item column from the training model.
  - `users_to_items`: This is the same as `items`.
  - `items_to_users`: This is the same as `users`.
  - `items_to_items`: Use this option to recommend similar items for items.

The target column is `item_recommendation`, and the values are:

```
JSON_OBJECT("column_item_id_name", JSON_ARRAY("item_1", ... , "item_k"))
```

The input table must contain at least contain a column with the same name as the item column from the training model.

- `users_to_users`: Use this option to recommend similar users for users.

  The target column is `user_recommendation`, and the values are:

  ```
  JSON_OBJECT("column_user_id_name", JSON_ARRAY("user_1", ... , "user_k"))
  ```

  The input table must contain at least one column with the same name as the user column from the training model.

- `remove_seen`: If the input table overlaps with the training table, and `remove_seen` is `true`, then the model will not repeat existing interactions. The default is `true`. Set `remove_seen` to `false` to repeat existing interactions from the training table.

Options for Anomaly Detection Models

Set the following options as needed for anomaly detection models.

- `threshold`: The threshold you set on anomaly detection models determines which rows in the output table are labeled as anomalies with an anomaly score of `1`, or normal with an anomaly score of `0`. The value for the threshold is the degree to which a row of data or log segment is considered for anomaly detection. Any sample with an anomaly score above the threshold is classified an anomaly. The default value is (1 - `contamination`)-th percentile of all the anomaly scores.

Syntax Examples

- The following example generates a prediction on a single row of data. The results include the `ml_results` field, which uses `JSON` format. Optionally use `\G` to display the information in an easily readable format.

```
mysql> SET @row_input = JSON_OBJECT(
          "age", 25,
          "workclass", "Private",
          "fnlwgt", 226802,
          "education", "11th",
          "education-num", 7,
          "marital-status", "Never-married",
          "occupation", "Machine-op-inspct",
          "relationship", "Own-child",
          "race", "Black",
          "sex", "Male",
          "capital-gain", 0,
          "capital-loss", 0,
          "hours-per-week", 40,
          "native-country", "United-States");
mysql> SELECT sys.ML_PREDICT_ROW(@row_input, @census_model, NULL)\G
*************************** 1. row ***************************
sys.ML_PREDICT_ROW(@row_input, @census_model, NULL):
{
    "age": 25,
    "sex": "Male",
    "race": "Black",
    "fnlwgt": 226802,
    "education": "11th",
    "workclass": "Private",
    "Prediction": "<=50K",
    "ml_results": {
        "predictions": {
            "revenue": "<=50K"
        },
```

```
        "probabilities": {
            ">50K": 0.0032,
            "<=50K": 0.9968
        }
    },
    "occupation": "Machine-op-inspct",
    "capital-gain": 0,
    "capital-loss": 0,
    "relationship": "Own-child",
    "education-num": 7,
    "hours-per-week": 40,
    "marital-status": "Never-married",
    "native-country": "United-States"
}
1 row in set (2.2218 sec)
```

- The following example generates predictions on two rows of data from the input table. Optionally use `\G` to display the information in an easily readable format.

```
mysql> SELECT sys.ML_PREDICT_ROW(JSON_OBJECT(
 "age", census_train.`age`,
 "workclass", census_train.`workclass`,
 "fnlwgt", census_train.`fnlwgt`,
 "education", census_train.`education`,
 "education-num", census_train.`education-num`,
 "marital-status", census_train.`marital-status`,
 "occupation", census_train.`occupation`,
 "relationship", census_train.`relationship`,
 "race", census_train.`race`,
 "sex", census_train.`sex`,
 "capital-gain", census_train.`capital-gain`,
 "capital-loss", census_train.`capital-loss`,
 "hours-per-week", census_train.`hours-per-week`,
 "native-country", census_train.`native-country`),
 @census_model, NULL)FROM census_data.census_train LIMIT 2\G
*************************** 1. row ***************************
sys.ML_PREDICT_ROW(JSON_OBJECT(
"age", census_train.`age`,
"workclass", census_train.`workclass`,
"fnlwgt", census_train.`fnlwgt`,
"education", census_train.`education`,
"education-num", census_train.`education-num`,
"marital-status", census_train.`marita: {
                                        "age": 62,
                                        "sex": "Female",
                                        "race": "White",
                                        "fnlwgt": 123582,
                                        "education": "10th",
                                        "workclass": "Private",
                                        "Prediction": "<=50K",
                                        "ml_results": {
                                            "predictions": {
                                                "revenue": "<=50K"
                                            },
                                            "probabilities": {
                                                ">50K": 0.0106,
                                                "<=50K": 0.9894
                                            }
                                        },
                                        "occupation": "Other-service",
                                        "capital-gain": 0,
                                        "capital-loss": 0,
                                        "relationship": "Unmarried",
                                        "education-num": 6,
                                        "hours-per-week": 40,
                                        "marital-status": "Divorced",
                                        "native-country": "United-States"
                                    }
*************************** 2. row ***************************
sys.ML_PREDICT_ROW(JSON_OBJECT(
"age", census_train.`age`,
"workclass", census_train.`workclass`,
```

```
"fnlwgt", census_train.`fnlwgt`,
"education", census_train.`education`,
"education-num", census_train.`education-num`,
"marital-status", census_train.`marita: {
                                        "age": 32,
                                        "sex": "Female",
                                        "race": "White",
                                        "fnlwgt": 174215,
                                        "education": "Bachelors",
                                        "workclass": "Federal-gov",
                                        "Prediction": "<=50K",
                                        "ml_results": {
                                            "predictions": {
                                                "revenue": "<=50K"
                                            },
                                            "probabilities": {
                                                ">50K": 0.3249,
                                                "<=50K": 0.6751
                                            }
                                        },
                                        "occupation": "Exec-managerial",
                                        "capital-gain": 0,
                                        "capital-loss": 0,
                                        "relationship": "Not-in-family",
                                        "education-num": 13,
                                        "hours-per-week": 60,
                                        "marital-status": "Never-married",
                                        "native-country": "United-States"
                                    }
2 rows in set (9.6548 sec)
```

- The following example uses explicit feedback and runs the `ML_PREDICT_ROW` routine to predict the top 3 items that a particular user will like with the `users_to_items` option.

```
mysql> SELECT sys.ML_PREDICT_ROW('{"user_id": "846"}', @model, JSON_OBJECT("recommend", "users_to_ite
+-------------------------------------------------------------------------------------------------------
| sys.ML_PREDICT_ROW('{"user_id": "846"}', @model,  JSON_OBJECT("recommend", "users_to_items", "topk"
+-------------------------------------------------------------------------------------------------------
| {"user_id": "846", "ml_results": "{"predictions": {"item_id": ["313", "483", "64"], "rating": [4.06
+-------------------------------------------------------------------------------------------------------
1 row in set (0.2811 sec)
```

- The following example generates predictions on ten rows from an input table. The `additional_details` parameter is set to `TRUE, so you can review data drift details.`

```
mysql> SELECT sys.ML_PREDICT_ROW(JSON_OBJECT(
 "age", census_test.`age`,
 "workclass", census_test.`workclass`,
 "fnlwgt", census_test.`fnlwgt`,
 "education", census_test.`education`,
 "education-num", census_test.`education-num`,
 "marital-status", census_test.`marital-status`,
 "occupation", census_test.`occupation`,
 "relationship", census_test.`relationship`,
 "race", census_test.`race`,
 "sex", census_test.`sex`,
 "capital-gain", census_test.`capital-gain`,
 "capital-loss", census_test.`capital-loss`,
 "hours-per-week", census_test.`hours-per-week`,
 "native-country", census_test.`native-country`),
 @census_model, JSON_OBJECT('additional_details', TRUE))FROM census_data.census_test LIMIT 10;
+-------------------------------------------------------------------------------------------------------
| sys.ML_PREDICT_ROW(JSON_OBJECT(
"age", census_test.`age`,
"workclass", census_test.`workclass`,
"fnlwgt", census_test.`fnlwgt`,
"education", census_test.`education`,
"education-num", census_test.`education-num`,
"ma
+-------------------------------------------------------------------------------------------------------
```

```
| {"age": 37, "sex": "Male", "race": "White", "fnlwgt": 99146, "education": "Bachelors", "workclass": "Pr
| {"age": 34, "sex": "Male", "race": "White", "fnlwgt": 27409, "education": "9th", "workclass": "Private"
| {"age": 30, "sex": "Female", "race": "White", "fnlwgt": 299507, "education": "Assoc-acdm", "workclass"
| {"age": 62, "sex": "Female", "race": "White", "fnlwgt": 102631, "education": "Some-college", "workclas
| {"age": 51, "sex": "Male", "race": "White", "fnlwgt": 153486, "education": "Some-college", "workclass"
| {"age": 34, "sex": "Male", "race": "Black", "fnlwgt": 434292, "education": "HS-grad", "workclass": "Pri
| {"age": 28, "sex": "Male", "race": "White", "fnlwgt": 240172, "education": "Masters", "workclass": "Sel
| {"age": 56, "sex": "Male", "race": "White", "fnlwgt": 219426, "education": "10th", "workclass": "Privat
| {"age": 46, "sex": "Female", "race": "White", "fnlwgt": 295791, "education": "HS-grad", "workclass": "P
| {"age": 46, "sex": "Male", "race": "White", "fnlwgt": 114032, "education": "Some-college", "workclass"
+-----------------------------------------------------------------------------------------------------------
10 rows in set (6.8109 sec)
```

- The following example uses a recommendation model with implicit feedback to predict the ranking for a particular user and item.

```
mysql> SELECT sys.ML_PREDICT_ROW('{"user_id": "836", "item_id": "226"}', @model,  NULL);
+-----------------------------------------------------------------------------------+
| sys.ML_PREDICT_ROW('{"user_id": "836", "item_id": "226"}', @model,  NULL)         |
+-----------------------------------------------------------------------------------+
| {"item_id": "226", "user_id": "836", "ml_results": {"predictions": {"rating": 2.46}}} |
+-----------------------------------------------------------------------------------+
1 row in set (0.1390 sec)
```

- The following example uses a recommendation model with explicit feedback to predict the top two items that a particular user will like with the `users_to_items` option.

```
mysql> SELECT sys.ML_PREDICT_ROW('{"user_id": "846"}', @model, JSON_OBJECT("recommend", "users_to_items",
+-----------------------------------------------------------------------------------------------------------
| sys.ML_PREDICT_ROW('{"user_id": "846"}', @model,  JSON_OBJECT("recommend", "users_to_items", "topk", 2)
+-----------------------------------------------------------------------------------------------------------
| {"user_id": "846", "ml_results": "{"predictions": {"item_id": ["313", "483"], "rating": [4.06, 4.05]}}"
+-----------------------------------------------------------------------------------------------------------
1 row in set (0.2811 sec)
```

- The following example uses a recommendation model with explicit feedback to predict the top two items similar to another item.

```
mysql> SELECT sys.ML_PREDICT_ROW('{"item_id": "524"}', @model, JSON_OBJECT("recommend", "items_to_items",
+-----------------------------------------------------------------------------------------------------------
| sys.ML_PREDICT_ROW('{"item_id": "524"}', @model,  JSON_OBJECT("recommend", "items_to_items", "topk", 2)
+-----------------------------------------------------------------------------------------------------------
| {"item_id": "524", "ml_results": "{"predictions": {"item_id": ["665", "633"], "similarity": [1.0, 1.0]}
+-----------------------------------------------------------------------------------------------------------
```

See Also

- Generate Predictions for a Row of Data
- Analyze Data Drift

10.2.6 ML_PREDICT_TABLE

`ML_PREDICT_TABLE` generates predictions for an entire table of unlabeled data and saves the results to an output table. MySQL HeatWave AutoML performs the predictions in parallel.

This topic has the following sections.

- ML_PREDICT_TABLE Overview
- ML_PREDICT_TABLE Syntax
- Required ML_PREDICT_TABLE Parameters
- ML_PREDICT_TABLE Options
- Options for Recommendation Models
- Requirements and Options for Anomaly Detection Models

- Options for Forecasting Models
- Syntax Examples
- See Also

ML_PREDICT_TABLE Overview

`ML_PREDICT_TABLE` is a compute intensive process. Depending on your MySQL version, we recommend the following:

- Before MySQL 9.4.1, limit operations to batches of rows by splitting large tables into smaller tables by using the `batch_size` option.
- For MySQL 9.4.1 and later, if `ML_PREDICT_TABLE` takes a long time to complete, manually limit input tables to a maximum of 1,000 rows.

A call to `ML_PREDICT_TABLE` can include columns that were not present during `ML_TRAIN`. A table can include extra columns, and still use the MySQL HeatWave AutoML model. This allows side by side comparisons of target column labels, ground truth, and predictions in the same table. `ML_PREDICT_TABLE` ignores any extra columns, and appends them to the results.

The output table includes a primary key:

- If the input table has a primary key, the output table has the same primary key.
- If the input table does not have a primary key, the output table has a new primary key column that auto increments.
  - As of MySQL 8.4.1, the name of the new primary key column is `_4aad19ca6e_pk_id`. The input table must not have a column with the name `_4aad19ca6e_pk_id` that is not a primary key.
  - Before MySQL 8.4.1, the name of the new primary key column is `_id`. The input table must not have a column with the name `_id` that is not a primary key.

The output of predictions includes the `ml_results` column, which contains the prediction results and the data. The combination of results and data must be less than 65,532 characters.

As of MySQL 9.4.1, you have the option to specify the input table and output table as the same table if specific conditions are met. See Input Tables and Output Tables to learn more.

`ML_PREDICT_TABLE` supports data drift detection for classification, regression, and anomaly detection (as of MySQL 9.3.2) models with the following:

- The *`options`* parameter includes the `additional_details` boolean value.
- The `ml_results` column includes the `drift` JSON object literal.

See Analyze Data Drift.

ML_PREDICT_TABLE Syntax

```
mysql> CALL sys.ML_PREDICT_TABLE(table_name, model_handle, output_table_name), options);

options: {JSON_OBJECT(keyvalue[, keyvalue]...)|NULL}
keyvalue:
{
  'threshold', 'N'
  |'topk', 'N'
  |'recommend', {'ratings'|'items'|'users'|'users_to_items'|'items_to_users'|'items_to_items'|'users_to
  |'remove_seen', {'true'|'false'}
  |'batch_size', 'N'
  |'additional_details', {'true'|'false'}
  |'prediction_interval', 'N'
  |'item_metadata', JSON_OBJECT('table_name'['database_name.table_name'] ...)
  |'user_metadata', JSON_OBJECT('table_name'['database_name.table_name'] ...)
```

```
  |'logad_options', JSON_OBJECT(("key","value"[,"key","value"] ...)
}

logad_options: {JSON_OBJECT(keyvalue[, keyvalue]...)}
keyvalue:
{
  summarize_logs', {'true'|'false'}
  |'summary_threshold', 'N'
```

Required ML_PREDICT_TABLE Parameters

Set the following required parameters:

- `table_name`: Specifies the fully qualified name of the input table (*`database_name.table_name`*). The input table should contain the same feature columns as the training dataset. If the target column is included in the input table, it is not considered when generating predictions.
- `model_handle`: Specifies the model handle or a session variable containing the model handle. See Work with Model Handles.
- `output_table_name`: Specifies the table where predictions are stored. A fully qualified table name must be specified (*`database_name.table_name`*). As of MySQL 9.4.1, you have the option to specify the input table and output table as the same table if specific conditions are met. See Input Tables and Output Tables to learn more.

ML_PREDICT_TABLE Options

Set the following options in `JSON` format as needed.

- To view data drift detection values for classification and regression models, set the `additional_details` option to `true`. The `ml_results` includes the `drift` JSON object literal.
- `batch_size`: Deprecated as of MySQL 9.4.1. The size of each batch. You can set a value between 1 and 1,000. The default is 1,000, and this provides the best results. If on MySQL 9.4.1 and later and `ML_PREDICT_TABLE` takes a long time to complete, manually limit input tables to a maximum of 1,000 rows.

Additional options are available for recommendation, anomaly detection, and forecasting models.

Options for Recommendation Models

Set the following options as needed for recommendation models.

- `threshold`: The optional threshold that defines positive feedback, and a relevant sample. Only use with ranking metrics. It can be used for either explicit or implicit feedback.
- `topk`: The optional top number of recommendations to provide. The default is `3`. Set a positive integer between 1 and the number of rows in the table.

  A `recommendation` task with implicit feedback can use both `threshold` and `topk`.
- `recommend`: Specify what to recommend.
  - `ratings`: Use this option to predict ratings. This is the default value.

    The target column is `prediction`, and the values are `float`.

    The input table must contain at least two columns with the same names as the user column and item column from the training model.
  - `items`: Use this option to recommend items for users.

    The target column is `item_recommendation`, and the values are:

    ```
    JSON_OBJECT("column_item_id_name", JSON_ARRAY("item_1", ... , "item_k"), "column_rating_name" , JSON_AR
    ```

The input table must contain at least one column with the same name as the user column from the training model.

- `users`: Use this option to recommend users for items.

  The target column is `user_recommendation`, and the values are:

  ```
  JSON_OBJECT("column_user_id_name", JSON_ARRAY("user_1", ... , "user_k"), "column_rating_name" , JSO
  ```

  The input table must contain at least one column with the same name as the item column from the training model.

- `users_to_items`: This is the same as `items`.
- `items_to_users`: This is the same as `users`.
- `items_to_items`: Use this option to recommend similar items for items.

  The target column is `item_recommendation`, and the values are:

  ```
  JSON_OBJECT("column_item_id_name", JSON_ARRAY("item_1", ... , "item_k"))
  ```

  The input table must contain at least one column with the same name as the item column from the training model.

- `users_to_users`: Use this option to recommend similar users for users.

  The target column is `user_recommendation`, and the values are:

  ```
  JSON_OBJECT("column_user_id_name", JSON_ARRAY("user_1", ... , "user_k"))
  ```

  The input table must at least contain a column with the same name as the user column from the training model.

- `remove_seen`: If the input table overlaps with the training table, and `remove_seen` is `true`, then the model will not repeat existing interactions. The default is `true`. Set `remove_seen` to `false` to repeat existing interactions from the training table.
- `item_metadata`: As of MySQL 9.5.0, it defines the table that has item descriptions. It is a JSON object that has the `table_name` option as a key, which specifies the table that has item descriptions. One column must be the same as the `item_id` in the input table.
- `user_metadata`: As of MySQL 9.5.0, it defines the table that has user descriptions. It is a JSON object that has the `table_name` option as a key, which specifies the table that has user descriptions. One column must be the same as the `user_id` in the input table.
  - `table_name`: To be used with the `item_metadata` and `user_metadata` options. It specifies the table name that has item or user descriptions. It must be a string in a fully qualified format (schema_name.table_name) that specifies the table name.

Requirements and Options for Anomaly Detection Models

As of MySQL 9.4.1, if you run `ML_PREDICT_TABLE` with the `log_anomaly_detection` task, at least one column must act as the primary key to establish the temporal order of logs.

Set the following options as needed for anomaly detection models.

- `threshold`: The threshold you set on anomaly detection models determines which rows in the output table are labeled as anomalies with an anomaly score of `1`, or normal with an anomaly score of `0`. The value for the threshold is the degree to which a row of data or log segment is considered for anomaly detection. Any sample with an anomaly score above the threshold is classified an anomaly. The default value is (1 - `contamination`)-th percentile of all the anomaly scores.

- `topk`: The optional top K rows to display with the highest anomaly scores. Set a positive integer between 1 and the number of rows in the table. If `topk` is not set, `ML_PREDICT_TABLE` uses `threshold`.

  Do not set both `threshold` and `topk`. Use `threshold` or `topk`, or set *`options`* to `NULL`.

- `logad_options`: A `JSON_OBJECT` that allows you to configure the following options for running an anomaly detection model on log data (MySQL 9.2.2 and later).

  - `summarize_logs`: Allows you to leverage MySQL HeatWave GenAI to generate textual summaries of results. Enable this option by setting it to `TRUE`. If enabled, summaries are generated for log segments that are labeled as an anomaly or have anomaly scores higher than the value set for the `summary_threshold`.

  - `summary_threshold`: Determines the rows in the output table that are summarized. This does not affect how the `contamination` and `threshold` options determine anomalies. You can set a value greater than 0 and less than 1. The default value is `NULL`. If `NULL` is selected, only the log segments tagged with `is_anomaly` are used to generate summaries.

Options for Forecasting Models

Set the following options as needed for forecasting models.

- `prediction_interval`: Use this to generate forecasted values with lower and upper bounds based on a specific prediction interval (level of confidence). For the `prediction_interval` value:

  - The default value is 0.95.

  - The data type for this value must be FLOAT.

  - The value must be greater than 0 and less than 1.

Syntax Examples

- A typical usage example that specifies the fully qualified name of the table to generate predictions for, the session variable containing the model handle, and the fully qualified output table name.

```
mysql> CALL sys.ML_PREDICT_TABLE('census_data.census_train', @census_model, 'census_data.census_train_pre
```

  To view `ML_PREDICT_TABLE` results, query the output table. The table shows the predictions and the feature column values used to make each prediction. The table includes the primary key, `_4aad19ca6e_pk_id`, and the `ml_results` column, displays the predictions and probabilities for each prediction.

```
mysql> SELECT * FROM census_train_predictions LIMIT 5;
+------------------+-----+------------------+--------+-------------+---------------+------------------
| _4aad19ca6e_pk_id | age | workclass        | fnlwgt | education    | education-num | marital-status
+------------------+-----+------------------+--------+-------------+---------------+------------------
|                1 |  37 | Private          |  99146 | Bachelors    |            13 | Married-civ-spouse
|                2 |  34 | Private          |  27409 | 9th          |             5 | Married-civ-spouse
|                3 |  30 | Private          | 299507 | Assoc-acdm   |            12 | Separated
|                4 |  62 | Self-emp-not-inc | 102631 | Some-college |            10 | Widowed
|                5 |  51 | Private          | 153486 | Some-college |            10 | Married-civ-spouse
+------------------+-----+------------------+--------+-------------+---------------+------------------
5 rows in set (0.0014 sec)
```

- The following example generates a table of recommendations. The output recommends the top three items that particular users will like.

```
mysql> CALL sys.ML_PREDICT_TABLE('mlcorpus.test_sample',  @model, 'mlcorpus.table_predictions_users',
                                 JSON_OBJECT("recommend", "items", "topk", 3));
Query OK, 0 rows affected (5.0672 sec)

mysql> SELECT * FROM mlcorpus.table_predictions_users LIMIT 3;
+------------------+---------+--------+--------+-----------------------------------------------------
```

```
| _4aad19ca6e_pk_id | user_id | item_id | rating | ml_results
+-------------------+---------+---------+--------+-----------------------------------------------
|                 1 | 1026    | 13763   |      1 | {"predictions": {"item_id": ["10", "14", "11"], "r
|                 2 | 992     | 16114   |      1 | {"predictions": {"item_id": ["10", "14", "11"], "r
|                 3 | 1863    | 4527    |      1 | {"predictions": {"item_id": ["10", "14", "11"], "r
+-------------------+---------+---------+--------+-----------------------------------------------
```

- The following example generates a table of anomaly detection predictions. A threshold value of 1% is specified. The `ml_results` column displays the prediction if each row is an anomaly, and the probability for that prediction.

```
mysql> CALL sys.ML_PREDICT_TABLE('volcano_data.volcano_data_train', 'anomaly_model', 'volcano_data.vo
                                 JSON_OBJECT('threshold', 0.01));
Query OK, 0 rows affected (4.8868 sec)

mysql> SELECT * FROM mlcorpus_anomaly_detection.volcanoes-predictions_threshold LIMIT 5;
+-------------------+-------+-------+----------+--------+-----------------------------------------
| _4aad19ca6e_pk_id | V1    | V2    | V3       | target | ml_results
+-------------------+-------+-------+----------+--------+-----------------------------------------
|                 1 | 806.0 | 962.0 | 0.353207 | n      | {"predictions": {"is_anomaly": 1}, "probabi
|                 2 | 326.0 | 254.0 | 0.368590 | n      | {"predictions": {"is_anomaly": 1}, "probabi
|                 3 | 586.0 | 654.0 | 0.409559 | n      | {"predictions": {"is_anomaly": 1}, "probabi
|                 4 | 370.0 | 816.0 | 0.404861 | n      | {"predictions": {"is_anomaly": 1}, "probabi
|                 5 | 918.0 | 952.0 | 0.419940 | n      | {"predictions": {"is_anomaly": 1}, "probabi
+-------------------+-------+-------+----------+--------+-----------------------------------------
5 rows in set (0.0409 sec)
```

- The following example generates a table of anomaly detection predictions by using semi-supervised learning. It overrides the `ensemble_score` value from the `ML_TRAIN` routine to a new value of 0.5. The `ml_results` column displays the prediction if each row is an anomaly, and the probability for that prediction.

```
mysql> CALL sys.ML_PREDICT_TABLE('anomaly_data.anomaly_data_train', 'semisupervised_model',
                                 'anomaly_data.anomaly_predictions_semisupervised',
                                 CAST('{"experimental": {"semisupervised": {"supervised_submodel_wei
Query OK, 0 rows affected (1.5698 sec)
mysql> SELECT * FROM anomaly_predictions_semisupervised LIMIT 5;
+-------------------+----------------------+---------------------+---------------------+-------------
| _4aad19ca6e_pk_id | att1                 | att2                | att3                | att4
+-------------------+----------------------+---------------------+---------------------+-------------
|                 1 | 0.114285714285714280 | 0.50000000000000000 | 0.46808510638297873 | 0.0201454952
|                 2 | 0.171428571428571430 | 0.47159090909090910 | 0.25531914893617020 | 0.0179071068
|                 3 | 0.142857142857142850 | 0.50568181818181820 | 0.12765957446808510 | 0.0201454952
|                 4 | 0.114285714285714280 | 0.47727272727272730 | 0.25531914893617020 | 0.0201454952
|                 5 | 0.114285714285714280 | 0.47727272727272730 | 0.34042553191489360 | 0.0167879127
+-------------------+----------------------+---------------------+---------------------+-------------
5 rows in set (0.0491 sec)
```

- The following example generates a table of anomaly detection predictions for log data (MySQL 9.2.2 and later). It disables log summaries in the results.

```
mysql> CALL sys.ML_PREDICT_TABLE('mlcorpus.`log_anomaly_just_patterns`', @logad_model, 'mlcorpus.log_
                                 JSON_OBJECT('logad_options', JSON_OBJECT('summarize_logs', FALSE)))
mysql> SELECT * FROM mlcorpus.log_anomaly_test_out LIMIT 1;
+----+-----------------------------------------------------------------------------------------------
| id | parsed_log_segment
+----+-----------------------------------------------------------------------------------------------
|  1 | 2024-04-11T14:39:45.443597Z 1 [Note] [MY-013546] [InnoDB] Atomic write enabled
|    | 2024-04-11T14:39:45.443618Z 1 [Note] [MY-012932] [InnoDB] PUNCH HOLE support available
|    | 2024-04-11T14:39:45.443631Z 1 [Note] [MY-012944] [InnoDB] Uses event mutexes
|    | 2024-04-11T14:39:45.443635Z 1 [Note] [MY-012945] [InnoDB] GCC builtin __atomic_thread_fence()
|    | 2024-04-11T14:39:45.443646Z 1 [Note] [MY-012948] [InnoDB] Compressed tables use zlib 1.2.13
|    | 2024-04-11T14:40:25.128143Z 0 [Note] [MY-010264] [Server] - '127.0.0.1' resolves to '127.0.0.1
|    | 2024-04-11T14:40:25.128182Z 0 [Note] [MY-010251] [Server] Server socket created on IP: '127.0.
|    | 2024-04-11T14:40:25.128245Z 0 [Note] [MY-010252] [Server] Server hostname (bind-address): '10.
|    | 2024-04-11T14:40:25.128272Z 0 [Note] [MY-010264] [Server] - '10.0.1.125' resolves to '10.0.1.1
|    | 2024-04-26T13:01:30.287325Z 0 [Warning] [MY-015116] [Server] Background histogram update on ne
|    | Lock wait timeout exceeded; try restarting transaction
+----+------+------+----------+--------+-----------------------------------------------------------
```

See Also

- Generate Predictions for a Table
- Analyze Data Drift

10.2.7 ML_EXPLAIN_ROW

The `ML_EXPLAIN_ROW` routine generates explanations for one or more rows of unlabeled data. Invoke `ML_EXPLAIN_ROW` with a `SELECT` statement. It limits explanations to the 100 most relevant features.

A loaded and trained model with the appropriate prediction explainer is required to run `ML_EXPLAIN_ROW`. See Generate Prediction Explanations for a Row of Data.

`ML_EXPLAIN_ROW` does not support recommendation, anomaly detection and topic modeling models. A call with one of these models produces an error.

A call to `ML_EXPLAIN_ROW` can include columns that were not present during `ML_TRAIN`. A table can include extra columns, and still use the MySQL HeatWave AutoML model. This allows side by side comparisons of target column labels, ground truth, and explanations in the same table. `ML_EXPLAIN_ROW` ignores any extra columns, and appends them to the results.

`ML_EXPLAIN_ROW` does not support the following model types:

- Forecasting
- Recommendation
- Anomaly detection
- Anomaly detection for logs
- Topic modeling

This topic has the following sections.

- ML_EXPLAIN_ROW Syntax
- Required ML_EXPLAIN_ROW Parameters
- ML_EXPLAIN_ROW Options
- Syntax Examples
- See Also

ML_EXPLAIN_ROW Syntax

```
mysql> SELECT sys.ML_EXPLAIN_ROW(input_data, model_handle, options);

options: {JSON_OBJECT(keyvalue[, keyvalue]...)|NULL}
keyvalue:
{
  'prediction_explainer', {'permutation_importance'|'shap'|NULL}
}
```

Required ML_EXPLAIN_ROW Parameters

Set the following required parameters:

- `input_data`: Define the data to generate explanations for. The column names must match the feature column names in the table used to train the model. You can define the input data in the following ways:

  Specify a single row of data in `JSON` format:

```
mysql> SELECT sys.ML_EXPLAIN_ROW(JSON_OBJECT("column_name", value, "column_name", value, ...)', model
```

To run `ML_EXPLAIN_ROW` on multiple rows of data, specify the columns in `JSON` key-value format and select from an input table:

```
mysql> SELECT sys.ML_EXPLAIN_ROW(JSON_OBJECT("output_col_name", schema.`input_col_name`, "output_col_
       model_handle, options) FROM input_table_name LIMIT N;
```

- `model_handle`: Specifies the model handle or a session variable containing the model handle. See Work with Model Handles.

ML_EXPLAIN_ROW Options

You can set the following option in JSON format as needed:

- `prediction_explainer`: The name of the prediction explainer that you have trained for this model using `ML_EXPLAIN`.
  - `permutation_importance`: The default prediction explainer.
  - `shap`: The SHAP prediction explainer, which produces feature importance values based on Shapley values.

Syntax Examples

- The following example generates a prediction explainer on a single row of data with the default Permutation Importance prediction explainer. The results include the `ml_results` field, which uses `JSON` format. Optionally, use `\G` to display the output in an easily readable format.

```
mysql> SET @row_input = JSON_OBJECT(
          "age", 31,
          "workclass", "Private",
          "fnlwgt", 45781,
          "education", "Masters",
          "education-num", 14,
          "marital-status", "Married-civ-spouse",
          "occupation", "Prof-specialty",
          "relationship", "Not-in-family",
          "race", "White",
          "sex", "Female",
          "capital-gain", 14084,
          "capital-loss", 2042,
          "hours-per-week", 40,
          "native-country", "India");
mysql> SELECT sys.ML_EXPLAIN_ROW(@row_input, @census_model, JSON_OBJECT('prediction_explainer', 'perm
*************************** 1. row ***************************
sys.ML_EXPLAIN_ROW(@row_input, @census_model,
          JSON_OBJECT('prediction_explainer', 'permutation_importance')):
          {
                "age": 31,
                "sex": "Female",
                "race": "White",
                "Notes": "capital-gain (14084) had the largest impact towards predicting >50K",
                "fnlwgt": 45781,
                "education": "Masters",
                "workclass": "Private",
                "Prediction": ">50K",
                "ml_results": {
                   "notes": "capital-gain (14084) had the largest impact towards predicting >50K",
                   "predictions": {
                       "revenue": ">50K"
                    },
                    "attributions": {
                        "age": 0.34,
                        "sex": 0,
                        "race": 0,
                        "fnlwgt": 0,
                        "education": 0,
```

```
                        "workclass": 0,
                        "occupation": 0,
                        "capital-gain": 0.97,
                        "capital-loss": 0,
                        "relationship": 0,
                        "education-num": 0.04,
                        "hours-per-week": 0,
                        "marital-status": 0
                    }
                },
                "occupation": "Prof-specialty",
                "capital-gain": 14084,
                "capital-loss": 2042,
                "relationship": "Not-in-family",
                "education-num": 14,
                "hours-per-week": 40,
                "marital-status": "Married-civ-spouse",
                "native-country": "India",
                "age_attribution": 0.34,
                "sex_attribution": 0,
                "race_attribution": 0,
                "fnlwgt_attribution": 0,
                "education_attribution": 0,
                "workclass_attribution": 0,
                "occupation_attribution": 0,
                "capital-gain_attribution": 0.97,
                "capital-loss_attribution": 0,
                "relationship_attribution": 0,
                "education-num_attribution": 0.04,
                "hours-per-week_attribution": 0,
                "marital-status_attribution": 0
            }
1 row in set (6.3072 sec)
```

- The following example generates prediction explainers on two rows of the input table with the SHAP prediction explainer.

```
mysql> SELECT sys.ML_EXPLAIN_ROW(JSON_OBJECT(
 "age", census_train.`age`,
 "workclass", census_train.`workclass`,
 "fnlwgt", census_train.`fnlwgt`,
 "education", census_train.`education`,
 "education-num", census_train.`education-num`,
 "marital-status", census_train.`marital-status`,
 "occupation", census_train.`occupation`,
 "relationship", census_train.`relationship`,
 "race", census_train.`race`,
 "sex", census_train.`sex`,
 "capital-gain", census_train.`capital-gain`,
 "capital-loss", census_train.`capital-loss`,
 "hours-per-week", census_train.`hours-per-week`,
 "native-country", census_train.`native-country`),
 @census_model, JSON_OBJECT('prediction_explainer', 'shap'))FROM census_data.census_train LIMIT 2\G
*************************** 1. row ***************************
sys.ML_EXPLAIN_ROW(JSON_OBJECT( "age", census_train.`age`, "workclass", census_train.`workclass`, "fnlwgt
    "age": 22,
    "sex": "Female",
    "race": "Black",
    "fnlwgt": 310380,
    "education": "HS-grad",
    "workclass": "Private",
    "Prediction": "<=50K",
    "ml_results": {
        "predictions": {
            "revenue": "<=50K"
        },
        "attributions": {
            "age_attribution": 0.055990096751945995,
            "sex_attribution": 0.011676016319165776,
            "race_attribution": 0.005258734090653583,
            "fnlwgt_attribution": 0,
            "education_attribution": 0,
```

```
            "workclass_attribution": 0,
            "occupation_attribution": 0.0036531218497025536,
            "capital-gain_attribution": 0.017052572967215754,
            "capital-loss_attribution": 0,
            "relationship_attribution": 0.03019321048408115,
            "education-num_attribution": 0.01749651048882997,
            "hours-per-week_attribution": 0.003671861337781857,
            "marital-status_attribution": 0.03869036669327783
        }
    },
    "occupation": "Adm-clerical",
    "capital-gain": 0,
    "capital-loss": 0,
    "relationship": "Unmarried",
    "education-num": 9,
    "hours-per-week": 40,
    "marital-status": "Never-married",
    "native-country": "United-States",
    "age_attribution": 0.0559900968,
    "sex_attribution": 0.0116760163,
    "race_attribution": 0.0052587341,
    "fnlwgt_attribution": 0,
    "education_attribution": 0,
    "workclass_attribution": 0,
    "occupation_attribution": 0.0036531218,
    "capital-gain_attribution": 0.017052573,
    "capital-loss_attribution": 0,
    "relationship_attribution": 0.0301932105,
    "education-num_attribution": 0.0174965105,
    "hours-per-week_attribution": 0.0036718613,
    "marital-status_attribution": 0.0386903667
}
*************************** 2. row ***************************
sys.ML_EXPLAIN_ROW(JSON_OBJECT( "age", census_train.`age`, "workclass", census_train.`workclass`, "fn
    "age": 45,
    "sex": "Male",
    "race": "White",
    "fnlwgt": 182100,
    "education": "Bachelors",
    "workclass": "Local-gov",
    "Prediction": ">50K",
    "ml_results": {
        "predictions": {
            "revenue": ">50K"
        },
        "attributions": {
            "age_attribution": 0.10591945090998228,
            "sex_attribution": 0.013172526260700925,
            "race_attribution": 0.007606345008707882,
            "fnlwgt_attribution": 0.018097167152459265,
            "education_attribution": -0.007944704365873384,
            "workclass_attribution": 0.01615429281764716,
            "occupation_attribution": 0.08573874801531925,
            "capital-gain_attribution": -0.003364275424074914,
            "capital-loss_attribution": 0,
            "relationship_attribution": 0.099373669980131,
            "education-num_attribution": 0.1380689603088001,
            "hours-per-week_attribution": 0.0124334565747376,
            "marital-status_attribution": 0.0938256104928338
        }
    },
    "occupation": "Sales",
    "capital-gain": 0,
    "capital-loss": 0,
    "relationship": "Husband",
    "education-num": 13,
    "hours-per-week": 40,
    "marital-status": "Married-civ-spouse",
    "native-country": "United-States",
    "age_attribution": 0.1059194509,
    "sex_attribution": 0.0131725263,
    "race_attribution": 0.007606345,
```

```
    "fnlwgt_attribution": 0.0180971672,
    "education_attribution": -0.0079447044,
    "workclass_attribution": 0.0161542928,
    "occupation_attribution": 0.085738748,
    "capital-gain_attribution": -0.0033642754,
    "capital-loss_attribution": 0,
    "relationship_attribution": 0.09937367,
    "education-num_attribution": 0.1380689603,
    "hours-per-week_attribution": 0.0124334566,
    "marital-status_attribution": 0.0938256105
}
2 rows in set (5.5382 sec)
```

See Also

- Generate Prediction Explanations for a Row of Data

10.2.8 ML_EXPLAIN_TABLE

`ML_EXPLAIN_TABLE` explains predictions for an entire table of unlabeled data. It limits explanations to the 100 most relevant features.

ML_EXPLAIN_TABLE Overview

Note

`ML_EXPLAIN_TABLE` is a very memory-intensive process. Depending on your MySQL version, we recommend the following:

- Before MySQL 9.4.1, use the `batch_size` option to limit operations to batches of 10 to 100 rows by splitting large tables into smaller tables. For tables with over ten columns, we suggest a `batch_size` value of `10`.
- As of MySQL 9.4.1, the `batch_size` option is deprecated. Limit the input table to a maximum of 100 rows. If the input table has more than ten columns, limit it to ten rows.

A call to `ML_EXPLAIN_TABLE` can include columns that were not present during `ML_TRAIN`. A table can include extra columns, and still use the MySQL HeatWave AutoML model. This allows side by side comparisons of target column labels, ground truth, and explanations in the same table. `ML_EXPLAIN_TABLE` ignores any extra columns, and appends them to the results.

A loaded model and trained with the appropriate prediction explainer is required to run `ML_EXPLAIN_TABLE`. See Generate Prediction Explanations for a Table.

The output table includes a primary key:

- If the input table has a primary key, the output table will have the same primary key.
- If the input table does not have a primary key, the output table will have a new primary key column that auto increments.
  - As of MySQL 8.4.1, the name of the new primary key column is `_4aad19ca6e_pk_id`. The input table must not have a column with the name `_4aad19ca6e_pk_id` that is not a primary key.
  - Before MySQL 8.4.1, the name of the new primary key column is `_id`. The input table must not have a column with the name `_id` that is not a primary key.

As of MySQL 9.4.1, you have the option to specify the input table and output table as the same table if specific conditions are met. See Input Tables and Output Tables to learn more.

`ML_EXPLAIN_TABLE` does not support recommendation, anomaly detection, and topic modeling models. A call with one of these models produces an error.

`ML_EXPLAIN_TABLE` does not support the following model types:

- Forecasting
- Recommendation
- Anomaly detection
- Anomaly detection for logs
- Topic modeling

ML_EXPLAIN_TABLE Syntax

```
mysql> CALL sys.ML_EXPLAIN_TABLE(table_name, model_handle, output_table_name, options);

options: {JSON_OBJECT(keyvalue[, keyvalue]...)|NULL}
keyvalue:
{
  'prediction_explainer', {'permutation_importance'|'shap'|NULL}
  |'batch_size', 'N'
}
```

Required ML_EXPLAIN_TABLE Parameters

Set the following required parameters.

- `table_name`: Specifies the fully qualified name of the input table (*`database_name.table_name`*). The input table should contain the same feature columns as the table used to train the model. If the target column is included in the input table, it is not considered when generating prediction explanations.
- `model_handle`: Specifies the model handle or a session variable containing the model handle. See Work with Model Handles.
- `output_table_name`: Specifies the table where explanation data is stored. A fully qualified table name must be specified (*`database_name.table_name`*). As of MySQL 9.4.1, you have the option to specify the input table and output table as the same table if specific conditions are met. See Input Tables and Output Tables to learn more.

ML_EXPLAIN_TABLE Options

Set the following options as needed.

- `prediction_explainer`: The name of the prediction explainer that you have trained for this model using `ML_EXPLAIN`.
  - `permutation_importance`: The default prediction explainer.
  - `shap`: The SHAP prediction explainer, which produces feature importance values based on Shapley values.
- `batch_size`: Deprecated as of MySQL 9.4.1. The size of each batch. You can set a value between 1 and 100. For tables with over ten columns, we recommend a value of `10`. If on MySQL 9.4.1 and later, manually limit the size of the input table to 100 rows. If the input table has over ten columns, limit it to ten rows.

Syntax Examples

- The following example generates explanations for a table of data with the default Permutation Importance prediction explainer. The `ML_EXPLAIN_TABLE` call specifies the fully qualified name of the table to generate explanations for, the session variable containing the model handle, and the fully qualified output table name.

```
mysql> CALL sys.ML_EXPLAIN_TABLE('census_data.census_train', @census_model, 'census_data.census_train
                                  JSON_OBJECT('prediction_explainer', 'permutation_importance'));
```

To view ML_EXPLAIN_TABLE results, query the output table. The SELECT statement retrieves explanation data from the output table. The table includes the primary key, _4aad19ca6e_pk_id, and the ml_results column, which uses JSON format:

```
mysql> SELECT * FROM census_train_permutation LIMIT 3;
+-------------------+-----+-----------+--------+------------+--------------+-------------------+-------
| _4aad19ca6e_pk_id | age | workclass | fnlwgt | education  | education-num | marital-status    | occupa
+-------------------+-----+-----------+--------+------------+--------------+-------------------+-------
|                 1 |  37 | Private   |  99146 | Bachelors  |           13 | Married-civ-spouse | Exec-m
|                 2 |  34 | Private   |  27409 | 9th        |            5 | Married-civ-spouse | Craft-
|                 3 |  30 | Private   | 299507 | Assoc-acdm |           12 | Separated          | Other-
+-------------------+-----+-----------+--------+------------+--------------+-------------------+-------
```

See Also

- Generate Predictions Explanations for a Table

10.2.9 ML_SCORE

ML_SCORE scores a model by generating predictions using the feature columns in a labeled dataset as input and comparing the predictions to ground truth values in the target column of the labeled dataset. The dataset used with ML_SCORE should have the same feature columns as the dataset used to train the model but the data should be different. For example, you might reserve 20 to 30 percent of the labeled training data for scoring.

ML_SCORE returns a computed metric indicating the quality of the model.

ML_SCORE does not support the topic modeling model type.

ML_SCORE Syntax

```
mysql> CALL sys.ML_SCORE(table_name, target_column_name, model_handle, metric, score, options);

options: {JSON_OBJECT(keyvalue[, keyvalue]...)|NULL}
keyvalue:
{
  'threshold', 'N'
  |'topk', 'N'
  |'remove_seen', {'true'|'false'}
  |'item_metadata', JSON_OBJECT('table_name'['database_name.table_name'] ...)
  |'user_metadata', JSON_OBJECT('table_name'['database_name.table_name'] ...)
}
```

Required ML_SCORE Parameters

Set the following required parameters.

- table_name: Specifies the fully qualified name of the table used to compute model quality (*database_name.table_name*). The table must contain the same columns as the training dataset.
- target_column_name: If scoring a supervised or semi-supervised model, specify the name of the target column containing ground truth values. If scoring an unsupervised model, set to NULL. See MySQL HeatWave AutoML Learning Types.
- model_handle: Specifies the model handle or a session variable containing the model handle. See Work with Model Handles.
- metric: Specifies the name of the metric. The metric selected must be compatible with the task type used for training the model. See Optimization and Scoring Metrics.
- score: Specifies the user-defined variable name for the computed score. The ML_SCORE routine populates the variable. User variables are written as *@var_name*. Any valid name for a user-defined variable is permitted.

The following options in JSON format are available for recommendation and anomaly detection models.

Options for Recommendation Models

Set the following options as needed for recommendation models.

- `threshold`: The optional threshold that defines positive feedback, and a relevant sample. Only use with ranking metrics. It can be used for either explicit or implicit feedback.
- `topk`: The optional top number of recommendations to provide. The default is `3`. Set a positive integer between 1 and the number of rows in the table.

  A `recommendation` task and ranking metrics can use both `threshold` and `topk`.
- `remove_seen`: If the input table overlaps with the training table, and `remove_seen` is `true`, then the model will not repeat existing interactions. The default is `true`. Set `remove_seen` to `false` to repeat existing interactions from the training table.
- `item_metadata`: As of MySQL 9.5.0, it defines the table that has item descriptions. It is a JSON object that has the `table_name` option as a key, which specifies the table that has item descriptions. One column must be the same as the `item_id` in the input table.
- `user_metadata`: As of MySQL 9.5.0, it defines the table that has user descriptions. It is a JSON object that has the `table_name` option as a key, which specifies the table that has user descriptions. One column must be the same as the `user_id` in the input table.
  - `table_name`: To be used with the `item_metadata` and `user_metadata` options. It specifies the table name that has item or user descriptions. It must be a string in a fully qualified format (schema_name.table_name) that specifies the table name.

Requirements and Options for Anomaly Detection Models

As of MySQL 9.4.1, if you run `ML_PREDICT_TABLE` with the `log_anomaly_detection` task, at least one column must act as the primary key to establish the temporal order of logs.

Set the following options as needed for anomaly detection models.

- `threshold`: The threshold you set on anomaly detection models determines which rows in the output table are labeled as anomalies with an anomaly score of `1`, or normal with an anomaly score of `0`. The value for the threshold is the degree to which a row of data or log segment is considered for anomaly detection. Any sample with an anomaly score above the threshold is classified an anomaly. The default value is (1 - `contamination`)-th percentile of all the anomaly scores.
- `topk`: The optional top K rows to display with the highest anomaly scores. Set a positive integer between 1 and the number of rows in the table. If `topk` is not set, `ML_SCORE` uses `threshold`.

  Do not set both `threshold` and `topk`. Use `threshold` or `topk`, or set *`options`* to `NULL`.

Syntax Examples

- The following example runs generates a score by using the `balanced_accuracy` metric. Query the score with the session variable for the `ML_SCORE` routine.

```
mysql> CALL sys.ML_SCORE('census_data.census_train', 'revenue', 'census_data.census_train_admin_17454
                         'balanced_accuracy', @score, NULL);
Query OK, 0 rows affected (3.0536 sec)
mysql> SELECT @score;
+--------------------+
| @score             |
+--------------------+
| 0.8151071071624756 |
+--------------------+
1 row in set (0.0411 sec)
```

- The following example uses the `accuracy` metric with a `threshold` set to 90%.

```
mysql> CALL sys.ML_SCORE('mlcorpus_anomaly_detection.volcanoes-b3_anomaly_train', 'target', @anomaly,
```

```
                          'accuracy', @score, JSON_OBJECT('threshold', 0.9));
Query OK, 0 rows affected (1.86 sec)

mysql> SELECT @score;
+--------------------+
| @score             |
+--------------------+
| 0.9791129231452942 |
+--------------------+
1 row in set (0.00 sec)
```

- The following example uses the `precision_at_k` metric with a `topk` value of 10.

```
mysql> CALL sys.ML_SCORE('mlcorpus_anomaly_detection.volcanoes-b3_anomaly_train', 'target', @anomaly,
                          'precision_at_k', @score, JSON_OBJECT('topk', 10));
Query OK, 0 rows affected (5.84 sec)

mysql> SELECT @score;
+---------------------+
| @score              |
+---------------------+
| 0.30000001192092896 |
+---------------------+
1 row in set (0.0443 sec)
```

- The following example overrides the `ensemble_score` value from the `ML_TRAIN` routine to a new value of 0.5.

```
mysql> CALL sys.ML_SCORE('mlcorpus.anomaly_train_with_target', "target", @semsup_gknn,
                          'precision_at_k', @semsup_score_gknn_weighted,
                          CAST('{"topk": 10, "experimental": {"semisupervised": {"supervised_submodel_wei
Query OK, 0 rows affected (1.7685 sec)
mysql> SELECT @score;
+---------------------+
| @score              |
+---------------------+
| 0.98844456000887996 |
+---------------------+
1 row in set (0.0443 sec)
```

See Also

- Score a Model

10.2.10 ML_MODEL_LOAD

The `ML_MODEL_LOAD` routine loads a model from the model catalog. A model remains loaded until the model is unloaded using the `ML_MODEL_UNLOAD` routine or until the MySQL HeatWave Cluster restarts.

MySQL 9.0.0 introduces support for large models that changes how MySQL HeatWave AutoML stores models. See Section 6.9.1, “The Model Catalog”. `ML_MODEL_LOAD` upgrades older models.

Use `ML_MODEL_ACTIVE` to check which models are active for which users. All active users and models share the amount of memory defined by the shape, and it might be necessary to schedule users. `ML_MODEL_LOAD` generates an error if there are memory limitations.

ML_MODEL_LOAD Syntax

```
mysql> CALL sys.ML_MODEL_LOAD(model_handle, user);
```

ML_MODEL_LOAD Parameters

Set the following parameters.

- `model_handle`: Specifies the model handle or a session variable containing the model handle. To look up a model handle, see Query the Model Handle.

- `user`: The MySQL user name of the model owner. You can set this to `NULL`. To learn how to share models with other users, see Share a Model.

Syntax Examples

- An example that specifies the model handle and sets the `user` parameter to `NULL`.

```
mysql> CALL sys.ML_MODEL_LOAD('ml_data.iris_train_user1_1636729526', NULL);
```

- An example that specifies a session variable containing the model handle.

```
mysql> CALL sys.ML_MODEL_LOAD(@iris_model, NULL);
```

- An example that specifies the model handle and the model owner.

```
mysql> CALL sys.ML_MODEL_LOAD('ml_data.iris_train_user1_1636729526', user1);
```

See Also

- Load a Model
- Work with Model Handles

10.2.11 ML_MODEL_UNLOAD

`ML_MODEL_UNLOAD` unloads a model from MySQL HeatWave AutoML.

As of MySQL 9.0.0, `ML_MODEL_UNLOAD` does not check whether the model specified is in the model catalog. If it is not, `ML_MODEL_UNLOAD` will succeed, but will not unload any model. Use `ML_MODEL_ACTIVE` to check which models are active and owned by the user.

ML_MODEL_UNLOAD Syntax

```
mysql> CALL sys.ML_MODEL_UNLOAD(model_handle);
```

To run `ML_MODEL_UNLOAD`, define the `model_handle`. To look up a model handle, see Query the Model Handle.

Syntax Examples

- An example that specifies the model handle.

```
mysql> CALL sys.ML_MODEL_UNLOAD('ml_data.iris_train_user1_1636729526');
```

- An example that specifies a session variable containing the model handle.

```
mysql> CALL sys.ML_MODEL_UNLOAD(@iris_model);
```

See Also

- Unload a Model

10.2.12 ML_MODEL_ACTIVE

MySQL 9.0.0 introduces the `ML_MODEL_ACTIVE` routine. Use this routine to check which models are loaded and active for which users. All active users and models share the amount of memory defined by the shape, and it might be necessary to schedule users.

ML_MODEL_ACTIVE Syntax

```
mysql> CALL sys.ML_MODEL_ACTIVE (user, model_info);
```

`ML_MODEL_ACTIVE` parameters:

- `user`: The user to provide information for. Set to `current` or `all` or `NULL`. `NULL` is equivalent to `current`.

- `model_info`: The name of the JSON array session variable that contains the active user and model information. There are two JSON object literals.

  If `user` is set to `current` or `NULL`, the following information displays.

  - A JSON object literal that displays:
    - Key: The total model size (bytes).
    - Value: The sum of model sizes for the current user.
  - A second JSON object literal that displays:
    - Key: The model handle for a loaded and active model owned by the current user.
    - Value: The `model_metadata` for the model.

  If `user` is set to `all`, the following information displays.

  - A JSON object literal that displays:
    - Key: The total model size (bytes).
    - Value: The sum of model sizes for all users.
  - A second JSON object literal that displays:
    - Key: The name of a user who has loaded and active models.
    - Value: A list of JSON object literals of the model handle and brief model metadata for each loaded and active model.

Syntax Examples

- *`user1`* checks their own models:

```
mysql> CALL sys.ML_MODEL_ACTIVE('current', @model_info);
Query OK, 0 rows affected (0.10 sec)

mysql> SELECT JSON_PRETTY(@model_info);
+------------------------------------------------------------+
| JSON_PRETTY(@model_info)                                   |
+------------------------------------------------------------+
| [                                                          |
{                                                            |
"total model size(bytes)": 348954                          |
},                                                           |
{                                                            |
"iris_export_user1": {                                     |
  "task": "classification",                                |
  "notes": "",                                             |
  "chunks": 1,                                             |
  "format": "HWMLv2.0",                                    |
  "n_rows": 120,                                           |
  "status": "Ready",                                       |
  "options": {                                             |
    "model_explainer": "permutation_importance, shap",     |
    "prediction_explainer": "shap"                         |
  },                                                       |
  "n_columns": 4,                                          |
  "pos_class": null,                                       |
  "column_names": [                                        |
    "sepal length",                                        |
    "sepal width",                                         |
    "petal length",                                        |
    "petal width"                                          |
  ],                                                       |
```

```
  "contamination": null,                                    |
  "model_quality": "high",                                  |
  "training_time": 18.363686,                               |
  "algorithm_name": "ExtraTreesClassifier",                 |
  "training_score": -0.10970368035588404,                   |
  "build_timestamp": 1697524180,                            |
  "n_selected_rows": 96,                                    |
  "training_params": {                                      |
    "sp_arr": null,                                         |
    "timezone": null,                                       |
    "recommend": "ratings",                                 |
    "force_use_X": false,                                   |
    "recommend_k": 3,                                       |
    "remove_seen": true,                                    |
    "contamination": null,                                  |
    "feedback_threshold": 1                                 |
  },                                                        |
  "train_table_name": "mlcorpus.iris_train",                |
  "model_explanation": {                                    |
    "shap": {                                               |
      "petal width": 0.3139,                                |
      "sepal width": 0.0296,                                |
      "petal length": 0.2787,                               |
      "sepal length": 0.0462                                |
    },                                                      |
    "permutation_importance": {                             |
      "petal width": 0.2301,                                |
      "sepal width": 0.0056,                                |
      "petal length": 0.2192,                               |
      "sepal length": 0.0056                                |
    }                                                       |
  },                                                        |
  "model_object_size": 348954,                              |
  "n_selected_columns": 4,                                  |
  "target_column_name": "class",                            |
  "optimization_metric": "neg_log_loss",                    |
  "selected_column_names": [                                |
    "petal length",                                         |
    "petal width",                                          |
    "sepal length",                                         |
    "sepal width"                                           |
  ]                                                         |
}                                                           |
}                                                             |
] |                                                             |
+-----------------------------------------------------------+
1 row in set (0.00 sec)
```

- *user1* checks their own models, and extracts specific information:

```
mysql> CALL sys.ML_MODEL_ACTIVE('current', @model_info);
Query OK, 0 rows affected (0.12 sec)

mysql> SELECT JSON_KEYS(JSON_EXTRACT(@model_info, '$[1]'));
+-----------------------------------------------------------+
| JSON_KEYS(JSON_EXTRACT(@model_info, '$[1]'))              |
+-----------------------------------------------------------+
| ["iris_export", "mlcorpus.iris_train_user1_1697524152037"] |
+-----------------------------------------------------------+
1 row in set (0.00 sec)

mysql> SELECT JSON_EXTRACT(@model_info, '$[0]');
+-------------------------------------+
| JSON_EXTRACT(@model_info, '$[0]')   |
+-------------------------------------+
| {"total model size(bytes)":  697908} |
+-------------------------------------+
1 row in set (0.01 sec)
```

- *user1* checks the models for all users:

```
mysql> CALL sys.ML_MODEL_ACTIVE('all', @model_info);
```

```
Query OK, 0 rows affected (0.11 sec)

mysql> SELECT JSON_PRETTY(@model_info);
+-----------------------------------------------------+
| JSON_PRETTY(@model_info)                            |
+-----------------------------------------------------+
| [                                                   |
{                                                     |
"total model size(bytes)": 1046862                    |
},                                                    |
{                                                     |
"user2": [                                            |
  {                                                   |
    "iris_export_user2": {                            |
      "format": "HWMLv2.0",                           |
      "model_size(byte)": 348954                      |
    }                                                 |
  }                                                   |
],                                                    |
"user1": [                                            |
  {                                                   |
    "mlcorpus.iris_train_user1_1697524152037": {      |
      "format": "HWMLv2.0",                           |
      "model_size(byte)": 348954                      |
    }                                                 |
  },                                                  |
  {                                                   |
    "iris_export": {                                  |
      "format": "HWMLv2.0",                           |
      "model_size(byte)": 348954                      |
    }                                                 |
  }                                                   |
]                                                     |
}                                                       |
] |                                                     |
+-----------------------------------------------------+
1 row in set (0.00 sec)
```

10.2.13 NL2ML

MySQL 9.4.0 introduces the `NL2ML` (natural language to machine learning) routine, which provides step-by-step guidelines on using MySQL HeatWave AutoML for a particular business problem. Use this routine to ask general questions about MySQL HeatWave AutoML and to generate AutoML queries you can copy and run.

Note

- Before MySQL 9.6.0, `NL2ML` requires Oracle Cloud Infrastructure Generative AI, which is available in specific regions. To confirm if your region has Generative AI, see Regions with Generative AI.

  As of MySQL 9.6.0, if your region does not have OCI Generative AI, or you want to bypass OCI Generative AI and instead use in-database LLMs to generate responses, you can set the `skip_generate` option to `true`. See Use NL2ML with In-Database LLMs.

- NL2ML uses pretrained foundational models supported by OCI Generative AI.

  - Before MySQL 9.6.0, MySQL HeatWave in-database LLMs are not supported. To review available OCI Generative AI models, see Pretrained Foundational Models in Generative AI. For every major MySQL release, NL2ML creates a new vector store from the MySQL HeatWave User Guide documentation. After creating each new vector store, the old vector store gets deleted. Due to the creation of vector stores, some NL2ML calls may take longer to complete.

- As of MySQL 9.6.0, the `skip_generate` option allows you to use in-database LLMs instead of foundational models supported by OCI Generative AI. See Use NL2ML with In-Database LLMs.

NL2ML Syntax

```
mysql> CALL sys.NL2ML (query, response);
```

`NL2ML` parameters:

- `query`: Enter a question in natural language related to MySQL HeatWave AutoML. For example, "What are the different types of machine learning models I can create?".
- `response`: The name of the JSON object session variable that contains the response to the question.

The nl2ml_options Session Variable

The `@nl2ml_options` session variable allows you to do the following:

- Set the `skip_generate` option to `true` or `false`. The default value is `false`. You must set this to `true` to bypass OCI Generative AI and only provide citations. This also allows you to leverage the citations as context to generate model training guidance and SQL queries with in-database LLMs. To learn more, see Use NL2ML with In-Database LLMs.
- View the chat history of the last 20 questions and current OCI Generative AI model:

  ```
  mysql> SELECT JSON_PRETTY(@nl2ml_options);
  ```

- Reset the chat history. Use one of the following commands:

  ```
  mysql> SET @nl2ml_options = NULL;

  mysql> SET @nl2ml_options = JSON_REMOVE(@nl2ml_options, '$.chat_history');
  ```

- Change the current OCI Generative AI model. To review available options for `model_id`, see Pretrained Foundational Models in Generative AI.

  Change the current model and reset the chat history:

  ```
  mysql> SET @nl2ml_options = JSON_OBJECT("model_id", "meta.llama-3.1-405b-instruct");
  ```

  Change the current model and maintain the current chat history:

  ```
  mysql> SET @nl2ml_options = JSON_SET(@nl2ml_options, '$.model_id', "meta.llama-3.1-405b-instruct");
  ```

Syntax Examples

- Ask a question about using MySQL HeatWave AutoML:

  ```
  mysql> CALL sys.NL2ML("Is there any limit on the size of training table?", @output);
  Query OK, 0 rows affected (11.0086 sec)

  mysql> SELECT @output;
  +-----------------------------------------------------------------------------------------------------------
  | @output |
  +-----------------------------------------------------------------------------------------------------------
  | {"text": "Yes, there are limits on the size of the training table for HeatWave AutoML models. The t
  +-----------------------------------------------------------------------------------------------------------
  1 row in set (0.0003 sec)
  ```

- Ask a question, and then ask a follow-up question based on the first one:

  ```
  mysql> CALL sys.NL2ML("What is the purpose of task in ML_TRAIN?", @output);
  Query OK, 0 rows affected (32.4279 sec)
  ```

```
mysql> SELECT @output;
+-----------------------------------------------------------------------------------------------------------------------
| @output |
+-----------------------------------------------------------------------------------------------------------------------
| {"text": "The purpose of 'task' in ML_TRAIN is to specify the machine learning task, such as classifica
+-----------------------------------------------------------------------------------------------------------------------
1 row in set (0.0003 sec)
mysql> CALL sys.NL2ML("Can you give me an example of using that?", @output);
Query OK, 0 rows affected (11.7221 sec)

mysql> SELECT @output;
+-----------------------------------------------------------------------------------------------------------------------
| @output |
+-----------------------------------------------------------------------------------------------------------------------
| {"text": "Here's an example of specifying a machine learning task in ML_TRAIN:
          \n\n```sql\nCALL sys.ML_TRAIN('mlcorpus.emails', 'spam_label', JSON_OBJECT('task', 'classifica
          \n```\n\nIn this example, the machine learning task is specified as 'classification' and the t
+-----------------------------------------------------------------------------------------------------------------------
1 row in set (0.0003 sec)
```

- Ask for an appropriate table in the DB System to use for a training query:

```
mysql> CALL sys.NL2ML("Can you give me a training query for email spam detection?", @output);
Query OK, 0 rows affected (8.5084 sec)

mysql> SELECT @output;
+-----------------------------------------------------------------------------------------------------------------------
| @output |
+-----------------------------------------------------------------------------------------------------------------------
| {"text": "```sql\nCALL sys.ML_TRAIN('mlcorpus.emails', 'spam_label', \n JSON_OBJECT('task', 'classifica
+-----------------------------------------------------------------------------------------------------------------------
1 row in set (0.0003 sec)
```

- Request debugging assistance for an error message:

```
mysql> CALL sys.NL2ML("When I tried to train a model I got this error: can not train a regression or reco
Query OK, 0 rows affected (18.0766 sec)

mysql> SELECT @output;
+-----------------------------------------------------------------------------------------------------------------------
| @output
+-----------------------------------------------------------------------------------------------------------------------
| {"text": "The error \"can not train a regression or recommendation task with non-numeric target column\
           \n\nRegression models require a numeric target column, as they are designed to predict contin
           \n\nTo resolve this issue, you need to either:\n\n1. Choose a different machine learning task
           For example, if your target column is a categorical variable with a limited number of categor
           \n\nIn your case, since you are using the \"food_delivery\" table, you may want to check the
+-----------------------------------------------------------------------------------------------------------------------
1 row in set (0.0003 sec)
```

- Retrieve the chat history of questions and responses, along with the current OCI Generative AI model, with the `@nl2ml_options` session variable:

```
mysql> SELECT JSON_PRETTY(@nl2ml_options);
+-----------------------------------------------------------------------------------------------------------------------
| JSON_PRETTY(@nl2ml_options) |
+-----------------------------------------------------------------------------------------------------------------------
| {
"model_id": "meta.llama-3.1-405b-instruct",
"chat_history": [
{
"user_message": "Is there any limit on the size of training table?",
"chat_bot_message": "Yes, there is a size limit for training tables in HeatWave. The table used to train
}
]
} |
+-----------------------------------------------------------------------------------------------------------------------
1 row in set (0.0003 sec)
```

- Use the `skip_generate` option to bypass OCI Generative AI content generation and only provide citations to relevant documentation. Then, retrieve a relevant table schema related to the

question (ML_RETRIEVE_SCHEMA_METADATA), build a compact context string from the citations (GROUP_CONCAT), and generate a response that includes the citations, context, and retrieved table schema (ML_GENERATE).

```
mysql> SET @input = "How can I train a model to predict net worth of a singer?";

mysql> SET @nl2ml_options = JSON_OBJECT("skip_generate", true);

mysql> CALL sys.NL2ML(@input, @out);

mysql> SELECT JSON_PRETTY(@out);
JSON_PRETTY(@out)
{
  "citations": [
    {
      "segment": "<segment content>",
      "distance": 0.1023,
      "document_name": "https://.../heatwave-en-ml-9.6.0.pdf",
      "segment_number": <segment number>
    },
    ...
  ],
  "vector_store": [
    "`ML_SCHEMA_userOne`.`hwml_vector_store_heatwave_en_ml_9_6_0`"
  ],
  "retrieval_info": {
    "method": "n_citations",
    "threshold": 0.114
  }
}

mysql> CALL sys.ML_RETRIEVE_SCHEMA_METADATA(@input, @retrieved, NULL);

mysql> SELECT @retrieved;
@retrieved
CREATE TABLE `mlcorpus`.`singer`(
  `Singer_ID` int,
  `Name` varchar,
  `Birth_Year` double,
  `Net_Worth_Millions` double,
  `Citizenship` varchar
);

mysql> SELECT GROUP_CONCAT(seg SEPARATOR '\n') INTO @ctx
FROM JSON_TABLE(JSON_EXTRACT(@out,'$.citations'),
'$[*]' COLUMNS (seg LONGTEXT PATH '$.segment')) AS jt;

mysql> SET @final_ctx = CONCAT(@ctx, '\n\nRetrieved tables:\n', @retrieved);

mysql> SELECT sys.ML_GENERATE(
@input,
JSON_OBJECT(
"task", "generation",
"model_id", "llama3.2-3b-instruct-v1",
"context", @final_ctx
)
) INTO @result;

mysql> SELECT JSON_UNQUOTE(JSON_EXTRACT(@result,'$.text')) AS generated_sql;
generated_sql
To train a model to predict the net worth of a singer, you can use the ML_TRAIN routine. First, prepa
which in this case seems to be the 'singer' table in the 'mlcorpus' schema. Ensure that the table has
such as 'Singer_ID', 'Name', 'Birth_Year', 'Net_Worth_Millions', and 'Citizenship'.

The 'Net_Worth_Millions' column will be your target column for prediction. You may need to preprocess
for example, converting categorical variables like 'Name' and 'Citizenship' into numerical variables

Then, you can call the ML_TRAIN routine with the appropriate options. For a regression task like pred
you would specify the task as 'regression' in the JSON options. Here's a simplified example:

```sql
CALL sys.ML_TRAIN('mlcorpus.singer',
```
```

```
                  @model_handle,
                  'Net_Worth_Millions',
                  JSON_OBJECT('task', 'regression',
                              'algorithm', 'XGBRegressor'));
```

```
Replace '@model_handle' with your actual model handle variable. This will train a model to predict the 'N
based on the other columns in your 'singer' table. After training, you can use the ML_PREDICT_ROW or ML_P
to generate predictions for new, unseen data.
```

See Also

- Learn About MySQL HeatWave AutoML with Oracle Cloud Infrastructure Generative AI

10.2.14 TRAIN_TEST_SPLIT

MySQL 9.4.1 introduces the `TRAIN_TEST_SPLIT` routine, which automatically splits your data into training and testing datasets.

Two new tables in the same database are created with the following names:

- `[original_table_name]_train`
- `[original_table_name]_test`

The split of the data between training and testing datasets depends on the machine learning task.

- Classification: A stratified split of data. For each class in the dataset, 80% of the samples goes into the training dataset, and the remaining goes into the testing dataset. If the number of samples in the 80% subset is fewer than 5, then instead select 5 of the samples for the training dataset.
- Regression: A random split of data.
- Forecasting: A time-based split of data. Order the table by the `datetime_index` values and select the first 80% of the samples for the training dataset. Insert the subsequent samples into the testing dataset.
- Unsupervised anomaly detection: A random split of data. Select 80% of the samples for the training dataset, and select the remaining samples for the testing dataset.
- Semi-supervised anomaly detection: A stratified split of data.
- Anomaly detection for log data: A split of data based on primary key values. The first 80% of the samples go into the training dataset. The remaining samples go into the testing dataset. Review requirements when running Anomaly Detection for Logs.
- Recommendations: A random split of data.
- Topic modeling: A random split of data.

TRAIN_TEST_SPLIT Syntax

```
mysql> CALL sys.TRAIN_TEST_SPLIT ('table_name', 'target_column_name', options);

options: {JSON_OBJECT(keyvalue[, keyvalue]...)|NULL}
keyvalue:
{
  'task', {'classification'|'regression'|'forecasting'|'anomaly_detection'|'log_anomaly_detection'|'recomme
  |'datetime_index', 'column'
  |'semisupervised', {'true'|'false'}
}
```

`TRAIN_TEST_SPLIT` parameters:

- `table_name`: You must provide the fully qualified name of the table that contains the dataset to split (*`schema_name.table_name`*).

- `target_column_name`: Classification and semi-supervised anomaly detection tasks require a target column. All other tasks do not require a target column. If a target column is not required, you can set this parameter to `NULL`.
- `options`: Set the following options as needed as key-value pairs in JSON object format. If no options are needed, set this to `NULL`.
  - `task`: If the machine learning task is not set, the default task is `classification`.
  - `datetime_index`: The column that has datetime values. This parameter is required for forecasting tasks.

    The following data types for this column are supported:

    - `DATETIME`
    - `TIMESTAMP`
    - `DATE`
    - `TIME`
    - `YEAR`
  - `semisupervised`: If running an anomaly detection task, set this to `true` for semi-supervised learning, or `false` for unsupervised learning. If this is set to `NULL`, then the default value of `false` is selected.

Syntax Examples

- A classification task:

```
mysql> CALL sys.TRAIN_TEST_SPLIT('data_files_db.data_files_1', 'class', JSON_OBJECT('task', 'classifi
mysql> SHOW TABLES;
+-------------------------+
| Tables_in_data_files_db |
+-------------------------+
| data_files_1            |
| data_files_1_test       |
| data_files_1_train      |
+-------------------------+
```

- A regression task:

```
mysql> CALL sys.TRAIN_TEST_SPLIT('data_files_db.food_delivery_data', NULL, JSON_OBJECT('task', 'regre
mysql> SHOW TABLES;
+-------------------------+
| Tables_in_data_files_db |
+-------------------------+
| food_delivery_data      |
| food_delivery_data_test |
| food_delivery_data_train |
+-------------------------+
```

- A forecasting task:

```
mysql> CALL sys.TRAIN_TEST_SPLIT('data_files_db.forecasting_data', NULL,
                                 JSON_OBJECT('task', 'forecasting',
                                             'datetime_index', 'timestamp'));
mysql> SHOW TABLES;
+-------------------------+
| Tables_in_data_files_db |
+-------------------------+
| forecasting_data        |
| forecasting_data_test   |
| forecasting_data_train  |
+-------------------------+
```

- An unsupervised anomaly detection task:

```
mysql> CALL sys.TRAIN_TEST_SPLIT('data_files_db.anomaly_detection_data', NULL,
                                  JSON_OBJECT('task', 'anomaly_detection'));
mysql> SHOW TABLES;
+------------------------------+
| Tables_in_data_files_db      |
+------------------------------+
| anomaly_detection_data       |
| anomaly_detection_data_test  |
| anomaly_detection_data_train |
+------------------------------+
```

- A semi-supervised anomaly detection task:

```
mysql> CALL sys.TRAIN_TEST_SPLIT('data_files_db.anomaly_detection_semi', 'anomaly',
                                  JSON_OBJECT('task', 'anomaly_detection',
                                              'semisupervised', 'true'));
mysql> SHOW TABLES;
+------------------------------+
| Tables_in_data_files_db      |
+------------------------------+
| anomaly_detection_semi       |
| anomaly_detection_semi_test  |
| anomaly_detection_semi_train |
+------------------------------+
```

- A task for anomaly detection on log data:

```
mysql> CALL sys.TRAIN_TEST_SPLIT('data_files_db.anomaly_detection_logs', NULL,
                                  JSON_OBJECT('task', 'log_anomaly_detection'));
mysql> SHOW TABLES;
+------------------------------+
| Tables_in_data_files_db      |
+------------------------------+
| anomaly_detection_logs       |
| anomaly_detection_logs_test  |
| anomaly_detection_logs_train |
+------------------------------+
```

- A recommendation task:

```
mysql> CALL sys.TRAIN_TEST_SPLIT('data_files_db.rec_data', NULL, JSON_OBJECT('task', 'recommendation'));
mysql> SHOW TABLES;
+-------------------------+
| Tables_in_data_files_db |
+-------------------------+
| rec_data                |
| rec_data_test           |
| rec_data_train          |
+-------------------------+
```

- A topic modeling task:

```
mysql> CALL sys.TRAIN_TEST_SPLIT('data_files_db.text_data', NULL, JSON_OBJECT('task', 'topic_modeling'));
mysql> SHOW TABLES;
+-------------------------+
| Tables_in_data_files_db |
+-------------------------+
| text_data               |
| text_data_test          |
| text_data_train         |
+-------------------------+
```

10.2.15 Model Types

MySQL HeatWave AutoML supports the following training models. When training a model, use the `ML_TRAIN` `model_list` and `exclude_model_list` options to specify the training models to consider or exclude. The Model Metadata includes the `algorithm_name` field, which defines the model type.

Classification Models

- LogisticRegression
- GaussianNB
- DecisionTreeClassifier
- RandomForestClassifier
- XGBClassifier
- LGBMClassifier
- SVC
- LinearSVC
- ExtraTreesClassifier

Regression Models

- DecisionTreeRegressor
- RandomForestRegressor
- LinearRegression
- LGBMRegressor
- XGBRegressor
- SVR
- LinearSVR
- ExtraTreesRegressor

Forecasting Models

Univariate endogenous models:

- NaiveForecaster
- ThetaForecaster
- ExpSmoothForecaster
- ETSForecaster
- `STLwESForecaster`: STLForecast with ExponentialSmoothing substructure
- `STLwARIMAForecaster`: STLForecast with ARIMA substructure

Univariate endogenous with exogenous models:

- SARIMAXForecaster
- OrbitForecaster (As of MySQL 9.1.1)

Multivariate endogenous with exogenous models:

- VARMAXForecaster

Univariate or multivariate endogenous with exogenous models:

- DynFactorForecaster

Anomaly Detection Models

- GkNN: Generalized kth Nearest Neighbors
- PCA: Principal Component Analysis
- GLOF: Generalized Local Outlier Factor

Recommendation Models

As of MySQL 9.5.0, the TwoTower model is the default model and can generate recommendations with implicit or explicit feedback. See Recommendation Training Models to learn more.

Recommendation models that rate users or items to use with explicit feedback:

- Baseline
- CoClustering
- NormalPredictor
- SlopeOne
- Matrix factorization models:
  - SVD
  - SVDpp
  - NMF

Recommendation models that rank users or items to use with implicit feedback:

- BPR: Bayesian Personalized Ranking from Implicit Feedback
- CTR: Collaborative Topic Regression

10.2.16 Optimization and Scoring Metrics

The `ML_TRAIN` routine includes the `optimization_metric` option, and the `ML_SCORE` routine includes the `metric` option. Both of these options define a metric that must be compatible with the `task` type and the target data. Model Metadata includes the `optimization_metric` field.

For more information about scoring metrics, see: scikit-learn.org. For more information about forecasting metrics, see: sktime.org and statsmodels.org.

Classification Metrics

Binary-only metrics:

- f1
- precision
- recall
- roc_auc

Binary and multi-class metrics:

- accuracy

- balanced_accuracy (`ML_SCORE` only)
- f1_macro
- f1_micro
- f1_samples (`ML_SCORE` only)
- f1_weighted
- neg_log_loss
- precision_macro
- precision_micro
- precision_samples (`ML_SCORE` only)
- precision_weighted
- recall_macro
- recall_micro
- recall_samples (`ML_SCORE` only)
- recall_weighted

Regression Metrics

- neg_mean_absolute_error
- neg_mean_squared_error
- neg_mean_squared_log_error
- neg_median_absolute_error
- r2

Forecasting Metrics

- neg_max_absolute_error
- neg_mean_absolute_error
- neg_mean_abs_scaled_error
- neg_mean_squared_error
- neg_root_mean_squared_error
- neg_root_mean_squared_percent_error
- neg_sym_mean_abs_percent_error

Anomaly Detection Metrics

Metrics for anomaly detection can only be used with the `ML_SCORE` routine. They cannot be used with the `ML_TRAIN` routine.

- roc_auc: You must not specify `threshold` or `topk` options.
- `precision_k`: An Oracle implementation of a common metric for fraud detection and lead scoring. You must use the `topk` option. You cannot use the `threshold` option.

The following metrics can use the `threshold` option, but cannot use the `topk` option:

- accuracy
- balanced_accuracy
- f1
- neg_log_loss
- precision
- recall

Recommendation Model Metrics

The following rating metrics can be used for explicit feedback:

- neg_mean_absolute_error
- neg_mean_squared_error
- neg_root_mean_squared_error
- r2

For recommendation models that use implicit feedback:

- If a user and item combination in the input table is not unique, the input table is grouped by user and item columns, and the result is the average of the rankings.
- If the input table overlaps with the training table, and `remove_seen` is `true`, which is the default setting, then the model will not repeat a recommendation and it ignores the overlap items.

The following ranking metrics can be used for implicit and explicit feedback:

- `precision_at_k` is the number of relevant `topk` recommended items divided by the total `topk` recommended items for a particular user:

  `precision_at_k` = (relevant `topk` recommended items) / (total `topk` recommended items)

  For example, if 7 out of 10 items are relevant for a user, and `topk` is 10, then `precision_at_k` is 70%.

  The `precision_at_k` value for the input table is the average for all users. If `remove_seen` is `true`, the default setting, then the average only includes users for whom the model can make a recommendation. If a user has implicitly ranked every item in the training table, the model cannot recommend any more items for that user, and they are ignored from the average calculation if `remove_seen` is `true`.

- `recall_at_k` is the number of relevant `topk` recommended items divided by the total relevant items for a particular user:

  `recall_at_k` = (relevant `topk` recommended items) / (total relevant items)

  For example, there is a total of 20 relevant items for a user. If `topk` is 10, and 7 of those items are relevant, then `recall_at_k` is 7 / 20 = 35%.

  The `recall_at_k` value for the input table is the average for all users.

- `hit_ratio_at_k` is the number of relevant `topk` recommended items divided by the total relevant items for all users:

`hit_ratio_at_k` = (relevant `topk` recommended items, all users) / (total relevant items, all users)

The average of `hit_ratio_at_k` for the input table is `recall_at_k`. If there is only one user, `hit_ratio_at_k` is the same as `recall_at_k`.

- `ndcg_at_k` is normalized discounted cumulative gain, which is the discounted cumulative gain of the relevant `topk` recommended items divided by the discounted cumulative gain of the relevant `topk` items for a particular user.

  The discounted gain of an item is the true rating divided by $\log_2(r+1)$ where `r` is the ranking of this item in the relevant `topk` items. If a user prefers a particular item, the rating is higher, and the ranking is lower.

  The `ndcg_at_k` value for the input table is the average for all users.

10.3 MySQL HeatWave GenAI Routines

Topics in this section describe MySQL HeatWave GenAI routines.

Some of the MySQL HeatWave GenAI routines reside in the MySQL `sys` schema.

MySQL JavaScript Stored Programs include a GenAI API that you can use to call different MySQL HeatWave GenAI routines using JavaScript functions. For more information, see JavaScript GenAI API.

10.3.1 ML_GENERATE

The `ML_GENERATE` routine uses the specified large language model (LLM) to generate text-based content as a response for the given natural-language query.

As of MySQL 9.4.1, the `ML_GENERATE` lets you generate text-based responses for queries pertaining to an image.

This topic contains the following sections:

- ML_GENERATE Syntax
- Syntax Examples
- See Also

ML_GENERATE Syntax

```
mysql> SELECT sys.ML_GENERATE('QueryInNaturalLanguage', options);

options: {JSON_OBJECT(keyvalue[, keyvalue]...)|NULL}
keyvalue:
{
  'task', {'generation'|'summarization'}
  |'model_id', 'ModelID'
  |'context', 'Context'
  |'language', 'Language'
  |'temperature', Temperature
  |'max_tokens', MaxTokens
  |'top_k', K
  |'top_p', P
  |'repeat_penalty', RepeatPenalty
  |'frequency_penalty', FrequencyPenalty
  |'presence_penalty', PresencePenalty
  |'stop_sequences', JSON_ARRAY('StopSequence'[, 'StopSequence'] ...)
  |'speculative_decoding', {true|false}
  |'image', Base64ImageEncoding
}
```

Following are `ML_GENERATE` parameters:

- *`QueryInNaturalLanguage`*: specifies the natural-language query that is passed to the large language model (LLM) handle.
- `options`: specifies optional parameters as key-value pairs in JSON format. It can include the following parameters:
  - `task`: specifies the task expected from the LLM. Default value is `generation`. Possible values are:
    - `generation`: generates text-based content.
    - `summarization`: generates a summary for existing text-based content.
  - `model_id`: specifies the LLM to use for the task.

    As of MySQL 9.3.1, default value is `llama3.2-3b-instruct-v1`. In earlier versions of MySQL, default value is `mistral-7b-instruct-v1`. Possible values are:
    - As of MySQL 9.4.0, to use an OCI Generative AI service Dedicated AI Cluster model, specify the model endpoint OCID. For more information, see Creating an Endpoint in Generative AI.
    - The following MySQL HeatWave In-Database LLMs are available as of MySQL 9.3.1:
      - `llama3.1-8b-instruct-v1`
      - `llama3.2-1b-instruct-v1`
      - `llama3.2-3b-instruct-v1`
      - `mistral-7b-instruct-v3`
    - The following OCI Generative AI Service LLMs are available as of MySQL 9.2.2:
      - `meta.llama-3.2-90b-vision-instruct`
      - `meta.llama-3.3-70b-instruct`
    - The following OCI Generative AI Service LLMs are available as of MySQL 9.1.2:
      - `cohere.command-r-plus-08-2024`
      - `cohere.command-r-08-2024`
      - `meta.llama-3.1-405b-instruct`
    - The following MySQL HeatWave In-Database LLMs are available as of MySQL 9.0.0:
      - `mistral-7b-instruct-v1`

        As of MySQL 9.4.0, support for `mistral-7b-instruct-v1` is deprecated.
      - `llama2-7b-v1`

        As of MySQL 9.0.1-u1, support for `llama2-7b-v1` is deprecated.
      - `llama3-8b-instruct-v1`

        As of MySQL 9.4.0, support for `llama3-8b-instruct-v1` has been deprecated.

    To view the lists of available LLMs, see MySQL HeatWave In-Database LLMs and OCI Generative AI Service LLMs.

Note

The `summarization` task supports MySQL HeatWave In-Database LLMs only.

- `context`: specifies the context to be used for augmenting the query and guide the text generation of the LLM. Default value is `NULL`.
- `language`: specifies the language to be used for writing queries, ingesting documents, and generating the output. To set the value of the `language` parameter, use the two-letter `ISO 639-1` code for the language. This parameter is available as of MySQL 9.0.1-u1.

  Default value is `en`.

  For possible values, to view the list of supported languages, see Languages.
- `temperature`: specifies a non-negative float that tunes the degree of randomness in generation. Lower temperatures mean less random generations.

  Default value is `0` for all LLMs.

  Possible values are float values between:

  - `0` and `5` for the MySQL HeatWave In-Database LLMs.
  - `0` and `1` for the Cohere OCI Generative AI Service LLMs.
  - `0` and `5` for the Meta OCI Generative AI Service LLMs.

  It is suggested that:

  - To generate the same output for a particular prompt every time you run it, set the temperature to `0`.
  - To generate a random new statement for a particular prompt every time you run it, increase the temperature.
- `max_tokens`: specifies the maximum number of tokens to predict per generation using an estimate of three tokens per word. Default value is `256`. Possible values are:

  - For Llama 3.1 and 3.2 LLMs, integer values between `1` and `128256`.
  - For Mistral V3 LLM, integer values between `1` and `32000`.
  - For Mistral V1 LLM, integer values between `1` and `8000`.
  - For Llama 2 and 3 LLMs, integer values between `1` and `4096`.
  - For OCI Generative AI Service LLMs, integer values between `1` and `4000`.
- `top_k`: specifies the number of top most likely tokens to consider for text generation at each step. Default value is `40`, which means that top 40 most likely tokens are considered for text generation at each step. Possible values are integer values between `0` and `32000`.
- `top_p`: specifies a number, `p`, and ensures that only the most likely tokens with the sum of probabilities `p` are considered for generation at each step. A higher value of `p` introduces more randomness into the output. Default value is `0.95`. Possible values are float values between `0` and `1`.

  - To disable this method, set to `1.0` or `0`.

  - To eliminate tokens with low likelihood, assign `p` a lower value. For example, if set to `0.1`, tokens within top 10% probability are included.
  - To include tokens with low likelihood, assign `p` a higher value. For example, if set to `0.9`, tokens within top 90% probability are included.

  If you are also specifying the `top_k` parameter, the LLM considers only the top tokens whose probabilities add up to `p` percent. It ignores the rest of the `k` tokens.

- `repeat_penalty`: assigns a penalty when a token appears repeatedly. High penalties encourage less repeated tokens and produce more random outputs. Default value is `1.1`. Possible values are float values between `0` and `2`.

  Note

  This parameter is supported for MySQL HeatWave In-Database LLMs only.

- `frequency_penalty`: assigns a penalty when a token appears frequently. High penalties encourage less repeated tokens and produce more random outputs. Default value is `0`. Possible values are float values between `0` and `1`.

- `presence_penalty`: assigns a penalty to each token when it appears in the output to encourage generating outputs with tokens that haven't been used. This is similar to `frequency_penalty`, except that this penalty is applied equally to all tokens that have already appeared, irrespective of their exact frequencies.

  Note

  This parameter is supported for OCI Generative AI Service LLMs only.

  Default value is `0`. Possible values are:

  - For Cohere LLMs, float values between `0` and `1`.
  - For Meta LLMs, float values between `-2` and `2`.

- `stop_sequences`: specifies a list of characters such as a word, a phrase, a newline, or a period that tells the LLM when to end the generated output. If you have more than one stop sequence, then the LLM stops when it reaches any of those sequences. Default value is `NULL`.

- `speculative_decoding`: enables or disables speculative decoding when generating tokens with an LLM. If set to `true`, speculative decoding enables faster response token generation, which speeds up LLM text generation. Speculative decoding is supported for the `llama3.1-8b-instruct-v1` LLM, which uses `llama3.2-1B-instruct-v1` as the draft LLM. Default value is `true`.

  This parameter is available as of MySQL 9.3.2.

- `image`: specifies the Base64 binary encoding corresponding to an image. Default value is `NULL`.

  This parameter is available as of MySQL 9.4.1.

Syntax Examples

- Generating text-based content in English using the `mistral-7b-instruct-v3` model:

```
mysql> SELECT sys.ML_GENERATE("What is AI?", JSON_OBJECT("task", "generation", "model_id", "mistral-7b-in
```

- Summarizing English text using the `mistral-7b-instruct-v3` model:

```
mysql> SELECT sys.ML_GENERATE(@text, JSON_OBJECT("task", "summarization", "model_id", "mistral-7b-ins
```

Where, `@text` is set as shown below:

```
SET @text="Artificial Intelligence (AI) is a rapidly growing field that has the potential to
revolutionize how we live and work. AI refers to the development of computer systems that can
perform tasks that typically require human intelligence, such as visual perception, speech
recognition, decision-making, and language translation.\n\nOne of the most significant developments i
AI in recent years has been the rise of machine learning, a subset of AI that allows computers to lea
from data without being explicitly programmed. Machine learning algorithms can analyze vast amounts
of data and identify patterns, making them increasingly accurate at predicting outcomes and making
decisions.\n\nAI is already being used in a variety of industries, including healthcare, finance, and
transportation. In healthcare, AI is being used to develop personalized treatment plans for patients
based on their medical history and genetic makeup. In finance, AI is being used to detect fraud and m
investment recommendations. In transportation, AI is being used to develop self-driving cars and impr
traffic flow.\n\nDespite the many benefits of AI, there are also concerns about its potential impact
society. Some worry that AI could lead to job displacement, as machines become more capable of perfor
tasks traditionally done by humans. Others worry that AI could be used for malicious ";
```

- Generating content pertaining to an image:

```
SELECT sys.ML_GENERATE("Describe the image", JSON_OBJECT("model_id", "meta.llama-3.2-90b-vision-instr
```

Where the `@image_base64` variable contains the Base64 binary encoding corresponding to an image.

For small images, you can directly copy the Base64 binary encoding into a variable. For large images, you can load the Base64 binary encoding from the client machine using the `LOAD DATA` statement, and then copy the encoding to a variable as shown below:

```
CREATE TABLE demo_image_table (image_base64 LONGTEXT);
LOAD LOCAL DATA INFILE 'demo_image_base64'
          INTO TABLE demo_image_table
          CHARACTER SET binary
          FIELDS TERMINATED BY '\t'
          LINES TERMINATED BY '\r' (image_base64);
SELECT image_base64 FROM demo_image_table INTO @image_base64;
```

Where `demo_image_base64` is the name of the file containing the Base64 binary encoding of the image.

Note

To use the `LOCAL` modifier, the `local_infile` variable must be set to 1.

See Also

- Section 7.6, “Generate Text-Based Content”
- Section 10.3.2, “ML_GENERATE_TABLE”

10.3.2 ML_GENERATE_TABLE

The `ML_GENERATE_TABLE` routine runs multiple text generation or summarization queries in a batch, in parallel. The output generated for every input query is the same as the output generated by the `ML_GENERATE` routine.

Note

In versions older than MySQL 9.2.1, to alter an existing table or create a new table, MySQL requires you to set the sql-require-primary-key system variable to `0`.

This routine is available as of MySQL 9.0.1-u1.

This topic contains the following sections:

- ML_GENERATE_TABLE Syntax
- Syntax Examples
- See Also

To learn about the privileges you need to run this routine, see Section 7.3, “MySQL HeatWave GenAI Roles and Privileges”.

ML_GENERATE_TABLE Syntax

```
mysql> CALL sys.ML_GENERATE_TABLE('InputTableColumn', 'OutputTableColumn', options);

options: {JSON_OBJECT(keyvalue[, keyvalue]...)|NULL}
keyvalue:
{
  'task', {'generation'|'summarization'}
  |'model_id', 'ModelID'
  |'context_column', 'ContextColumn'
  |'language', 'Language'
  |'temperature', Temperature
  |'max_tokens', MaxTokens
  |'top_k', K
  |'top_p', P
  |'repeat_penalty', RepeatPenalty
  |'frequency_penalty', FrequencyPenalty
  |'presence_penalty', PresencePenalty
  |'stop_sequences', JSON_ARRAY('StopSequence'[, 'StopSequence'] ...)
  |'batch_size', BatchSize
  |'speculative_decoding', {true|false}
  |'image_column', InputTableImageColumn
}
```

Following are `ML_GENERATE_TABLE` parameters:

- *`InputTableColumn`*: specifies the names of the input database, table, and column that contains the natural-language queries. The *`InputTableColumn`* is specified in the following format: *`DBName`*.*`TableName`*.*`ColumnName`*.
  - The specified input table can be an internal or external table.
  - The specified input table must already exist, must not be empty, and must have a primary key.
  - The input column must already exist and must contain `text` or `varchar` values.
  - The input column must not be a part of the primary key and must not have `NULL` values or empty strings.
  - There must be no backticks used in the *`DBName`*, *`TableName`*, or *`ColumnName`* and there must be no period used in the *`DBName`* or *`TableName`*.
- *`OutputTableColumn`*: specifies the names of the database, table, and column where the generated text-based response is stored. The *`OutputTableColumn`* is specified in the following format: *`DBName`*.*`TableName`*.*`ColumnName`*.
  - The specified output table must be an internal table.
  - If the specified output table already exists, then it must be the same as the input table. And, the specified output column must not already exist in the input table. A new JSON column is added to the table. External tables are read only. So if input table is an external table, then it cannot be used to store the output.

- If the specified output table doesn't exist, then a new table is created. The new output table has key columns which contains the same primary key values as the input table and a JSON column that stores the generated text-based responses.
- There must be no backticks used in the *`DBName`*, *`TableName`*, or *`ColumnName`* and there must be no period used in the *`DBName`* or *`TableName`*.

- `options`: specifies optional parameters as key-value pairs in JSON format. It can include the following parameters:

  - `task`: specifies the task expected from the large language model (LLM). Default value is `generation`. Possible values are:

    - `generation`: generates text-based content.
    - `summarization`: generates a summary for existing text-based content.

  - `model_id`: specifies the LLM to use for the task.

    As of MySQL 9.3.1, default value is `llama3.2-3b-instruct-v1`. In earlier versions of MySQL, default value is `mistral-7b-instruct-v1`. Possible values are:

    - As of MySQL 9.4.0, to use an OCI Generative AI service Dedicated AI Cluster model, specify the model endpoint OCID. For more information, see Creating an Endpoint in Generative AI.
    - The following MySQL HeatWave In-Database LLMs are available as of MySQL 9.3.1:
      - `llama3.1-8b-instruct-v1`
      - `llama3.2-1b-instruct-v1`
      - `llama3.2-3b-instruct-v1`
      - `mistral-7b-instruct-v3`
    - The following OCI Generative AI Service LLMs are available as of MySQL 9.2.2:
      - `meta.llama-3.2-90b-vision-instruct`
      - `meta.llama-3.3-70b-instruct`
    - The following OCI Generative AI Service LLMs are available as of MySQL 9.1.2:
      - `cohere.command-r-plus-08-2024`
      - `cohere.command-r-08-2024`
      - `meta.llama-3.1-405b-instruct`
    - The following MySQL HeatWave In-Database LLMs are available as of MySQL 9.0.0:
      - `mistral-7b-instruct-v1`

        As of MySQL 9.4.0, support for `mistral-7b-instruct-v1` is deprecated.

      - `llama2-7b-v1`

        As of MySQL 9.0.1-u1, support for `llama2-7b-v1` is deprecated.

      - `llama3-8b-instruct-v1`

        As of MySQL 9.4.0, support for `llama3-8b-instruct-v1` has been deprecated.

    To view the lists of available LLMs, see MySQL HeatWave In-Database LLMs and OCI Generative AI Service LLMs.

Note

The `summarization` task supports MySQL HeatWave In-Database LLMs only.

- `context_column`: specifies the table column that contains the context to be used for augmenting the queries and guiding the text generation of the LLM. The specified column must be an existing column in the input table. Default value is `NULL`.
- `language`: specifies the language to be used for writing queries, ingesting documents, and generating the output. To set the value of the `language` parameter, use the two-letter `ISO 639-1` code for the language.

  Default value is `en`.

  For possible values, to view the list of supported languages, see Languages.
- `temperature`: specifies a non-negative float that tunes the degree of randomness in generation. Lower temperatures mean less random generations.

  Default value is `0` for all LLMs.

  Possible values are float values between:

  - `0` and `5` For the MySQL HeatWave In-Database LLMs.
  - `0` and `1` for the Cohere OCI Generative AI Service LLMs.
  - `0` and `5` for the Meta OCI Generative AI Service LLMs.

  It is suggested that:

  - To generate the same output for a particular prompt every time you run it, set the temperature to `0`.
  - To generate a random new statement for a particular prompt every time you run it, increase the temperature.
- `max_tokens`: specifies the maximum number of tokens to predict per generation using an estimate of three tokens per word. Default value is `256`. Possible values are:

  - For Llama 3.1 and 3.2 LLMs, integer values between `1` and `128256`.
  - For Mistral V3 LLM, integer values between `1` and `32000`.
  - For Mistral V1 LLM, integer values between `1` and `8000`.
  - For Llama 2 and 3 LLMs, integer values between `1` and `4096`.
  - For OCI Generative AI Service LLMs, integer values between `1` and `4000`.
- `top_k`: specifies the number of top most likely tokens to consider for text generation at each step. Default value is `40`, which means that top 40 most likely tokens are considered for text generation at each step. Possible values are integer values between `0` and `32000`.
- `top_p`: specifies a number, `p`, and ensures that only the most likely tokens with the sum of probabilities `p` are considered for generation at each step. A higher value of `p` introduces more randomness into the output. Default value is `0.95`. Possible values are float values between `0` and `1`.

  - To disable this method, set to `1.0` or `0`.

- To eliminate tokens with low likelihood, assign `p` a lower value. For example, if set to `0.1`, tokens within top 10% probability are included.
- To include tokens with low likelihood, assign `p` a higher value. For example, if set to `0.9`, tokens within top 90% probability are included.

If you are also specifying the `top_k` parameter, the LLM considers only the top tokens whose probabilities add up to `p` percent. It ignores the rest of the `k` tokens.

- `repeat_penalty`: assigns a penalty when a token appears repeatedly. High penalties encourage less repeated tokens and produce more random outputs. Default value is `1.1`. Possible values are float values between `0` and `2`.

  Note

  This parameter is supported for MySQL HeatWave In-Database LLMs only.

- `frequency_penalty`: assigns a penalty when a token appears frequently. High penalties encourage less repeated tokens and produce more random outputs. Default value is `0`. Possible values are float values between `0` and `1`.
- `presence_penalty`: assigns a penalty to each token when it appears in the output to encourage generating outputs with tokens that haven't been used. This is similar to `frequency_penalty`, except that this penalty is applied equally to all tokens that have already appeared, irrespective of their exact frequencies.

  Note

  This parameter is supported for OCI Generative AI Service LLMs only.

  Default value is `0`. Possible values are:

  - For Cohere LLMs, float values between `0` and `1`.
  - For Meta LLMs, float values between `-2` and `2`.

- `stop_sequences`: specifies a list of characters such as a word, a phrase, a newline, or a period that tells the LLM when to end the generated output. If you have more than one stop sequence, then the LLM stops when it reaches any of those sequences. Default value is `NULL`.
- `batch_size`: specifies the batch size for the routine. This parameter is supported for internal tables only. Default value is `1000`. Possible values are integer values between `1` and `1000`.
- `speculative_decoding`: enables or disables speculative decoding when generating tokens with an LLM. If set to `true`, speculative decoding enables faster response token generation, which speeds up LLM text generation. Speculative decoding is supported for the `llama3.1-8b-`

`instruct-v1` LLM, which uses `llama3.2-1B-instruct-v1` as the draft LLM. Default value is `true`.

This parameter is available as of MySQL 9.3.2.

- `image_column`: specifies the name of the column in the input table where the the base64 encoding of images are stored.
  - The image column must exist in the same table as the *`InputTableColumn`* and must contain `text` or `varchar` values.
  - The image column must not be a primary key, part of the primary key, and must be different from the input column used for providing text inputs. And, the image column must not have `NULL` values or empty strings.
  - The image column name must not contain back ticks (`).

  This parameter is available as of MySQL 9.4.1.

Syntax Examples

- Generate English text-based content in a batch using the `mistral-7b-instruct-v3` model for queries stored in `demo_db.input_table`:

```
mysql> CALL sys.ML_GENERATE_TABLE("demo_db.input_table.Input", "demo_db.output_table.Output", JSON_OB
```

- Generate text-based content in a batch pertaining to images stored in the `demo_db.input_table.Images` column using queries stored in the `demo_db.input_table.Input` column:

```
mysql> CALL sys.ML_GENERATE_TABLE("demo_db.input_table.Input", "demo_db.output_table.Output", JSON_OB
```

See Also

- Generate New Text - Run Batch Queries
- Summarize Content - Run Batch Queries
- Section 10.3.1, “ML_GENERATE”

10.3.3 VECTOR_STORE_LOAD

The `VECTOR_STORE_LOAD` routine generates vector embedding for the specified files or folders that are stored in the bucket, and loads the embeddings into a new vector store table.

This routine creates an asynchronous task which loads vector store tables in the background. It also returns a query that you can run to track the status of the vector store load task that is running in the background.

Note

It is recommended that you create five or less vector store load tasks at a time. Too many tasks running at the same time might cause overloading issues.

This topic contains the following sections:

- VECTOR_STORE_LOAD Syntax
- Syntax Examples
- See Also

To learn about the privileges you need to run this routine, see Section 7.3, “MySQL HeatWave GenAI Roles and Privileges”.

VECTOR_STORE_LOAD Syntax

```
mysql> CALL sys.VECTOR_STORE_LOAD('URI', options);

options: {JSON_OBJECT(keyvalue[, keyvalue]...)|NULL}
keyvalue:
{
  'formats', JSON_ARRAY('Format'[, 'Format'] ...)
  |'schema_name', 'SchemaName'
  |'table_name', 'TableName'
  |'region', 'Region'
  |'task_name', 'TaskName'
  |'language', 'Language'
  |'embed_model_id', {'EmbeddingModelID'}
  |'description', 'Description'
  |'uris', JSON_ARRAY(urioptions[, urioptions] ...)
  |'ocr', {true|false}
  |'chunking', chunkingoptions
}
```

Following are `VECTOR_STORE_LOAD` parameters:

- *`URI`*: specifies the unique reference index (URI) or pre-authenticated request (PAR) of the bucket files or folders to be ingested into the vector store.

  A URI is considered to be one of the following:

  - A glob pattern, if it contains at least one unescaped `?` or `*` character.
  - A prefix, if it is not a pattern and ends with a `/` character like a folder path.
  - A file path, if it is neither a glob pattern nor a prefix.

  To learn how to create PAR for your , see Creating a PAR request in .

- `options`: specifies optional parameters as key-value pairs in JSON format. It can include the following parameters:

  - `formats`: specifies the list of formats to be loaded. The supported file formats are `pdf`, `ppt`, `txt`, `html`, and `doc`. By default, the routine uses all the supported formats.

    If the routine detects multiple files with the same or different file formats in a single load, it creates a separate table for every format it finds. The table name for each format is the specified or default table name followed by the format: *`TableName_Format`*.

  - `schema_name`: specifies the name of the schema where the vector embeddings are to be loaded. By default, this procedure uses the current schema from the session.

  - `table_name`: specifies the name of the vector store table to create. By default, the routine generates a unique table name with format `vector_store_data_x`, where `x` is a counter.

  - `region`: specifies the region of the bucket. Default value is the region where the current DB System is running.

  - `task_name`: specifies a name for the loading task to be mentioned in the task status. Default value is `Vector Store Loader`.

  - `language`: specifies the text content language used in the files to be ingested into the vector store. To set the value of the `language` parameter, use the two-letter `ISO 639-1` code for the language. This parameter is available as of MySQL 9.0.1-u1.

    Default value is `en`.

    For possible values, to view the list of supported languages, see Languages.

- `embed_model_id`: specifies the embedding model to use for encoding the text.

  As of 9.3.0, default value is `multilingual-e5-small`. In earlier versions of MySQL, default value is `minilm` for English text and `multilingual-e5-small` for text in languages other than English.

  For possible values, to view the list of available embedding models, see MySQL HeatWave In-Database Embedding Models and OCI Generative AI Service Embedding Models. OCI Generative AI Service models are supported as of MySQL 9.5.0. In earlier versions of MySQL, if you want to use OCI Generative AI Service Embedding Models, then you can use the HEATWAVE_LOAD routine to ingest files into a vector store.

  This parameter is available as of MySQL 9.2.1.

- `description`: specifies a description of document collection being loaded to be mentioned in the task status. Default value is `NULL`.

- `uris`: specifies a list of additional URIs to include along with an optional name for the vector store table to be created for the specified URI. Default value is `NULL`.

  Each object in the list can include the following parameters:

  ```
  urioptions: JSON_OBJECT(urioptskeyvalue[, urioptskeyvalue])
  urioptskeyvalue:
  {
    'uri', 'URI'
    |'table_name', 'TableName'
  }
  ```

  - `uri`: specifies the additional URI. If only `uri` is provided, the routine uses the specified URI as an additional URI, and loads it into the main table `options.table_name` or the generated table with the unique table name.

  - `table_name`: if both `uri` and `table_name` are provided, the routine loads the specified URI into the specified table.

    If only `table_name` is provided, the routine loads the specified vector store table into MySQL HeatWave.

- `ocr`: specifies whether to enable or disable Optical Character Recognition (OCR). If set to `false`, disables OCR. Default value is `true`, which means OCR is enabled by default.

  This parameter is available as of MySQL 9.1.1.

  - As of MySQL 9.1.2, the default value is `true`.

  - As of MySQL 9.1.1, default value is `false`.

- `chunking`: specifies parameter values for customized text segmentation during vector store creation.

  It can include the following parameters:

  ```
  chunkingoptions: JSON_OBJECT(chunkingkeyvalue[, chunkingkeyvalue]...)
  chunkingkeyvalue:
  {
    'split_by', {'page'|'paragraph'|'sentence'|'document'|'recursive'}
    |'max', MaxValue
    |'by', {'characters'|'words'}
    |'truncate', {true|false}
    |'overlap', OverlapValue
  ```

```
}
```

- `split_by`: specifies the method to use for splitting the text into segments. It can be one of the following:

  - `page`: for text segmentation based on the pagination available in the document. This segmentation method is supported for PDF and PPT documents only. If used for other documents, the routine falls back to the `document` text segmentation method for unsupported documents.

  - `paragraph`: for text segmentation based on the paragraphs identified in the document. Wherein, a paragraph is a piece of text separated from another piece of text using `\n\n` characters.

  - `sentence`: for text segmentation based on the sentences identified in the document. Wherein, a sentence is a sequence of words that is separated from another sequence of words using a punctuation that marks the end of a sentence: `.`, `!`, or `?`. For PDF files, when OCR is enabled, any sentence that spills into the next page is considered as a two separate sentences.

  - `document`: for putting an entire document into one text segment. In case the `max` parameter is not set, you cannot overlap across documents.

  - `recursive`: for using the default text segmentation method used while creating vector store tables in previous versions of MySQL. This method provides backward compatibility with vector store tables created in previous versions of MySQL.

  Default value is `recursive`.

- `max`: specifies the maximum number of characters or words to be included in each segment. If left unspecified, there is no maximum size enforcement on the chunks of text. For words, this value can be up to `100000` and for characters, this value can be up to `1000000`. This is unspecified by default.

- `by`: specifies the unit to use for defining the maximum and overlap text segment limits using the `max` and `overlap` parameter. It can be set to `characters` or `words`. Default value is `characters`.

- `truncate`: if set to `true`, enables truncation of text segments that are too large for the embedding model to handle without truncation. However, when tructation is enabled, the end of the text segment is trimmed out, and you might lose the information from the part of the text segment that is trimmed out.

  If set to `false`, disables truncation and instead throw an error when a text segment that is too large for the embeeding model is found. In this case, the entire load fails and exits with an error. You can adjust the text segmentation method used to ensure that the size of the text segments is within the segment size limit of the embedding model.

  Default value is `true`.

- `overlap`: specifies the maximum number of characters or words to overlap between segments on each side. For words, this value can be up to `50000` and for characters, this value can be up to `500000`. Default value is `0`.

The `chunking` parameters are available as of MySQL 9.5.0.

Syntax Examples

- Specifying the file to ingest using the URI in `VECTOR_STORE_LOAD`:

```
mysql> CALL sys.VECTOR_STORE_LOAD('oci://demo_bucket@demo_namespace/heatwave-en.pdf', '{"table_name": "de
```

- Specifying the file to ingest using the PAR in VECTOR_STORE_LOAD:

```
mysql> CALL sys.VECTOR_STORE_LOAD('https://demo.objectstorage.us-ashburn-1.oci.customer-oci.com/p/dem
```

- Specifying additional options such the schema name, table name, language, format, and table description in VECTOR_STORE_LOAD:

```
mysql> CALL sys.VECTOR_STORE_LOAD('oci://demo_bucket@demo_namespace/german_files/de*', '{"schema_name
```

- Tracking the progress of a load task by running the task query displayed as output for the VECTOR_STORE_LOAD routine:

  - As of MySQL 9.3.1:

    ```
    SELECT mysql_tasks.task_status_brief("TaskID");
    ```

    The output looks similar to the following:

    ```
    +-----------------------------------------------------------------------------------------+
    | mysql_tasks.task_status_brief("TaskID")                                                 |
    +-----------------------------------------------------------------------------------------+
    | {"data": {"tables_to_load": "[{\"table_name\": \"quickstart_embeddings\",               |
    | \"load_progress\": 40.0}]"}, "status": "RUNNING", "message": "Loading in progress...",  |
    | "progress": 40}                                                                         |
    +-----------------------------------------------------------------------------------------+
    ```

  - In earlier versions:

    ```
    mysql> SELECT id, name, message, progress, status, scheduled_time,estimated_completion_time, estima
    ```

    The output looks similar to the following:

    ```
               id: 1
             name: Vector Store Loader
          message: Task starting.
          progress: 0
            status: RUNNING
    scheduled_time: 2024-07-02 14:42:38
    estimated_completion_time: 2024-07-22 10:19:53
    estimated_remaining_time: 52.50000
      progress_bar: __________
    ```

    Getting more details about the load task by querying the task logs for the given task id:

    ```
    SELECT * from mysql_task_management.task_log where task_id = 1;
    ```

    The output looks similar to the following:

    ```
    +----------------------------------+---------+----------------------------+--------------------
    | id                               | task_id | log_time                   | message
    +----------------------------------+---------+----------------------------+--------------------
    | 0x11EF799F5D99054288CC020017091C01 |       1 | 2024-09-23 11:31:24.884514 | Task created by user
    | 0x11EF799F5D99812188CC020017091C01 |       1 | 2024-09-23 11:31:24.887685 | Task starting.
    | 0x11EF799F6390FD9788CC020017091C01 |       1 | 2024-09-23 11:31:34.898219 | Loading in progress.
    | 0x11EF799F668C172F88CC020017091C01 |       1 | 2024-09-23 11:31:39.899271 | Loading in progress.
    | 0x11EF799F6987348588CC020017091C01 |       1 | 2024-09-23 11:31:44.900419 | Loading in progress.
    | 0x11EF799F6C82547D88CC020017091C01 |       1 | 2024-09-23 11:31:49.901634 | Loading in progress.
    +----------------------------------+---------+----------------------------+--------------------
    ```

See Also

Ingest Files Using Asynchronous Load

10.3.4 ML_RAG

The ML_RAG routine performs retrieval-augmented generation (RAG) by:

1. Taking a natural-language query.

2. Retrieving context from relevant documents using semantic search.
3. Generating a response that integrates information from the retrieved documents.

This routine aims to provide detailed, accurate, and contextually relevant answers by augmenting a generative model with information retrieved from a comprehensive knowledge base.

This topic contains the following sections:

- ML_RAG Syntax
- Syntax Examples
- See Also

ML_RAG Syntax

```
mysql> CALL sys.ML_RAG('QueryInNaturalLanguage', 'Output', options);

options: {JSON_OBJECT(keyvalue[, keyvalue]...)|NULL}
keyvalue:
{
  'vector_store', JSON_ARRAY('VectorStoreTableName'[, 'VectorStoreTableName']...)
  |'schema', JSON_ARRAY('SchemaName'[, 'SchemaName']...)
  |'n_citations', NumberOfCitations
  |'distance_metric', {'COSINE'|'DOT'|'EUCLIDEAN'}
  |'document_name', JSON_ARRAY('DocumentName'[, 'DocumentName']...)
  |'skip_generate', {true|false}
  |'model_options', modeloptions
  |'exclude_vector_store', JSON_ARRAY('ExcludeVectorStoreTableName'[, 'ExcludeVectorStoreTableName']...)
  |'exclude_document_name', JSON_ARRAY('ExcludeDocumentName'[, 'ExcludeDocumentName']...)
  |'retrieval_options', retrievaloptions
  |'vector_store_columns', vscoptions
  |'embed_model_id', 'EmbeddingModelID'
  |'query_embedding', 'QueryEmbedding'
  |'semantic_search', {true|false}
  |'keyword_search', {true|false}
}
```

Following are `ML_RAG` parameters:

- *`QueryInNaturalLangugae`*: specifies the natural-language query.
- *`Output`*: stores the generated output. The output contains the following segments:
  - `text`: the generated text-based response.
  - `citations`: contains the following details:
    - `segment`: the textual content that is retrieved from the vector store through semantic search, and used as context generating the response.
    - `distance`: the distance between the query embedding the segment embedding.
    - `document_name`: the name of the document from which the segment is retrieved.
  - `vector_store`: the list of vector store tables used for context retrieval.
- `options`: specifies optional parameters as key-value pairs in JSON format. It can include the following parameters:
  - `vector_store`: specifies a list of loaded vector store tables to use for context retrieval. The routine ignores invalid table names. By default, the routine performs a global search across all the available vector store tables in the DB system.

- `schema`: specifies a list of schemas to check for loaded vector store tables. By default, the routine performs a global search across all the available vector store tables in all the schemas that are available in the DB system.
- `n_citations`: specifies the number of segments to consider for context retrieval. Default value is `3`. Possible values are integer values between `0` and `100`.
- `distance_metric`: specifies the distance metrics to use for context retrieval. Default value is `COSINE`. Possible values are `COSINE`, `DOT`, and `EUCLIDEAN`.
- `document_name`: limits the documents to use for context retrieval. Only the specified documents are used. By default, the routine performs a global search across all the available documents stored in all the available vector stores in the DB system.
- `skip_generate`: specifies whether to skip generation of the text-based response, and only perform context retrieval from the available or specified vector stores, schemas, or documents. Default value is `false`.
- `model_options`: additional options that you can set for generating the text-based response. These are the same options that are available in the `ML_GENERATE` routine, which alter the text-based response per the specified settings. However, the `context` option is not supported as an `ML_RAG` model option. As of MySQL 9.3.1, default value is `'{"model_id": "llama3.2-3b-instruct-v1"}'`, in previous versions, default value is `'{"model_id": "mistral-7b-instruct-v1"}'`.
- `exclude_vector_store`: specifies a list of loaded vector store tables to exclude from context retrieval. The routine ignores invalid table names. Default value is `NULL`. This option is available as of MySQL 9.0.1-u1.
- `exclude_document_name`: specifies a list of documents to exclude from context retrieval. Default value is `NULL`. This parameter is available as of MySQL 9.0.1-u1.
- `retrieval_options`: specifies optional context retrieval parameters as key-value pairs in JSON format. If a parameter value in `retrieval_options` is set to `auto`, the default value for that parameter is used.

  The `retrieval_options` parameters are available as of MySQL 9.1.2.

  It can include the following parameters:

```
retrievaloptions: JSON_OBJECT(retrievaloptkeyvalue[, retrievaloptkeyvalue]...)
retrievaloptkeyvalue:
{
  'max_distance', MaxDistance
  |'percentage_distance', PercentageDistance
  |'segment_overlap', SegmentOverlap
}
```

  - `max_distance`: specifies a maximum distance threshold for filtering out segments from context retrieval. Segments for which the distance from the input query exceeds the specified maximum distance threshold are excluded from content retrieval. This ensures that only the segments that

are closer to the input query are included during context retrieval. However, if no segments are found within the specified distance, the routine fails to run.

Note

If this parameter is set, the default value of the `n_citations` parameter is automatically updated to `10`.

Default value is `0.6` for all distance metrics.

Possible values are decimal values between `0` and `999999.9999`.

- `percentage_distance`: specifies what percentage of distance to the nearest segment is to be used to determine the maximum distance threshold for filtering out segments from context retrieval.

  Following is the formula used for calculating the maximum distance threshold:

  `MaximumDistanceThreshold` = *`DistanceOfInputQueryToNearestSegment`* + [(`percentage_distance` / `100`) * *`DistanceOfInputQueryToNearestSegment`*]

  Which means that the segments for which the distance to the input query exceeds the distance of the input query to the nearest segment by the specified percentage are filtered out from context retrieval.

  Note

  If this parameter is set, the default value of the `n_citations` parameter is automatically updated to `10`.

  Default value is `20` for all distance metrics.

  Possible values are decimal values between `0` and `999999.9999`.

  Note

  If both `max_distance` and `percentage_distance` are set, the smaller threshold value is considered for filtering out the segments.

- `segment_overlap`: specifies the number of additional segments adjacent to the nearest segments to the input query to be included in context retrieval. These additional segments provide more continuous context for the input query. Default value is `1`. Possible values are integer values between `0` and `5`.

- `vector_store_columns`: specifies column names for finding relevant vector and embedding tables for context retrieval as key-value pairs in JSON format. If multiple tables contain columns with the same name and data type, then all such tables are used for context retrieval.

  It can include the following parameters:

```
vscoptions: JSON_OBJECT('segment', 'SegmentColName', 'segment_embedding', 'EmbeddingColName'[, vsckeyva
vsckeyvalue:
{
  'document_name', 'DocumentName'
  |'document_id', DocumentID
  |'metadata', 'Metadata'
  |'segment_number', SegmentNumber
}
```

  - `segment`: specifies the name of the mandatory string column that contains the text segments. Default value is `segment`.

- `segment_embedding`: specifies the name of the mandatory vector column that contains vector embeddings of the text segments. Default value is `segment_embedding`.
- `document_name`: specifies the name of the optional column that contains the document names. This column can be of any data type supported by MySQL HeatWave. Default value is `document_name`.
- `document_id`: specifies the name of the optional integer column that contains the document IDs. Default value is `document_id`.
- `metadata`: specifies the name of the optional JSON column that contains additional table metadata. Default value is `metadata`.
- `segment_number`: specifies the name of the optional integer column that contains the segment numbers. Default value is `segment_number`.

Default value is `{"segment": "segment", "segment_embedding": "segment_embedding", "document_id: "document_id", "segment_number": "segment_number", "metadata": "metadata"}`, which means that by default, the routine uses the default values of all column names to find relevant tables for context retrieval.

This parameter is available as of MySQL 9.2.1.

- `embed_model_id`: specifies the embedding model to use for embedding the input query. If you are providing the query embedding, then set this parameter to specify the embedding model to use to embed the query. The routine uses vector store tables and embedding tables created using the same embedding model for context retrieval.

  As of 9.3.0, default value is `multilingual-e5-small`. In earlier versions of MySQL, default value is `minilm` for English text and `multilingual-e5-small` for text in languages other than English.

  To view the list of available embedding models, see MySQL HeatWave In-Database Embedding Models and OCI Generative AI Service Embedding Models.

  As of MySQL 9.4.0, to use an OCI Generative AI service Dedicated AI Cluster model, specify the model endpoint OCID. For more information, see Creating an Endpoint in Generative AI.

  This parameter is available as of MySQL 9.2.1.

- `query_embedding`: specifies the vector embedding of the input query. If this parameter is set, then the routine skips generating the vector embeddings of the input query. Instead, it uses this

embedding for context retrieval from valid vector store and embedding tables that contain vector embeddings created using the same embedding model.

This parameter is available as of MySQL 9.2.1.

- `semantic_search`: when set to `true`, enables vector-based semantic search also known as similarity search. When set to `false`, disables semantic search. Default value is `true`.

  This parameter is available as of MySQL 9.5.0.

- `keyword_search`: when set to `true`, enables keyword-based search. When set to `false`, disables keyword-based search. Default is `false`

  This parameter is available as of MySQL 9.5.0.

Note

Both `semantic_search` and `keyword_search` parameters cannot be set to `false` at the same time.

Syntax Examples

Retrieving context and generating output:

```
mysql> CALL sys.ML_RAG("What is AutoML",@output,@options);
```

Where, `@options` is set to specify the vector store table to use using `vector_store` key, as shown below:

```
mysql> SET @options = JSON_OBJECT("vector_store", JSON_ARRAY("demo_db.demo_embeddings"));
```

See Also

- Section 7.9, “Perform Vector Search with RAG”
- Section 10.3.5, “ML_RAG_TABLE”

10.3.5 ML_RAG_TABLE

The `ML_RAG_TABLE` routine runs multiple retrieval-augmented generation (RAG) queries in a batch, in parallel. The output generated for every input query is the same as the output generated by the `ML_RAG` routine.

Note

In versions older than MySQL 9.2.1, to alter an existing table or create a new table, MySQL requires you to set the sql-require-primary-key system variable to `0`.

This routine is available as of MySQL 9.0.1-u1.

This topic contains the following sections:

- ML_RAG_TABLE Syntax
- Syntax Examples
- See Also

To learn about the privileges you need to run this routine, see Section 7.3, “MySQL HeatWave GenAI Roles and Privileges”.

ML_RAG_TABLE Syntax

```
mysql> CALL sys.ML_RAG_TABLE('InputTableColumn', 'OutputTableColumn', options);
```

```
options: {JSON_OBJECT(keyvalue[, keyvalue]...)|NULL}
keyvalue:
{
  'vector_store', JSON_ARRAY('VectorStoreTableName'[, 'VectorStoreTableName']...)
  |'schema', JSON_ARRAY('SchemaName'[, 'SchemaName']...)
  |'n_citations', NumberOfCitations
  |'distance_metric', {'COSINE'|'DOT'|'EUCLIDEAN'}
  |'document_name', JSON_ARRAY('DocumentName'[, 'DocumentName']...)
  |'skip_generate', {true|false}
  |'model_options', modeloptions
  |'exclude_vector_store', JSON_ARRAY('ExcludeVectorStoreTableName'[, 'ExcludeVectorStoreTableName']...
  |'exclude_document_name', JSON_ARRAY('ExcludeDocumentName'[, 'ExcludeDocumentName']...)
  |'batch_size', BatchSize
  |'retrieval_options', retrievaloptions
  |'vector_store_columns', vscoptions
  |'embed_model_id', 'EmbeddingModelID'
  |'embed_column', 'EmbeddedQueriesColumnName'
  |'fail_on_embedding_error', {true|false}
  |'semantic_search', {true|false}
  |'keyword_search', {true|false}
}
```

Following are `ML_RAG_TABLE` parameters:

- *`InputTableColumn`*: specifies the names of the input database, table, and column that contains the natural-language queries. The *`InputTableColumn`* is specified in the following format: *`DBName`*.*`TableName`*.*`ColumnName`*.
  - The specified input table can be an internal or external table.
  - The specified input table must already exist, must not be empty, and must have a primary key.
  - The input column must already exist and must contain `text` or `varchar` values.
  - The input column must not be a part of the primary key and must not have `NULL` values or empty strings.
  - There must be no backticks used in the *`DBName`*, *`TableName`*, or *`ColumnName`* and there must be no period used in the *`DBName`* or *`TableName`*.
- *`OutputTableColumn`*: specifies the names of the database, table, and column where the generated text-based response is stored. The *`OutputTableColumn`* is specified in the following format: *`DBName`*.*`TableName`*.*`ColumnName`*.
  - The specified output table must be an internal table.
  - If the specified output table already exists, then it must be the same as the input table. And, the specified output column must not already exist in the input table. A new JSON column is added to the table. External tables are read only. So if input table is an external table, then it cannot be used to store the output.
  - If the specified output table doesn't exist, then a new table is created. The new output table has key columns which contains the same primary key values as the input table and a JSON column that stores the generated text-based responses.
  - There must be no backticks used in the *`DBName`*, *`TableName`*, or *`ColumnName`* and there must be no period used in the *`DBName`* or *`TableName`*.
- `options`: specifies optional parameters as key-value pairs in JSON format. It can include the following parameters:
  - `vector_store`: specifies a list of loaded vector store tables to use for context retrieval. The routine ignores invalid table names. By default, the routine performs a global search across all the available vector store tables in the DB system.

- `schema`: specifies a list of schemas to check for loaded vector store tables. By default, the routine performs a global search across all the available vector store tables in all the schemas that are available in the DB system.
- `n_citations`: specifies the number of segments to consider for context retrieval. Default value is `3`. Possible values are integer values between `0` and `100`.
- `distance_metric`: specifies the distance metrics to use for context retrieval. Default value is `COSINE`. Possible values are `COSINE`, `DOT`, and `EUCLIDEAN`.
- `document_name`: limits the documents to use for context retrieval. Only the specified documents are used. By default, the routine performs a global search across all the available documents stored in all the available vector stores in the DB system.
- `skip_generate`: specifies whether to skip generation of the text-based response, and only perform context retrieval from the available or specified vector stores, schemas, or documents. Default value is `false`.
- `model_options`: additional options that you can set for generating the text-based response. These are the same options that are available in the `ML_GENERATE` routine, which alter the text-based response per the specified settings. However, the `context` option is not supported as an `ML_RAG_TABLE` model option. As of MySQL 9.3.1, default value is `'{"model_id": "llama3.2-3b-instruct-v1"}'`, in previous versions, default value is `'{"model_id": "mistral-7b-instruct-v1"}'`.
- `exclude_vector_store`: specifies a list of loaded vector store tables to exclude from context retrieval. The routine ignores invalid table names. Default value is `NULL`.
- `exclude_document_name`: specifies a list of documents to exclude from context retrieval. Default value is `NULL`.
- `batch_size`: specifies the batch size for the routine. This parameter is supported for internal tables only. Default value is `1000`. Possible values are integer values between `1` and `1000`.
- `retrieval_options`: specifies optional context retrieval parameters as key-value pairs in JSON format. If a parameter value in `retrieval_options` is set to `auto`, the default value for that parameter is used.

  The `retrieval_options` parameters are available as of MySQL 9.1.2.

  It can include the following parameters:

```
retrievaloptions: JSON_OBJECT(retrievaloptkeyvalue[, retrievaloptkeyvalue]...)
retrievaloptkeyvalue:
{
  'max_distance', MaxDistance
  |'percentage_distance', PercentageDistance
  |'segment_overlap', SegmentOverlap
}
```

  - `max_distance`: specifies a maximum distance threshold for filtering out segments from context retrieval. Segments for which the distance from the input query exceeds the specified maximum distance threshold are excluded from content retrieval. This ensures that only the segments that

are closer to the input query are included during context retrieval. However, if no segments are found within the specified distance, the routine generates an output without using any context.

Note

If this parameter is set, the default value of the `n_citations` parameter is automatically updated to `10`.

Default value is `0.6` for all distance metrics.

Possible values are decimal values between `0` and `999999.9999`.

- `percentage_distance`: specifies what percentage of distance to the nearest segment is to be used to determine the maximum distance threshold for filtering out segments from context retrieval.

  Following is the formula used for calculating the maximum distance threshold:

  `MaximumDistanceThreshold` = *`DistanceOfInputQueryToNearestSegment`* + [(`percentage_distance` / `100`) * *`DistanceOfInputQueryToNearestSegment`*]

  Which means that the segments for which the distance to the input query exceeds the distance of the input query to the nearest segment by the specified percentage are filtered out from context retrieval.

  Note

  If this parameter is set, the default value of the `n_citations` parameter is automatically updated to `10`.

  Default value is `20` for all distance metrics.

  Possible values are decimal values between `0` and `999999.9999`.

  Note

  If both `max_distance` and `percentage_distance` are set, the smaller threshold value is considered for filtering out the segments.

- `segment_overlap`: specifies the number of additional segments adjacent to the nearest segments to the input query to be included in context retrieval. These additional segments provide more continuous context for the input query. Default value is `1`. Possible values are integer values between `0` and `5`.

- `vector_store_columns`: specifies column names for finding relevant vector and embedding tables for context retrieval as key-value pairs in JSON format. If multiple tables contain columns with the same name and data type, then all such tables are used for context retrieval.

  It can include the following parameters:

```
vscoptions: JSON_OBJECT('segment', 'SegmentColName', 'segment_embedding', 'EmbeddingColName'[, vsck
vsckeyvalue:
{
  'document_name', 'DocumentName'
  |'document_id', DocumentID
  |'metadata', 'Metadata'
  |'segment_number', SegmentNumber
}
```

  - `segment`: specifies the name of the mandatory string column that contains the text segments. Default value is `segment`.

- `segment_embedding`: specifies the name of the mandatory vector column that contains vector embeddings of the text segments. Default value is `segment_embedding`.
- `document_name`: specifies the name of the optional column that contains the document names. This column can be of any data type supported by MySQL HeatWave. Default value is `document_name`.
- `document_id`: specifies the name of the optional integer column that contains the document IDs. Default value is `document_id`.
- `metadata`: specifies the name of the optional JSON column that contains additional table metadata. Default value is `metadata`.
- `segment_number`: specifies the name of the optional integer column that contains the segment numbers. Default value is `segment_number`.

Default value is `{"segment": "segment", "segment_embedding": "segment_embedding", "document_id: "document_id", "segment_number": "segment_number", "metadata": "metadata"}`, which means that by default, the routine uses the default values of all column names to find relevant tables for context retrieval.

This parameter is available as of MySQL 9.2.1.

- `embed_model_id`: specifies the embedding model to use for embedding the input queries. If you are providing the query embeddings, then set this parameter to specify the embedding model to use to embed the queries. The routine uses vector store tables and embedding tables created using the same embedding model for context retrieval.

  As of MySQL 9.3.0, default value is `multilingual-e5-small`. In earlier versions of MySQL, default value is `all_minilm_l12_v2` if the output language is set to English and `multilingual-e5-small` if the output language is set to a language other than English.

  To view the list of available embedding models, see MySQL HeatWave In-Database Embedding Models and OCI Generative AI Service Embedding Models.

  As of MySQL 9.4.0, to use an OCI Generative AI service Dedicated AI Cluster model, specify the model endpoint OCID. For more information, see Creating an Endpoint in Generative AI.

  This parameter is available as of MySQL 9.2.1.

- `embed_column`: specifies the name of the input table colmn which contains vector embeddings of the input queries. If this parameter is set, then the routine skips generating the vector embeddings of the input queries. Instead, it uses the embeddings stored in this column for context retrieval from valid vector store and embedding tables that contain vector embeddings created using the same embedding model.

  This parameter is available as of MySQL 9.2.1.

- `fail_on_embedding_error`: if set to `true`, stops the batch processing of input queries and throws an error in case an error is encountered for an input row. If set to `false`, allows the batch

processing to partially fail for rows where errors are encountered, and lets the routine continue with processing the other rows. Default value is `true`.

This parameter is available as of MySQL 9.3.0.

- `semantic_search`: when set to `true`, enables vector-based semantic search also known as similarity search. When set to `false`, disables semantic search. Default value is `true`.

  This parameter is available as of MySQL 9.5.0.

- `keyword_search`: when set to `true`, enables keyword-based search. When set to `false`, disables keyword-based search. Default is `false`

  This parameter is available as of MySQL 9.5.0.

Note

Both `semantic_search` and `keyword_search` parameters cannot be set to `false` at the same time.

Syntax Examples

Running retrieval-augmented generation in a batch of 10:

```
mysql> CALL sys.ML_RAG_TABLE("demo_db.input_table.Input", "demo_db.output_table.Output", JSON_OBJECT("v
```

In this example, the routine performs RAG for 10 input queries stored in the `demo_db.input_table.Input` column, and creates a column of 10 rows `demo_db.output_table.Output` where it stores the generated outputs.

See Also

- Run Retrieval-Augmented Generation - Run Batch Queries
- Use Your Own Embeddings with Retrieval-Augmented Generation - Run Batch Queries
- Section 10.3.4, “ML_RAG”

10.3.6 HEATWAVE_CHAT

The `HEATWAVE_CHAT` routine automatically calls the `ML_RAG` routine which loads an LLM and runs a semantic search on the available vector stores by default. If the routine cannot find a vector store, then it calls the `ML_GENERATE` routine and uses information available in LLM training data, which is primarily information that is available in public data sources, to generate a response for the entered query.

This topic contains the following sections:

- HEATWAVE_CHAT Syntax
- @chat_options Parameters
- Syntax Examples
- See Also

HEATWAVE_CHAT Syntax

```
mysql> CALL sys.HEATWAVE_CHAT('QueryInNaturalLanguage');
```

The `HEATWAVE_CHAT` routine accepts one input parameter:

- *`QueryInNaturalLanguage`*: specifies the query in natural language.

For specifying additional chat parameter settings, the `HEATWAVE_CHAT` routine reserves a variable, `@chat_options`. When you run the routine, it also updates the `@chat_options` variable with any additional information that is used or collected by the routine to generate the response.

@chat_options Parameters

Following is a list of all the parameters that you can set in the `@chat_options` variable:

- **Input only**: you can set these parameters to control the chat behaviour. The routine cannot change the values of these parameters.
  - `schema_name`: specifies the name of a schema. If set, the routine searches for vector store tables in this schema. This parameter cannot be used in combination with the `tables` parameter. Default value is `NULL`
  - `report_progress`: specifies whether information such as routine progress detail is to be reported. Default value is `false`.
  - `skip_generate`: specifies whether response generation is skipped. If set to `true`, the routine does not generate a response. Default value is `false`.
  - `return_prompt`: specifies whether to return the prompt that was passed to the `ML_RAG` or `ML_GENERATE` routines. Default value is `false`.
  - `re_run`: if set to `true`, it indicates that the request is a re-run of the previous request. For example, a re-run of a query with some different parameters. The new query and response replaces the last entry stored in the `chat_history` parameter. Default value is `false`.
  - `include_document_uris`: limits the documents used for context retrieval by including only the specified document URIs. Default value is `NULL`.
  - `retrieve_top_k`: specifies the context size. The default value is the value of the `n_citations` parameter of the `ML_RAG` routine. Possible values are integer values between `0` and `100`.
  - `chat_query_id`: specifies the chat query ID to be printed with the `chat_history` in the GUI. This parameter is reserved for GUI use. By default, the routine generates random IDs.
  - `history_length`: specifies the maximum history length, which is the number of question and answers, to include in the chat history. The specified value must be greater than or equal to `0`. Default value is `3`.
  - `vector_store_columns`: optional parameter which specifies column names for finding relevant vector and embedding tables for context retrieval as key-value pairs in JSON format. If multiple tables contain columns with the same name and data type, then all such tables are used for context retrieval.

    It can include the following parameters:

    ```
    JSON_OBJECT('segment', 'SegmentColName', 'segment_embedding', 'EmbeddingColName'[, vsckeyvalue]...)
    vsckeyvalue:
    {
      'document_name', 'DocumentName'
      |'document_id', DocumentID
      |'metadata', 'Metadata'
      |'segment_number', SegmentNumber
    }
    ```

    - `segment`: specifies the name of the mandatory string column that contains the text segments. Default value is `segment`.
    - `segment_embedding`: specifies the name of the mandatory vector column that contains vector embeddings of the text segments. Default value is `segment_embedding`.

- `document_name`: specifies the name of the optional column that contains the document names. This column can be of any data type supported by MySQL HeatWave. Default value is `document_name`.
- `document_id`: specifies the name of the optional integer column that contains the document IDs. Default value is `document_id`.
- `metadata`: specifies the name of the optional JSON column that contains additional table metadata. Default value is `metadata`.
- `segment_number`: specifies the name of the optional integer column that contains the segment numbers. Default value is `segment_number`.

Default value is `{"segment": "segment", "segment_embedding": "segment_embedding", "document_id: "document_id", "segment_number": "segment_number", "metadata": "metadata"}`, which means that by default, the routine uses the default values of all column names to find relevant tables for context retrieval.

This parameter is available as of MySQL 9.2.1.

- `embed_model_id`: specifies the embedding model to use for embedding the input query. The routine uses vector store tables and embedding tables created using the same embedding model for context retrieval.

  As of MySQL 9.3.0, default value is `multilingual-e5-small`. In earlier versions of MySQL, default value is `all_minilm_l12_v2` if the output language is set to English and `multilingual-e5-small` if the output language is set to a language other than English.

  To view the list of available embedding models, see MySQL HeatWave In-Database Embedding Models and OCI Generative AI Service Embedding Models.

  As of MySQL 9.4.0, to use an OCI Generative AI service Dedicated AI Cluster model, specify the model endpoint OCID. For more information, see Creating an Endpoint in Generative AI.

  This parameter is available as of MySQL 9.2.1.

- `retrieval_options`: specifies optional context retrieval parameters as key-value pairs in JSON format. If a parameter value in `retrieval_options` is set to `auto`, the default value for that parameter is used.

  The `retrieval_options` parameters are available as of MySQL 9.3.1.

  It can include the following parameters:

```
JSON_OBJECT(retrievaloptkeyvalue[, retrievaloptkeyvalue]...)
retrievaloptkeyvalue:
{
  'max_distance', MaxDistance
  |'percentage_distance', PercentageDistance
  |'segment_overlap', SegmentOverlap
}
```

  - `max_distance`: specifies a maximum distance threshold for filtering out segments from context retrieval. Segments for which the distance from the input query exceeds the specified maximum distance threshold are excluded from content retrieval. This ensures that only the segments that

are closer to the input query are included during context retrieval. However, if no segments are found within the specified distance, the routine fails to run.

Note

If this parameter is set, the default value of the `n_citations` parameter is automatically updated to `10`.

Default value is `0.6` for all distance metrics.

Possible values are decimal values between `0` and `999999.9999`.

- `percentage_distance`: specifies what percentage of distance to the nearest segment is to be used to determine the maximum distance threshold for filtering out segments from context retrieval.

  Following is the formula used for calculating the maximum distance threshold:

  `MaximumDistanceThreshold` = *`DistanceOfInputQueryToNearestSegment`* + [(`percentage_distance` / `100`) * *`DistanceOfInputQueryToNearestSegment`*]

  Which means that the segments for which the distance to the input query exceeds the distance of the input query to the nearest segment by the specified percentage are filtered out from context retrieval.

  Note

  If this parameter is set, the default value of the `n_citations` parameter is automatically updated to `10`.

  Default value is `20` for all distance metrics.

  Possible values are decimal values between `0` and `999999.9999`.

  Note

  If both `max_distance` and `percentage_distance` are set, the smaller threshold value is considered for filtering out the segments.

- `segment_overlap`: specifies the number of additional segments adjacent to the nearest segments to the input query to be included in context retrieval. These additional segments provide more continuous context for the input query. Default value is `1`. Possible values are integer values between `0` and `5`.

- `speculative_decoding`: enables or disables speculative decoding when generating tokens with an LLM. If set to `true`, speculative decoding enables faster response token generation, which speeds up LLM text generation. Speculative decoding is supported for the `llama3.1-8b-instruct-v1` LLM, which uses `llama3.2-1B-instruct-v1` as the draft LLM. Default value is `true`.

  This parameter is available as of MySQL 9.3.2.

- **Input-output**: both you and the routine can change the values of these parameters.

  - `chat_history`: JSON array that represents the current chat history. Default value is `NULL`.

    Syntax for each object in the `chat_history` array is as follows:

    ```
    JSON_OBJECT(keyvalue[, keyvalue]...)
      keyvalue: {
      'user_message', 'Message'
      |'chat_bot_message', 'Message'
    ```

```
  |'chat_query_id', 'ID'
}
```

Each parameter value in the array holds the following keys and their values:

- `user_message`: message entered by the user.
- `chat_bot_message`: message generated by the chat bot.
- `chat_query_id`: a query ID.

- `tables`: JSON array that represents the following:
  - For providing input, represents the list of vector store schema or table names to consider for context retrieval.
  - As routine output, represents the list of discovered vector store tables loaded to MySQL HeatWave, if any. Otherwise, it holds the same values as input.

  Default value is `NULL`.

  Syntax for each object in the `tables` array is as follows:

```
JSON_OBJECT(keyvalue[, keyvalue] ...)
keyvalue:
{
  'schema_name', 'SchemaName'
  |'table_name', 'TableName'
}
```

  Each parameter values in the array holds the following keys and their values:

  - `schema_name`: name of the schema.
  - `table_name`: name of the vector store table.

- `task`: specifies the task performed by the LLM. Default value is `generation`. Possible value is `generation`.
- `model_options`: optional model parameters specified as key-value pairs in JSON format. These are the same options that are available in the `ML_GENERATE` routine, which alter the text-based response per the specified settings. As of MySQL 9.3.1, default value is `'{"model_id": "llama3.2-3b-instruct-v1"}'`. In earlier versions of MySQL, default value is `'{"model_id": "mistral-7b-instruct-v1"}'`.

- **Output only**: only the routine can set or change values of these parameters.
  - `info`: contains information messages such as routine progress information. Default value is `NULL`. This parameter is populated only if `report_progress` is set to `true`.
  - `error`: contains the error message if an error occurred. Default value is `NULL`.
  - `error_code`: contains the error code if an error occurred. Default value is `NULL`.
  - `prompt`: contains the prompt passed to the `ML_RAG` or `ML_GENERATE` routine. Default value is `NULL`. This parameter is populated only if `return_prompt` is set to `true`.
  - `documents`: contains the names of the documents as well as segments used as context by the LLM for response generation. Default value is `NULL`.
  - `request_completed`: set to `true` when a response is the last response message to a request. Default value is `NULL`.
  - `response`: contains the final response from the routine. Default value is `NULL`.

Syntax Examples

- Entering a natural-language query using the `HEATWAVE_CHAT` routine:

```
mysql> CALL sys.HEATWAVE_CHAT("What is Lakehouse?");
```

- Modifying chat parameters using the `@chat_options` variable:
  - Modifying a chat parameter, `tables`, to specify the vector store table to use for context retrieval in the next chat session:

    ```
    mysql> SET @chat_options = '{"tables": [{"table_name": "demo_embeddings", "schema_name": "demo_db"}]}'
    ```

    This example resets the chat session and uses the specified vector store table in the new chat session.
  - Modifying a chat parameter, `tables`, to specify the vector store table to use for context retrieval in the same chat session:

    ```
    mysql> SET @chat_options = JSON_SET(@chat_options,'$.tables', JSON_ARRAY(JSON_OBJECT("table_name", "der
    ```

    This example uses the specified vector store table in the ongoing chat session. It does not reset the chat session.
  - Modifying a chat parameter, `temperature`, without resetting the chat session:

    ```
    mysql> SET @chat_options = json_set(@chat_options, '$.model_options.temperature', 0.5);
    ```
- Viewing the chat parameters and session details:

```
mysql> SELECT JSON_PRETTY(@chat_options);
```

See Also

- Section 7.10, “Start a Conversational Chat”
- For more information about the output generated by this command, see Section 7.10.2, “View Chat Session Details”.

10.3.7 ML_EMBED_ROW

The `ML_EMBED_ROW` routine uses the specified embedding model to encode the specified text or query into a vector embedding. The routine returns a `VECTOR` that contains a numerical representation of the specified text.

This topic contains the following sections:

- ML_EMBED_ROW Syntax
- Syntax Examples
- See Also

ML_EMBED_ROW Syntax

```
mysql> SELECT sys.ML_EMBED_ROW('Text', options);

options: {JSON_OBJECT(keyvalue[, keyvalue] ...)|NULL}
keyvalue:
{
  'model_id', {'ModelID'}
  |'truncate', {true|false}
}
```

Following are `ML_EMBED_ROW` parameters:

- *`Text`*: specifies the text to encode.
- `options`: specifies optional parameters as key-value pairs in JSON format. It can include the following parameters:
  - `model_id`: specifies the embedding model to use for encoding the text. Possible values are:
    - `multilingual-e5-small`:
      - As of MySQL 9.3.0, used as the default embedding model.
      - As of MySQL 9.2.1, can be used for encoding text in any supported language.

        In earlier versions of MySQL, can be used for encoding text or files in supported languages other than English.
      - This embedding model is available as of MySQL 9.0.1-u1.
    - `all_minilm_l12_v2`:
      - In MySQL 9.2.2 and earlier versions, used as the default model embedding.
      - As of MySQL 9.2.1, can be used for encoding text in any supported language.

        In earlier versions of MySQL, can be used for encoding text in English only.
    - `cohere.embed-english-v3.0`:
      - As of MySQL 9.2.1, can be used for encoding text in any supported language.

        In earlier versions of MySQL, can be used for encoding text in English only.
      - This embedding model is available as of MySQL 9.0.1-u1.
    - `cohere.embed-multilingual-v3.0`:
      - As of MySQL 9.2.1, can be used for encoding text in any supported language.

        In earlier versions of MySQL, can be used for encoding text in supported languages other than English.
      - This embedding model is available as of MySQL 9.0.1-u1.

- As of MySQL 9.4.0, to use an OCI Generative AI service Dedicated AI Cluster model, specify the model endpoint OCID. For more information, see Creating an Endpoint in Generative AI.

To view the lists of available embedding models, see MySQL HeatWave In-Database Embedding Models and OCI Generative AI Service Embedding Models. To view the list of supported languages, see Languages.

- `truncate`: specifies whether to truncate inputs longer than the maximum token size. Default value is `true`.

Syntax Examples

- Embed an English query using the `all_minilm_l12_v2` embedding model, and store the generated embedding in the `@text_embedding` variable:

```
mysql> SELECT sys.ML_EMBED_ROW("What is artificial intelligence?", JSON_OBJECT("model_id", "all_minilm_l1
```

Print the embedding stored in the `@text_embedding` variable:

```
mysql> SELECT @text_embedding;
```

The output, which is a binary representation of the specified text, looks similar to the following:

```
+-----------------------------------------------------------------------------------------------------------
| @text_embedding
+-----------------------------------------------------------------------------------------------------------
| 0x97325DBC090E45BDD46399BCC4147C3C73DADFBCE23F41BDB10E1B3D1F5F7D3DE0BEFD3C9A362E3D4258383D643E7ABC4E073
+-----------------------------------------------------------------------------------------------------------
```

To convert the binary representation of this embedding into its string representation, use the `VECTOR_TO_STRING()` function:

```
mysql> SELECT VECTOR_TO_STRING(@text_embedding);
```

The output is similar to the following:

```
+-----------------------------------------------------------------------------------------------------------
| VECTOR_TO_STRING(@text_embedding)
+-----------------------------------------------------------------------------------------------------------
| [-1.35008e-02,-4.81091e-02,-1.87244e-02,1.53858e-02,-2.73258e-02,-4.71801e-02,3.78558e-02,6.18583e-02,3
+-----------------------------------------------------------------------------------------------------------
```

The string representation of the embedding consists of a list one or more comma-separated float values, encased in square brackets (`[ ]`). The values are expressed using decimal or scientific notation.

See Also

Section 7.8, “Generate Vector Embeddings”

10.3.8 ML_EMBED_TABLE

The `ML_EMBED_TABLE` routine runs multiple embedding generations in a batch, in parallel.

Note

In versions older than MySQL 9.2.1, to alter an existing table or create a new table, MySQL requires you to set the sql-require-primary-key system variable to `0`.

This routine is available as of MySQL 9.0.1-u1.

This topic contains the following sections:

- ML_EMBED_TABLE Syntax
- Syntax Examples
- See Also

To learn about the privileges you need to run this routine, see Section 7.3, “MySQL HeatWave GenAI Roles and Privileges”.

ML_EMBED_TABLE Syntax

```
mysql> CALL sys.ML_EMBED_TABLE('InputTableColumn', 'OutputTableColumn', options);

options: {JSON_OBJECT(keyvalue[, keyvalue] ...)|NULL}
keyvalue:
{
  'model_id', {'ModelID'}
  |'truncate', {true|false}
  |'batch_size', BatchSize
  |'details_column', 'ErrorDetailsColumnName'
}
```

Following are `ML_EMBED_TABLE` parameters:

- *`InputTableColumn`*: specifies the names of the input database, table, and column that contains the text to encode. The *`InputTableColumn`* is specified in the following format: *`DBName`*.*`TableName`*.*`ColumnName`*.
  - The specified input table can be an internal or external table.
  - The specified input table must already exist, must not be empty, and must have a primary key.
  - The input column must already exist and must contain `text` or `varchar` values.
  - The input column must not be a part of the primary key and must not have `NULL` values or empty strings.
  - There must be no backticks used in the *`DBName`*, *`TableName`*, or *`ColumnName`* and there must be no period used in the *`DBName`* or *`TableName`*.
- *`OutputTableColumn`*: specifies the names of the database, table, and column where the generated embeddings are stored. The *`OutputTableColumn`* is specified in the following format: *`DBName`*.*`TableName`*.*`ColumnName`*.
  - The specified output table must be an internal table.
  - If the specified output table already exists, then it must be the same as the input table. And, the specified output column must not already exist in the input table. A new `VECTOR` column is added to the table. External tables are read only. So if input table is an external table, then it cannot be used to store the output.
  - If the specified output table doesn't exist, then a new table is created. The new output table has key columns which contains the same primary key values as the input table and a `VECTOR` column that stores the generated embeddings.
  - There must be no backticks used in the *`DBName`*, *`TableName`*, or *`ColumnName`* and there must be no period used in the *`DBName`* or *`TableName`*.
- `options`: specifies optional parameters as key-value pairs in JSON format. It can include the following parameters:
  - `model_id`: specifies the embedding model to use for encoding the text. Possible values are:
    - `multilingual-e5-small`:

    - As of MySQL 9.3.0, used as the default embedding model.
    - As of MySQL 9.2.1, can be used for encoding text in any supported language.

      In earlier versions of MySQL, can be used for encoding text or files in supported languages other than English.
    - This embedding model is available as of MySQL 9.0.1-u1.
  - `all_minilm_l12_v2`:
    - In MySQL 9.2.2 and earlier versions, used as the default model embedding.
    - As of MySQL 9.2.1, can be used for encoding text in any supported language.

      In earlier versions of MySQL, can be used for encoding text in English only.
  - `cohere.embed-english-v3.0`:
    - As of MySQL 9.2.1, can be used for encoding text in any supported language.

      In earlier versions of MySQL, can be used for encoding text in English only.
    - This embedding model is available as of MySQL 9.0.1-u1.
  - `cohere.embed-multilingual-v3.0`:
    - As of MySQL 9.2.1, can be used for encoding text in any supported language.

      In earlier versions of MySQL, can be used for encoding text in supported languages other than English.
    - This embedding model is available as of MySQL 9.0.1-u1.
  - As of MySQL 9.4.0, to use an OCI Generative AI service Dedicated AI Cluster model, specify the model endpoint OCID. For more information, see Creating an Endpoint in Generative AI.

  To view the lists of available embedding models, see MySQL HeatWave In-Database Embedding Models and OCI Generative AI Service Embedding Models. To view the list of supported languages, see Languages.
- `truncate`: specifies whether to truncate inputs longer than the maximum token size. Default value is `true`.
- `batch_size`: specifies the batch size for the routine. This parameter is supported for internal tables only. Default value is `1000`. Possible values are integer values between `1` and `1000`.
- `details_column`: specifies a name for the output table column that is created for adding details of errors encountered for rows that aren't processed successfully by the routine. Ensure that a column by the specified name does not already exist in the table. Default value is `details`.

  This parameter is available as of MySQL 9.3.0.

Syntax Examples

Generate embeddings for text stored in `demo_db.input_table.Input` using the `all_minilm_l12_v2` embedding model, and save the generated embeddings in the output table `demo_db.output_table.Output`:

```
mysql> CALL sys.ML_EMBED_TABLE("demo_db.input_table.Input", "demo_db.output_table.Output", JSON_OBJECT("mod
```

See Also

Generate Vector Embeddings - Run Batch Queries

10.3.9 NL_SQL

Generates SQL queries using natural-language statements. The routine also runs the generated SQL statement and displays the result set. You can use this routine for generating and running SQL queries only for databases and tables that you have access to.

Note

This routine can generate and run `SELECT` statements only.

The LLM-generated SQL statements might contain syntax errors. The routine automatically detects these errors, and retries the SQL generation until a syntactically valid SQL statement is generated, with a maximum of 3 generation attempts.

This routine is available as of MySQL 9.4.1.

This topic contains the following sections:

- NL_SQL Syntax
- Syntax Examples
- See Also

NL_SQL Syntax

```
mysql> CALL sys.NL_SQL("NaturalLanguageStatement", @output, options);

options: {JSON_OBJECT(keyvalue[, keyvalue]...)|NULL}
keyvalue:
{
  'execute', {true|false}
  | 'schemas', JSON_ARRAY('DBName'[, 'DBName'] ...)
  | 'tables', JSON_ARRAY(TableJSON[, TableJSON] ...)
  | 'model_id', 'ModelID'
  | 'verbose', {0|1|2}
  | 'include_comments', {true|false}
  | 'use_retry', {true|false}
}
```

Following are `NL_SQL` parameters:

- *`NaturalLanguageQuery`*: natural-language query pertaining to your data available in MySQL HeatWave that you want to convert to an SQL query.
- `@output`: output parameter that includes the list of tables and databases considered for generating the SQL query, Model ID of the Large Language Model (LLM) used for generating the query, the generated SQL query, and whether the generated SQL query is valid.
- *`options`*: specifies optional parameters as key-value pairs in JSON format. It can include the following parameters:
  - `execute`: specifies whether the procedure automatically runs the generated SQL statement. Default value is `true`.
  - `schemas`: specifies the databases to consider for generating and running SQL queries. You can specify up to five databases. By default, databases that the routine finds most relevant to the entered natural-language statement are considered.

- `tables`: specifies the tables to consider for generating and running SQL queries in JSON format. You can specify up to 50 tables. By default, tables that the routine finds most relevant to the entered natural-language statement are considered.

```
TableJSON: JSON_OBJECT('schema_name', 'DBName', 'table_name', 'TableName')
```

Note

You can either use the `schemas` option to specify the databases to consider or the `tables` option to specify the tables to consider. If you set both these options, the routine fails.

- `model_id`: specifies the LLM to use for generating the SQL query. Default value is `meta.llama-3.3-70b-instruct`. Possible values are `meta.llama-3.3-70b-instruct`, `llama3.1-8b-instruct-v1`, `llama3.2-3b-instruct-v1`, and as of MySQL 9.5.0, you can select one of the OCI Generative AI Service LLMs available in MySQL HeatWave, including the Gemini and OpenAI models that are supported as of MySQL 9.7.0.

- `verbose`: specifies whether to print an output. Possible values are: `0`: prints nothing, `1`: prints the generated SQL statement, and `2`: prints debugging information. Default value is `1`.

- `include_comments`: specifies whether comments are to be included during metadata collection for columns or tables. Default value is `true`.

- `use_retry`: specifies whether generation retries for syntactically invalid SQL statements can be attempted. Default value is `true`.

Syntax Examples

The examples in this topic uses a sample database, `airport`. To learn how to download and then load `airportdb` in MySQL HeatWave Cluster, see Section 3.2, “Quickstart: AirportDB Analytics”.

- Following example specifies the database to consider for the SQL query:

```
mysql> CALL sys.NL_SQL("How many flights are there in total?",@output, JSON_OBJECT('schemas',JSON_ARRAY('
+-----------------------------------------------------+
| Executing generated SQL statement...                |
+-----------------------------------------------------+
| SELECT COUNT(`flight_id`) FROM `airportdb`.`flight` |
+-----------------------------------------------------+
1 row in set (1 min 39.4060 sec)

+--------------------+
| COUNT(`flight_id`) |
+--------------------+
|             462553 |
+--------------------+
```

View the value stored in the variable `@output`:

```
mysql> SELECT JSON_PRETTY(@output);
+-----------------------------------------------------------------------------------------------------------------
| JSON_PRETTY(@output)
+-----------------------------------------------------------------------------------------------------------------
| {
  "tables": [
    "airportdb.airline",
    "airportdb.airplane",
    "airportdb.airplane_type",
    "airportdb.airport",
    "airportdb.airport_geo",
    "airportdb.airport_reachable",
    "airportdb.booking",
    "airportdb.employee",
    "airportdb.flight",
```

```
    "airportdb.flight_log",
    "airportdb.flightschedule",
    "airportdb.passenger",
    "airportdb.passengerdetails",
    "airportdb.weatherdata"
  ],
  "license": "Your use of this Llama model is subject to the Llama 3.1 Community License Agreement av
  "schemas": [
    "airportdb"
  ],
  "model_id": "llama3.1-8b-instruct-v1",
  "sql_query": "SELECT COUNT(`flight_id`) FROM `airportdb`.`flight`",
  "is_sql_valid": 1
} |
+-------------------------------------------------------------------------------------------------------
```

- Following example specifies the tables to consider for the SQL query:

```
mysql> CALL sys.NL_SQL("List five airlines that have the highest number of Airbus A330 aircrafts with
  @output, JSON_OBJECT('tables',
  JSON_ARRAY(JSON_OBJECT("schema_name","airportdb","table_name","airlines"),
  JSON_OBJECT("schema_name","airportdb","table_name","airplane"),
  JSON_OBJECT("schema_name","airportdb","table_name","airplane_type")),
  'model_id','llama3.1-8b-instruct-v1'));
+-------------------------------------------------------------------------------------------------------
| Executing generated SQL statement...
+-------------------------------------------------------------------------------------------------------
| SELECT `T1`.`airline_id`, COUNT(*) FROM `airportdb`.`airplane` AS `T1` JOIN `airportdb`.`airplane_t
+-------------------------------------------------------------------------------------------------------
1 row in set (1 min 40.3538 sec)

+------------+----------+
| airline_id | COUNT(*) |
+------------+----------+
|         78 |       13 |
|         55 |       11 |
|         46 |       11 |
|         73 |       10 |
|         33 |       10 |
+------------+----------+
```

View the value stored in the variable `@output`:

```
mysql> SELECT JSON_PRETTY(@output);
+-------------------------------------------------------------------------------------------------------
| JSON_PRETTY(@output)
+-------------------------------------------------------------------------------------------------------
| {
  "tables": [
    "airportdb.airplane",
    "airportdb.airplane_type"
  ],
  "license": "Your use of this Llama model is subject to the Llama 3.1 Community License Agreement av
  "schemas": [
    "airportdb"
  ],
  "model_id": "llama3.1-8b-instruct-v1",
  "sql_query": "SELECT `T1`.`airline_id`, COUNT(*) FROM `airportdb`.`airplane` AS `T1` JOIN `airportd
  "is_sql_valid": 1
} |
+-------------------------------------------------------------------------------------------------------
```

- Following example does not specify any additional options:

```
mysql> CALL sys.NL_SQL('Find five most frequent weekly flights from United States to United Kingdom.
+-------------------------------------------------------------------------------------------------------
| Executing generated SQL statement...
+-------------------------------------------------------------------------------------------------------
| SELECT `T1`.`flightno`, `T2`.`name` AS `departure_airport`, `T3`.`name` AS `arrival_airport`, SUM(`
+-------------------------------------------------------------------------------------------------------
1 row in set (27.2586 sec)
```

```
+----------+---------------------+-----------------+-----------+
| flightno | departure_airport   | arrival_airport | frequency |
+----------+---------------------+-----------------+-----------+
| UZ6906   | SANTA YNEZ          | BRISTOL         |         6 |
| ZI7472   | ENID WOODRING REGL  | ST ATHAN AB     |         6 |
| BR4282   | KINGS LAND O'LAKES  | SIBSON          |         6 |
| GR1230   | TIPTON              | WADDINGTON AB   |         6 |
| AZ4913   | BIG PINEY-MARBLETON | LUTON           |         5 |
+----------+---------------------+-----------------+-----------+
5 rows in set (27.2586 sec)
```

View the value stored in the variable `@output`:

```
mysql> SELECT JSON_PRETTY(@output);
+-----------------------------------------------------------------------------------------------------
| JSON_PRETTY(@output)
+-----------------------------------------------------------------------------------------------------
| {
  "tables": [
    "airportdb.flightschedule",
    "airportdb.flight",
    "airportdb.airport_reachable",
    "airportdb.booking",
    "airportdb.airplane",
    "airportdb.flight_log",
    "airportdb.airport_geo",
    "airportdb.passengerdetails",
    "airportdb.weatherdata",
    "airportdb.airline",
    "airportdb.passenger",
    "airportdb.airplane_type",
    "airportdb.airport",
    "airportdb.employee"
  ],
  "license": "Your use of this Llama model is subject to the Llama 3.1 Community License Agreement availa
  "schemas": [
    "airportdb"
  ],
  "model_id": "meta.llama-3.3-70b-instruct",
  "sql_query": "SELECT `T1`.`flightno`, `T2`.`name` AS `departure_airport`, `T3`.`name` AS `arrival_airpo
  "is_sql_valid": 1
} |
+-----------------------------------------------------------------------------------------------------
```

See Also

Section 7.11, “Generate SQL Queries From Natural-Language Statements”

10.3.10 ML_RETRIEVE_SCHEMA_METADATA

Retrieves the most relevant tables to a given natural-language statement and ranks them in the order of their relevance.

The output generated by the routine can be used to enhance large language model (LLM) prompts, `NL_SQL` workflows, and retrieval-augmented generation (RAG)-based analytics.

This routine is available as of MySQL 9.5.2.

ML_RETRIEVE_SCHEMA_METADATA Syntax

```
mysql> CALL sys.ML_RETRIEVE_SCHEMA_METADATA("NaturalLanguageStatement", @output, options);

options: {JSON_OBJECT(keyvalue[, keyvalue]...)|NULL}
keyvalue:
{
    'schemas', JSON_ARRAY('DBName'[, 'DBName'] ...)
  | 'tables', JSON_ARRAY(TableJSON[, TableJSON] ...)
  | 'include_comments', {true|false}
}
```

Following are `ML_RETRIEVE_SCHEMA_METADATA` parameters:

- *`NaturalLanguageQuery`*: natural-language query that you want to use to find relevant tables.
- `@output`: output parameter that includes the most relevant tables and databases. This output consists of a concise set of abridged `CREATE TABLE` statements for the tables that are most relevant to a natural-language statement, ranked in order of relevance.
- *`options`*: specifies optional parameters as key-value pairs in JSON format. It can include the following parameters:
  - `schemas`: specifies the databases to consider for retrieving most relevant tables. You can specify up to 128 databases. By default, all available databases are considered for retrieving the most relevant tables.
  - `tables`: specifies the tables to consider for retrieving most relevant tables. You can specify up to 128 tables. By default, all tables in all available databases are considered for retrieving the most relevant tables.

    ```
    TableJSON: JSON_OBJECT('schema_name', 'DBName', 'table_name', 'TableName')
    ```

    Note

    You can either use the `schemas` option to specify the databases to consider or the `tables` option to specify the tables to consider. If you set both these options, the routine fails.

  - `include_comments`: specifies whether table and column comments are used for retrieving the most relevant tables and also whether the comments are included with the generated output. Default value is `true`.

Syntax Examples

- Following example specifies only a natural-language statement for retrieving the most relevant tables:

```
mysql> CALL sys.ML_RETRIEVE_SCHEMA_METADATA("How many flights are there in total?", @output, NULL);
```

View the value stored in the variable `@output`:

```
mysql> SELECT @output;
+-----------------------------------------------------------------------------------------------------------------
| @output
+-----------------------------------------------------------------------------------------------------------------
| CREATE TABLE `airportdb`.`flightschedule`(
  `flightno` char,
  `from` smallint,
  `to` smallint,
  `departure` time,
  `arrival` time,
  `airline_id` smallint,
  `monday` tinyint,
  `tuesday` tinyint,
  `wednesday` tinyint,
  `thursday` tinyint,
  `friday` tinyint,
  `saturday` tinyint,
  `sunday` tinyint,
  FOREIGN KEY (`airline_id`) REFERENCES `airportdb`.`airline`(`airline_id`),
  FOREIGN KEY (`to`) REFERENCES `airportdb`.`airport`(`airport_id`),
  FOREIGN KEY (`from`) REFERENCES `airportdb`.`airport`(`airport_id`)
) COMMENT 'Flughafen DB by Stefan Pröll, Eva Zangerle, Wolfgang Gassler is licensed under CC BY 4.0.

CREATE TABLE `airportdb`.`flight`(
  `flight_id` int,
  `flightno` char,
```

```
  `from` smallint,
  `to` smallint,
  `departure` datetime,
  `arrival` datetime,
  `airline_id` smallint,
  `airplane_id` int,
  FOREIGN KEY (`flightno`) REFERENCES `airportdb`.`flightschedule`(`flightno`),
  FOREIGN KEY (`airplane_id`) REFERENCES `airportdb`.`airplane`(`airplane_id`),
  FOREIGN KEY (`airline_id`) REFERENCES `airportdb`.`airline`(`airline_id`),
  FOREIGN KEY (`to`) REFERENCES `airportdb`.`airport`(`airport_id`),
  FOREIGN KEY (`from`) REFERENCES `airportdb`.`airport`(`airport_id`)
) COMMENT 'Flughafen DB by Stefan Pröll, Eva Zangerle, Wolfgang Gassler is licensed under CC BY 4.0. To v

CREATE TABLE `airportdb`.`airplane`(
  `airplane_id` int,
  `capacity` mediumint,
  `type_id` int,
  `airline_id` int,
  FOREIGN KEY (`type_id`) REFERENCES `airportdb`.`airplane_type`(`type_id`)
) COMMENT 'Flughafen DB by Stefan Pröll, Eva Zangerle, Wolfgang Gassler is licensed under CC BY 4.0. To v

... |
+-----------------------------------------------------------------------------------------------------------
1 row in set (0.2438 sec)
```

- Following example specifies the database to use for retrieving most relevant tables for the given natural-language statement:

```
mysql> CALL sys.ML_RETRIEVE_SCHEMA_METADATA("How many threads are active?", @output, JSON_OBJECT('schemas
```

View the value stored in the variable `@output`:

```
mysql> SELECT @output;
+-----------------------------------------------------------------------------------------------------------
| @output
+-----------------------------------------------------------------------------------------------------------
| CREATE TABLE `performance_schema`.`status_by_thread`(
  `THREAD_ID` bigint,
  `VARIABLE_NAME` varchar,
  `VARIABLE_VALUE` varchar
);

CREATE TABLE `performance_schema`.`variables_by_thread`(
  `THREAD_ID` bigint,
  `VARIABLE_NAME` varchar,
  `VARIABLE_VALUE` varchar
);

CREATE TABLE `performance_schema`.`user_variables_by_thread`(
  `THREAD_ID` bigint,
  `VARIABLE_NAME` varchar,
  `VARIABLE_VALUE` longblob
);

... |
+-----------------------------------------------------------------------------------------------------------
1 row in set (0.4796 sec)
```

Chapter 11 Troubleshoot

Table of Contents

11.1 Issues and Errors 671
11.1.1 MySQL HeatWave Issues 671
11.1.2 MySQL HeatWave Errors 674
11.1.3 MySQL HeatWave AutoML Error Messages 699
11.1.4 MySQL HeatWave GenAI Issues 719
11.1.5 MySQL HeatWave Lakehouse Error Messages 720
11.2 Limitations 723
11.2.1 MySQL HeatWave Limitations 723
11.2.2 MySQL HeatWave Lakehouse Limitations 731
11.2.3 MySQL HeatWave AutoML Limitations 737

The sections in this chapter describe how to troubleshoot MySQL HeatWave errors and lists some of the limitations of different features of MySQL HeatWave.

11.1 Issues and Errors

11.1.1 MySQL HeatWave Issues

- *Problem:* Queries are not offloaded.
  - *Solution A:* Your query contains an unsupported predicate, function, operator, or has encountered some other limitation. See Section 5.4.1, “Requirements for Running Queries”.
  - *Solution B:* Query execution time is less than the query cost threshold.

    MySQL HeatWave is designed for fast execution of large analytic queries. Smaller, simpler queries, such as those that use indexes for quick lookups, often execute faster on the MySQL DB System. To avoid offloading inexpensive queries to MySQL HeatWave, the optimizer uses a query cost estimate threshold value. Only queries that exceed the threshold value on the MySQL DB System are considered for offload.

    The query cost threshold unit value is the same unit value used by the MySQL optimizer for query cost estimates. The threshold is 100000.00000. The ratio between a query cost estimate value and the actual time required to execute a query depends on the type of query, the type of hardware, and MySQL DB System configuration.

    To determine the cost of a query on the MySQL DB System:

    1. Disable `use_secondary_engine` to force MySQL DB System execution.

       ```
       mysql> SET SESSION use_secondary_engine=OFF;
       ```

    2. Run the query using `EXPLAIN`.

       ```
       mysql> EXPLAIN select_query;
       ```

    3. Query the `Last_query_cost` status variable. If the value is less than 100000.00000, the query cannot be offloaded.

       ```
       mysql> SHOW STATUS LIKE 'Last_query_cost';
       ```

  - *Solution C:* The table you are querying is not loaded. You can check the load status of a table in MySQL HeatWave by querying `LOAD_STATUS` data from MySQL HeatWave Performance Schema tables. For example:

```
mysql> USE performance_schema;
mysql> SELECT NAME, LOAD_STATUS FROM rpd_tables,rpd_table_id
          WHERE rpd_tables.ID = rpd_table_id.ID;
+-----------------------------+--------------------+
| NAME                        | LOAD_STATUS        |
+-----------------------------+--------------------+
| tpch.supplier               | AVAIL_RPDGSTABSTATE |
| tpch.partsupp               | AVAIL_RPDGSTABSTATE |
| tpch.orders                 | AVAIL_RPDGSTABSTATE |
| tpch.lineitem               | AVAIL_RPDGSTABSTATE |
| tpch.customer               | AVAIL_RPDGSTABSTATE |
| tpch.nation                 | AVAIL_RPDGSTABSTATE |
| tpch.region                 | AVAIL_RPDGSTABSTATE |
| tpch.part                   | AVAIL_RPDGSTABSTATE |
+-----------------------------+--------------------+
```

For information about load statuses, see Section 8.1.10, “The rpd_tables Table”.

Alternatively, run the following statement:

```
mysql> ALTER TABLE tbl_name SECONDARY_LOAD;
```

The following error is reported if the table is already loaded:

```
mysql> ERROR 13331 (HY000): Table is already loaded.
```

- *Solution D:* The MySQL HeatWave Cluster has failed. To determine the status of the MySQL HeatWave Cluster, run the following statement:

```
mysql> SHOW GLOBAL STATUS LIKE 'rapid_plugin_bootstrapped';
+---------------------------------+------------+
| Variable_name                   | Value      |
+---------------------------------+------------+
| rapid_plugin_bootstrapped       | YES        |
+---------------------------------+------------+
```

See Chapter 9, *System and Status Variables* for `rapid_plugin_bootstrapped` status values.

If the MySQL HeatWave Cluster has failed, restart it in the MySQL HeatWave Console and reload the data if necessary. The MySQL HeatWave recovery mechanism should reload the data automatically.

- *Problem:* You have encountered an out-of-memory error when executing a query.

  Solution: MySQL HeatWave optimizes for network usage rather than memory. If you encounter out of memory errors when running a query, try running the query with the `MIN_MEM_CONSUMPTION` strategy by setting `rapid_execution_strategy` before executing the query:

```
mysql> SET SESSION rapid_execution_strategy = MIN_MEM_CONSUMPTION;
```

  Also consider checking the size of your data by performing a cluster size estimate. If your data has grown substantially, you may require additional MySQL HeatWave nodes.

  - For MySQL HeatWave on OCI, see Generating a Node Count Estimate in the *MySQL HeatWave on OCI Service Guide*.
  - For MySQL HeatWave on AWS, see Estimating Cluster Size with MySQL HeatWave Autopilot in the *MySQL HeatWave on AWS Service Guide*.
  - For MySQL HeatWave for Azure, see Provisioning MySQL HeatWave Nodes in the *MySQL HeatWave for Azure Service Guide*.

  Avoid or rewrite queries that produce a Cartesian product. For more information, see Diagnosing and Resolving Out-of-Memory (OOM) Errors in MySQL Query Execution.

- *Problem:* A table load operation fails with “ERROR HY000: Error while running parallel scan.”

  Solution: A `TEXT` type column value larger than 65535 bytes is rejected during `SECONDARY_LOAD` operations. Reduce the size of the `TEXT` type column value to less than 65535 bytes or exclude the column before loading the table. See Section 4.2.6.1, “Exclude Table Columns”.

- *Problem:* Change propagation fails with the following error: “Blob/text value of *n* bytes was encountered during change propagation but RAPID supports text values only up to 65535 bytes.”

  Solution: `TEXT` type values larger than 65335 bytes are rejected during change propagation. Reduce the size of `TEXT` type values to less than 65535 bytes. Should you encounter this error, check the change propagation status for the affected table. If change propagation is disabled, reload the table. See Section 4.2.7, “About Change Propagation”.

- *Problem:* A warning was encountered when running Auto Parallel Load.

  Solution: When Auto Parallel Load encounters an issue that produces a warning, it automatically switches to `dryrun` mode to prevent further problems. In this case, the load statements generated by the Auto Parallel Load utility can still be obtained using the SQL statement provided in the utility output, but avoid those load statements or use them with caution, as they may be problematic.

  - If a warning message indicates that the MySQL HeatWave Cluster or service is not active or online, this means that the load cannot start because a MySQL HeatWave Cluster is not attached to the MySQL DB System or is not active. In this case, provision and enable a MySQL HeatWave Cluster and run Auto Parallel Load again.

  - If a warning message indicates that MySQL host memory is insufficient to load all of the tables, the estimated dictionary size for dictionary-encoded columns may be too large for MySQL host memory. Try changing column encodings to `VARLEN` to free space in MySQL host memory.

  - If a warning message indicates that MySQL HeatWave Cluster memory is insufficient to load all of the tables, the estimated table size is too large for MySQL HeatWave Cluster memory. Try excluding certain schemas or tables from the load operation or increase the size of the cluster.

  - If a warning message indicates that a concurrent table load is in progress, this means that another client session is currently loading tables into MySQL HeatWave. While the concurrent load operation is in progress, the accuracy of Auto Parallel Load estimates cannot be guaranteed. Wait until the concurrent load operation finishes before running Auto Parallel Load.

- *Problem:* During retrieval of the generated Auto Parallel Load or Advisor DDL statements, an error message indicates that the `heatwave_autopilot_report` table or the `heatwave_advisor_report` table or the `heatwave_load_report` table does not exist. For example:

```
mysql> SELECT log->>"$.sql" AS "SQL Script"
          FROM sys.heatwave_autopilot_report
          WHERE type = "sql"
          ORDER BY id;
ERROR 1146 (42S02): Table 'sys.heatwave_autopilot_report' does not exist

mysql> SELECT log->>"$.sql" AS "SQL Script"
          FROM sys.heatwave_advisor_report
          WHERE type = "sql"
          ORDER BY id;
ERROR 1146 (42S02): Table 'sys.heatwave_advisor_report' does not exist


mysql> SELECT log->>"$.sql" AS "SQL Script"
          FROM sys.heatwave_load_report
          WHERE type = "sql"
          ORDER BY id;
```

```
ERROR 1146 (42S02): Table 'sys.heatwave_load_report' does not exist
```

Solution: This error can occur when querying a report table from a different session. Query the report table using the same session that issued the Auto Parallel Load or Advisor `CALL` statement. This error also occurs if the session used to call Auto Parallel Load or Advisor has timed out or was terminated. In this case, run Auto Load or Advisor again before querying the report table.

- *Problem:* `ETAG()` function result mismatch between MySQL and MySQL HeatWave

  Solution: It is recommended to run the `ETAG` function on the actual inserted data rather than the `SELECT` output of a query especially when dealing with `BLOB` columns.

  `BLOB` values are stored as `BINARY` values in database storage, but they are represented in Base64 encoded format when projected as a `SELECT` query output. The generated ETAG value can be different when run with a `BLOB` input as compared to the Base64 formatted string that a `SELECT` query outputs.

11.1.2 MySQL HeatWave Errors

The following list contains all current MySQL HeatWave errors in numerical order. The entry for each error contains the error number, name (symbol), description, and suggested workaround for the condition that gave rise to the error.

- Error number: `1001`; Symbol: `HW_ER_CHANGE_PROP`

  Message: Errors detected while propagating changes - %s.

  Description: The change propagation operation was not completed successfully.

  Workaround: Retry the operation. If the issue persists, reload the table and retry the query with use_secondary_engine session variable set to FORCED.

- Error number: `1002`; Symbol: `HW_ER_INCOMPATIBLE_FUNC_ARGS`

  Message: Incompatible arguments, ['%s', '%s'], for function, [%s].

  Description: This function does not support arguments of the specified data types.

  Workaround: Modify the function arguments to use supported data types. If an error occurs with an ENUM argument, use a different data type. Refer to HeatWave documentation for valid function arguments.

- Error number: `1003`; Symbol: `HW_ER_QUERY_SCN_TOO_OLD`

  Message: Query cannot be offloaded to HeatWave because the resized cluster does not have the required data version.

  Description: The cluster lacks the necessary data version for query offloading.

  Workaround: Retry the query. If the issue persists, try rewriting the query to work around the error.

- Error number: `1004`; Symbol: `HW_ER_QUERY_CANCELLED`

  Message: Query was canceled during HeatWave query optimization.

  Description: The query was interrupted before completion.

  Workaround: Retry the query. If the issue persists, try rewriting the query to work around the error.

- Error number: `1005`; Symbol: `HW_ER_UNSUPPORTED_QUERY_OPT`

  Message: Currently unsupported HeatWave query optimization scenario.

Description: This error occurs when the query compiler encounters an unsupported condition or clause, preventing it from generating a valid query execution plan.

Workaround: Rewrite the query to ensure HeatWave compatibility. For example, modify filter conditions or adjust projected columns.

- Error number: `1006`; Symbol: `HW_ER_CLUSTER_OFFLINE`

  Message: Network error - HeatWave cluster is not online.

  Description: HeatWave cluster is unavailable.

  Workaround: Start the HeatWave cluster and ensure it is online.

- Error number: `1007`; Symbol: `HW_ER_INTERNAL`

  Message: HeatWave internal error - %s.

  Description: This error happens due to internal issues in HeatWave during logical and physical transformation, though it is not limited to this scenario.

  Workaround: Retry the query. If the issue persists, try rewriting the query to work around the error.

- Error number: `1008`; Symbol: `HW_ER_NOT_READY_FOR_REQUESTS`

  Message: HeatWave not in a ready state to handle client requests - %s.

  Description: HeatWave plugin is not in the correct state.

  Workaround: Verify that HeatWave plugin state is in the correct, ready state.

- Error number: `1009`; Symbol: `HW_ER_QUERY_EXEC`

  Message: Error encountered during HeatWave query execution - %s.

  Description: This error indicates issues such as memory limits, data type range violations, or interruption of the query by the user.

  Workaround: Retry the query. If the issue persists, try rewriting the query to work around the error.

- Error number: `1010`; Symbol: `HW_ER_SUBQUERY_TRANSFORM`

  Message: The subquery, [%s], cannot be transformed into a semi-join, anti-join or derived tables.

  Description: A problem occured during the logical transformation of the query.

  Workaround: Add join conditions for each join, or consider rewriting the query.

- Error number: `1011`; Symbol: `HW_ER_UNSUPPORTED_TYPE_IN_WHERE`

  Message: %s in WHERE clause, [%s] is not supported.

  Description: The specified data type is not supported in the WHERE clause.

  Workaround: Consider using a non-string or non-BIT type column in the WHERE clause condition.

- Error number: 1012; Symbol: HW_ER_UNSUPPORTED_ARGS_IN_COND

  Message: Unsupported arguments in condition, [%s] - %s.

  Description: The specified functionality is not supported for these arguments.

  Workaround: Please modify the expression to ensure the result is not a constant value. Alternatively, adjust the conditional expression by adding or removing arguments

- Error number: 1013; Symbol: HW_ER_UNSUPPORTED_COND_TYPE

  Message: The condition type, [%s], is not supported.

  Description: The specified condition type is not supported in HeatWave.

  Workaround: Use a supported condition type.

- Error number: 1014; Symbol: HW_ER_UNSUPPORTED_FUNC_TYPE

  Message: The function, [%s], is not supported.

  Description: The specified function is not available in HeatWave.

  Workaround: Use a supported function, or explore combining supported functions with other supported functions and/or expressions.

- Error number: 1015; Symbol: HW_ER_UNSUPPORTED_DECIMAL_PRECISION_FOR_FUNC

  Message: For the function, [%s], DECIMAL values with precision greater than 18 are not supported.

  Description: The precision of the DECIMAL datatype may exceed HeatWave's supported limits for arithmetic operations.

  Workaround: Use a datatype with precision not greater than 18.

- Error number: 1016; Symbol: HW_ER_UNSUPPORTED_ARG_FOR_FUNC

  Message: Unsupported argument, [%s], for function, [%s] - %s.

  Description: HeatWave does not support the specified argument type for this function.

  Workaround: Change the argument data type. Refer to HeatWave documentation for valid function arguments and data types.

- Error number: 1017; Symbol: HW_ER_UNSUPPORTED_AGGREGATE_FUNC_WITH_VARCHAR_OR_BIT_INPUT

  Message: The aggregate function, [%s], with VARCHAR or BIT input, [%s], is not supported.

  Description: HeatWave does not support this aggregation function with VARCHAR or BIT input.

  Workaround: Change the argument data type. Refer to HeatWave documentation for valid function arguments and data types.

- Error number: 1018; Symbol: HW_ER_UNSUPPORTED_OUTPUT_TYPE_IN_FUNC

  Message: The output data type, [%s], for the function, [%s], is not supported.

  Description: HeatWave does not support this function with the specified output data type.

  Workaround: Try casting the output to a different supported data type.

- Error number: 1019; Symbol: HW_ER_UNSUPPORTED_AGGREGATE_FUNC

Message: Aggregate function, [%s], is not supported.

Description: HeatWave does not support this aggregation function.

Workaround: To achieve the same result as the aggregate function, rewrite the query using a subquery or use supported aggregation functions.

- Error number: 1020; Symbol: HW_ER_UNSUPPORTED_TYPE_IN_PROJ

  Message: Unsupported projection item, [%s].

  Description: HeatWave is unable to project this item.

  Workaround: Use the BIT column in an expression or cast it to a different data type.

- Error number: 1021; Symbol: HW_ER_UNSUPPORTED_LATERAL_DERIVED_TABLE_VIEWS

  Message: LATERAL views or derived tables are not supported - while processing %s.

  Description: LATERAL views or derived tables are not supported.

  Workaround: Consider using non-LATERAL table.

- Error number: 1022; Symbol: HW_ER_UNSUPPORTED_CALC_FOUND_ROWS

  Message: SQL_CALC_FOUND_ROWS and FOUND_ROWS are not supported.

  Description: SQL_CALC_FOUND_ROWS and FOUND_ROWS deprecated.

  Workaround: Refer to HeatWave documentation for SQL_CALC_FOUND_ROWS or FOUND_ROWS alternatives.

- Error number: 1023; Symbol: HW_ER_UNSUPPORTED_TYPE_IN_GBY

  Message: Expression, [%s], in GROUP BY key list is not supported.

  Description: The specified data type is not supported in the GROUP BY key list.

  Workaround: Modify the expression so that the resulting data type is not BIT. Alternatively, cast the result of the expression to a different data type. If there are precision issues, consider casting to a double or float type, or using a generic floating-point data type for the key.

- Error number: 1024; Symbol: HW_ER_TABLE_NOT_LOADED

  Message: The table, `%s`.`%s`, %s.

  Description: Table is not available in HeatWave.

  Workaround: Load the table using ALTER TABLE [table] SECONDARY_LOAD.

- Error number: 1026; Symbol: HW_ER_UNSUPPORTED_AGG_OVER_ENUM

  Message: Aggregation over an ENUM column is not supported in, [%s].

  Description: HeatWave does not support this aggregation function with ENUM datatype.

  Workaround: Only count, average, and sum are supported. Use a non-ENUM type for aggregation, or convert the enum to a numeric.

- Error number: 1027; Symbol: HW_ER_CONFLICT_IN_OPTIMIZER_CONFIG

  Message: HeatWave requires either the Hypergraph Optimizer or the Old Optimizer, but not both.

Description: Only one query optimizer can be used at a time.

Workaround: Use either the Hypergraph Optimizer or the Old Optimizer. Set session variable to use optimizers, SET rapid_use_old_optimizer = 'ON' or SET optimizer_switch= 'hypergraph_optimizer=on'.

- Error number: 1028; Symbol: HW_ER_PLUGIN_NOT_READY

  Message: HeatWave plugin is not in READY state, yet. Please wait and try again.

  Description: For the HeatWave plugin to be usable, it must be in the correct state.

  Workaround: Wait for plugin to be in READY state.

- Error number: 1029; Symbol: HW_ER_UNEXPECTED_ITEM_TYPE

  Message: Unexpected item, [%s] of type, [%s] - %s.

  Description: A problem occurred during the logical transformation of the query.

  Workaround: Wrap the item inside a function.

- Error number: 1030; Symbol: HW_ER_UNSUPPORTED_RECURSIVE_QUERY_EXPRESSIONS

  Message: Recursive query expression, [%s], is not supported.

  Description: A problem occurred during the logical transformation of the query.

  Workaround: Make the expression non-recursive.

- Error number: 1031; Symbol: HW_ER_INVALID_JOIN_CONDITION

  Message: You have not specified the condition for one of the tables involved in the join condition, [%s].

  Description: Tables cannot be joined without a join condition.

  Workaround: Add a join condition for each table involved in the join.

- Error number: 1032; Symbol: HW_ER_NUM_PROJ_COLS_EXCEEDS_MAX_ALLOWED

  Message: The number of projection columns, [%d], exceeds maximum supported columns, [%d].

  Description: HeatWave supports projection of limited number of columns.

  Workaround: Reduce the number of projected columns. Alternately, split and project the columns across multiple queries.

- Error number: 1033; Symbol: HW_ER_FETCHING_NUM_ROWS

  Message: Error fetching number of rows for `%s`.`%s`.

  Description: The table statistics could be incorrect.

  Workaround: Retry the query. If the issue persists, try rewriting the query to work around the error.

- Error number: 1034; Symbol: HW_ER_ITEM_NOT_FOUND

  Message: No projectable field item found while compiling AccessPath for query block id, [%u].

  Description: A problem occurred during the logical transformation of the query.

  Workaround: Make sure that each query block projects a minimum of one column.

- Error number: 1035; Symbol: HW_ER_UNSUPPORTED_DUAL_INPUT

  Message: Projecting ['%s'], having input from both sides of a join, is not supported.

  Description: A problem occurred during the logical transformation of the query.

  Workaround: Ensure that join conditions and projections are specified. Retry the query.

- Error number: 1036; Symbol: HW_ER_UNSUPPORTED_ACCESS_PATH_TYPE

  Message: Unsupported AccessPath - %s.

  Description: A problem occurred during the logical transformation of the query.

  Workaround: Retry the query. If that doesn't help, try switching to a different optimizer.

- Error number: 1037; Symbol: HW_ER_UNSUPPORTED_EXPR_IN_CORRELATED_SUBQUERIES

  Message: The expression, [%s], is not supported for correlated subqueries.

  Description: The specific functionality for correlated subqueries is absent in HeatWave.

  Workaround: Simplify or modify the expression.

- Error number: 1038; Symbol: HW_ER_INCOMPATIBLE_FUNC_ARG_ENCODINGS

  Message: Incompatible argument encodings, ['%s', '%s'], for function, [%s].

  Description: The encoding of function arguments must be correct.

  Workaround: Use supported encoding. Refer to HeatWave documentation for function usage, argument data types and supported encodings.

- Error number: 1039; Symbol: HW_ER_DUMMY_PROJ_ABSENT

  Message: Could not find dummy projection for %s.

  Description: A problem occurred during the logical transformation of the query.

  Workaround: Verify that the SELECT list contains desired projection columns.

- Error number: 1040; Symbol: HW_ER_FILTER_ACCESS_PATH_CONDITION

  Message: Error in extracting required items from filter AccessPath's condition, [%s].

  Description: A problem occurred during the logical transformation of the query.

  Workaround: Modify the filter condition and retry the query.

- Error number: 1041; Symbol: HW_ER_MISSING_ACCESS_PATH_TYPE

  Message: AccessPath '%s' is absent - %s.

  Description: A problem occurred during the logical transformation of the query.

  Workaround: Retry the query. If the issue persists, try rewriting the query to work around the error.

- Error number: 1042; Symbol: HW_ER_TOPK_QKRN_CREATION

  Message: Error - failed to process the sort operation with select limit [%s] and offset limit [%s].

  Description: A problem occurred during the logical transformation of the query.

  Workaround: Examine and fix any other errors that have been thrown for Top-K.

- Error number: 1043; Symbol: HW_ER_ORDER_BY_LIMIT_OFFSET

  Message: Error while handling ORDER BY / LIMIT [OFFSET] for Query_term_unary with select_number, [%u].

  Description: A problem occurred during the logical transformation of the query.

  Workaround: Examine and fix any other errors that have been thrown for ORDER BY.

- Error number: 1044; Symbol: HW_ER_INCOMPATIBLE_FUNC_ARG_TYPES

  Message: Incompatible argument types, [%d, %d], for function, [%s].

  Description: The encoding of function arguments must be correct.

  Workaround: Use supported types in function arguments. Refer to HeatWave documentation for function usage and argument data types and supported encodings.

- Error number: 1045; Symbol: HW_ER_UNSUPPORTED_ACCESS_PATH_TYPE_WITHOUT_GBY

  Message: '%s' without GROUP BY is not supported.

  Description: A problem occurred during the logical transformation of the query.

  Workaround: Use supported clause within GROUP BY.

- Error number: 1046; Symbol: HW_ER_UNSUPPORTED_DERIVED_TABLE_WITHOUT_PROJ

  Message: Derived table without projection not supported.

  Description: A problem occurred during the logical transformation of the query.

  Workaround: Project necessary columns from the derived TABLE/VIEW/CTE.

- Error number: 1047; Symbol: HW_ER_EMPTY_BASE_FIELDS_IN_DERIVED_TABLE

  Message: Base fields of a derived table should not have been empty.

  Description: A problem occurred during the logical transformation of the query.

  Workaround: Project necessary columns from the derived TABLE/VIEW/CTE.

- Error number: 1048; Symbol: HW_ER_PROJECTION_OUT_OF_BOUNDS

  Message: Projection from derived table or CTE is out of bounds.

  Description: A problem occurred during the logical transformation of the query.

  Workaround: Project necessary columns from the derived TABLE/VIEW/CTE.

- Error number: 1049; Symbol: HW_ER_EMPTY_PROJ_LIST_IN_PARTIAL_PLAN

  Message: Projection list should not be empty.

  Description: Problem occured during logical transformation of query.

  Workaround: The SELECT list should project at least one column.

- Error number: 1050; Symbol: HW_ER_JOIN_ABSENT_AFTER_SET

  Message: Join absent after SET operation - %s.

  Description: Problem occured during logical transformation of query.

  Workaround: Retry the query. If the issue persists, try rewriting the query to work around the error.

- Error number: 1051; Symbol: HW_ER_DISTINCT_AFTER_SET_FAILURE

  Message: Failed to handle DISTINCT after SET operation.

  Description: Problem occured during logical transformation of query.

  Workaround: Retry the query. If the issue persists, try rewriting the query to work around the error.

- Error number: 1052; Symbol: HW_ER_ORDERBY_LIMIT_OFFSET_AFTER_SET_FAILURE

  Message: Error while handling ORDER BY / LIMIT [OFFSET] after SET operation.

  Description: Problem occured during logical transformation of query.

  Workaround: Check and fix any more errors thrown for ORDER BY.

- Error number: 1053; Symbol: HW_ER_UNION_TRANSLATION

  Message: Error while translating UNION.

  Description: Problem occured during logical transformation of query.

  Workaround: Check if UNION is well formed. Check and fix any more errors thrown for SET operation.

- Error number: 1054; Symbol: HW_ER_INTERSECT_EXCEPT_TRANSLATION

  Message: Error while translating INTERSECT/EXCEPT - %s.

  Description: Problem occured during logical transformation of query.

  Workaround: Check if INTERSECT/EXCEPT is well formed. Check and fix any more errors thrown for SET operation.

- Error number: 1055; Symbol: HW_ER_UNSUPPORTED_MATERIALIZE_TYPE

  Message: Unsupported MATERIALIZE type.

  Description: Problem occured during logical transformation of query.

  Workaround: Try making a few little adjustments to various sections of the query.

- Error number: 1056; Symbol: HW_ER_CANT_PROJ_WINDOW_FUNC_FROM_ZEROROWSAGG

  Message: Cannot project a WINDOW function from ZeroRowsAggregated - '%s' (while processing [%s]).

  Description: Problem occured during logical transformation of query.

  Workaround: Set appropriate projections and aggregations for the WINDOW function.

- Error number: 1057; Symbol: HW_ER_UNEXPECTED_NULL_JOIN

  Message: Root-level UNION ALL - unexpected NULL JOIN under STREAM Access Path.

  Description: Problem occured during logical transformation of query.

Workaround: Retry the query. If the issue persists, try rewriting the query to work around the error.

- Error number: 1058; Symbol: HW_ER_UNSUPPORTED_QUERY_TERM_TYPE

  Message: Query term type, [%d] is not supported in HeatWave.

  Description: Problem occured during logical transformation of query.

  Workaround: Retry the query. If the issue persists, try rewriting the query to work around the error.

- Error number: 1059; Symbol: HW_ER_UNSUPPORTED_CAST

  Message: Unsupported CAST - %s.

  Description: The specific functionality in casting operator is not available in HeatWave.

  Workaround: Use casting for supported data type.

- Error number: 1060; Symbol: HW_ER_INCOMPATIBLE_ENCODINGS

  Message: Incompatible encodings - %s.

  Description: The operator's specific functionality on the specified encoding is not present in HeatWave.

  Workaround: Use supported encoding for the operation.

- Error number: 1061; Symbol: HW_ER_INCOMPATIBLE_DATA_TYPES

  Message: Incompatible data types - %s.

  Description: The operator's specific functionality on the specified data type is not present in HeatWave.

  Workaround: Use compatible data types for the operation.

- Error number: 1062; Symbol: HW_ER_UNSUPPORTED_OPERATION_OVER_NON_VARLEN_COLS

  Message: Unsupported operation over non-varlen encoded cols - %s.

  Description: The operator's specific functionality on the non-varlen encoded columns is not present in HeatWave.

  Workaround: Either use varlen encoded column for operation.

- Error number: 1063; Symbol: HW_ER_UNSUPPORTED_OPERATION_OVER_DICT_ENCODED_COLS

  Message: Unsupported operation over dictionary encoded cols - %s.

  Description: The operator's specific functionality on the dictionary encoded columns is not present in HeatWave.

  Workaround: Use varlen encoded column for the operation.

- Error number: 1064; Symbol: HW_ER_UNSUPPORTED_OPERATION_OVER_NON_PVARCHAR_COLS

  Message: Compression, [%s], is supported only on padded VARCHAR columns.

  Description: The compression functionality for padded VARCHAR columns is not present in HeatWave.

  Workaround: Use operation on supported column type.

- Error number: 1065; Symbol: HW_ER_UNSUPPORTED_NUM_ARGS_IN_IN_CLAUSE

  Message: The number of arguments, [%u], for in-list, [%s], is not supported.

  Description: IN-list supports limited number of arguments.

  Workaround: Make sure there is one constant list entry on the right side for each row on the left side if the left side is row set.

- Error number: 1066; Symbol: HW_ER_UNSUPPORTED_EXPR_OVER_VARLEN_COLS

  Message: The expression, [%s], is not supported on varlen encoded columns.

  Description: The supported data type for this expression is absent.

  Workaround: Avoid varlen encoding in non-supported expression.

- Error number: 1067; Symbol: HW_ER_INCOMPATIBLE_CHARSET_COLLATION

  Message: Incompatible charset/collation - %s.

  Description: Supported of charset/collation for this functionality is absent.

  Workaround: Use a compatible charset/collation.

- Error number: 1068; Symbol: HW_ER_CANT_EXTRACT_TABLE_REF_FOR_EXPR

  Message: Could not extract table ref from expression, [%s].

  Description: Problem occured during logical transformation of query.

  Workaround: Retry the query. If the issue persists, try rewriting the query to work around the error.

- Error number: 1069; Symbol: HW_ER_UNSUPPORTED_OPERATION_OVER_DERIVED_TABLE

  Message: Trying to construct a column from an Item_field which points into a derived table, is not supported in expression, [%s].

  Description: Problem occured during logical transformation of query.

  Workaround: Retry the query. If the issue persists, try rewriting the query to work around the error.

- Error number: 1070; Symbol: HW_ER_VALUE_OUT_OF_RANGE

  Message: Value out of range - %s.

  Description: HeatWave supports data types with limited range.

  Workaround: Use the values within the range.

- Error number: 1071; Symbol: HW_ER_DECIMAL_PRECISION_TOO_LARGE

  Message: Decimal precision, [%s], is too large to be supported in HeatWave.

  Description: HeatWave supports finite precision.

  Workaround: Use precision within supported range.

- Error number: 1072; Symbol: HW_ER_CANNOT_REPLACE_ENUM

  Message: Cannot replace [%s], on ENUM input with [%s], in [%s].

  Description: The specific functionality for ENUM data type is not available in HeatWave.

  Workaround: Use valid ENUM value.
- Error number: 1073; Symbol: HW_ER_UNSUPPORTED_OPERATION_OVER_UNSORTED_DICT_COLS

  Message: Unsupported operation on unsorted column - %s.

  Description: The operation can be used only on sorted dictionary columns.

  Workaround: Use sorted dictionary column for the operation.
- Error number: 1074; Symbol: HW_ER_UNSUPPORTED_SORT_ON_UNSORTED_DICT

  Message: Sort on unsorted DICT, is not supported in [%s].

  Description: The sorting on unsorted dictionary column is not available.

  Workaround: Avoid sorting on dictionary encoded column.
- Error number: 1075; Symbol: HW_ER_UNEXPECTED_NON_OPERATOR_EXPR_FOR_JOIN

  Message: Unexpected non-operator expression for JOIN predicate, #%u [`%s`].

  Description: Problem occured during logical transformation of query.

  Workaround: Use a valid operator expression for JOIN predicate.
- Error number: 1076; Symbol: HW_ER_UNEXPECTED_NON_BINARY_EXPR_FOR_JOIN

  Message: Unexpected non-binary expression for JOIN predicate, [%s].

  Description: Problem occured during logical transformation of query.

  Workaround: Use a binary expression with correct number of arguments for JOIN predicate.
- Error number: 1077; Symbol: HW_ER_CANNOT_CREATE_ORDERBY_KEY

  Message: Error when creating ORDER BY key, `%s`.

  Description: Problem occured during logical transformation of query.

  Workaround: Please check and resolve if any other error.
- Error number: 1078; Symbol: HW_ER_INVALID_DATA_TYPE_FOR_CONST

  Message: Data type for constant is not supported - error while processing, [%s].

  Description: The constant value is of usupported data type.

  Workaround: Change the data type of constant or use a column.
- Error number: 1079; Symbol: HW_ER_DUMMY_OPERAND_NOT_POPULATED

  Message: RV - dummy operand [%s], not populated.

  Description: Problem occured during logical transformation of query.

  Workaround: Retry the query. If the issue persists, try rewriting the query to work around the error.

- Error number: 1080; Symbol: HW_ER_INVALID_DUMMY_PROJ

  Message: There is an invalid dummy projection, [%s], in a '%s' node.

  Description: Problem occured during logical transformation of query.

  Workaround: Remove the dummy projection.

- Error number: 1081; Symbol: HW_ER_UNSUPPORTED_COLUMN_ENCODING

  Message: Unsupported encoding, '%s', while processing [%s].

  Description: The columns has unsupported encoding.

  Workaround: Use supported encoding for column.

- Error number: 1082; Symbol: HW_ER_NONPRIMITIVE_GBY_TRANSFORMATION

  Message: GROUP BY transformation failed - %s.

  Description: Problem occured during logical transformation of query.

  Workaround: Avoid ROLLUP or CUBE in GROUP BY.

- Error number: 1083; Symbol: HW_ER_UNSUPPORTED_OPERATION_OVER_VARLEN_COLS

  Message: IS TRUE/FALSE function with varlen argument is not supported.

  Description: The specific functionality is not available in HeatWave.

  Workaround: Use non-varlen column with IS TRUE or IS FALSE.

- Error number: 1084; Symbol: HW_ER_CANT_FIND_SUBSTITUTE_ITEM_FIELD

  Message: Cannot find substitute item field for, [%s], in order by key, [%s].

  Description: Problem occured during logical transformation of query.

  Workaround: Retry the query. If the issue persists, try rewriting the query to work around the error.

- Error number: 1085; Symbol: HW_ER_UNSUPPORTED_WINDOW_FUNC_FRAME_DEF

  Message: WINDOW function's frame definition, is not supported.

  Description: The specific functionality of WINDOW function is not available in HeatWave.

  Workaround: Adjust the frame definition of the WINDOW function.

- Error number: 1086; Symbol: HW_ER_UNSUPPORTED_EXCL_IN_WINDOW_FUNC_FRAME

  Message: Exclusion in WINDOW function's frame, is not supported.

  Description: The specific functionality of WINDOW function is not available in HeatWave.

  Workaround: Remove the exclusion from WINDOW function definition.

- Error number: 1087; Symbol: HW_ER_UNSUPPORTED_NONPRIMITIVE_GBY_MODIFIER_WITH_DUPLICATE_KEYS

  Message: GROUP BY with %s modifier and duplicate GROUP BY keys, is not supported in [%s].

  Description: The specific functionality of GROUP BY clause is not available in HeatWave.

  Workaround: Remove duplicate keys from GROUP BY.

- Error number: 1088; Symbol: HW_ER_UNSUPPORTED_TYPE_IN_DISTINCT

  Message: Type in DISTINCT [%s], is not supported.

  Description: The specific functionality of DISTINCT clause is not available in HeatWave.

  Workaround: Change the type of DISTINCT.

- Error number: 1089; Symbol: HW_ER_INCOMPATIBLE_FUNC_ARG

  Message: Incompatible argument, ['%s'], for function [%s].

  Description: The support of this data type in argument of function is not available in HeatWave.

  Workaround: Use a supported type for the function parameter. Refer to HeatWave documentation for function usage and argument data types.

- Error number: 1090; Symbol: HW_ER_UNSUPPORTED_JOIN_PREDICATE

  Message: JOIN predicate, [%s], is not supported.

  Description: The specific functionality in JOIN predicate is not available in HeatWave.

  Workaround: Equality or NULL-safe equality should be present in the JOIN predicate.

- Error number: 1091; Symbol: HW_ER_UNSUPPORTED_GBY_ON_FIXED_PRECISION

  Message: GROUP BY on fixed precision floating point numbers, [%s], is not supported.

  Description: The specific functionality is not available in HeatWave.

  Workaround: Switch from fixed precision to variable precision GROUP BY key.

- Error number: 1092; Symbol: HW_ER_JOIN_BETWEEN_DISTINCT_ROOTS

  Message: Error during JOIN between distinct roots.

  Description: Problem occured during logical transformation of query.

  Workaround: Retry the query. Or, try to remove some COUNT DISTINCT keys.

- Error number: 1093; Symbol: HW_ER_GBY_KEY_EXTRACT_FOR_INTERSECT_EXCEPT

  Message: Extracting GROUP BY key failed for INTERSECT/EXCEPT operation.

  Description: Problem occured during logical transformation of query.

  Workaround: Retry the query. If the issue persists, try rewriting the query to work around the error.

- Error number: 1094; Symbol: HW_ER_UNSUPPORTED_NONPRIMITIVE_GBY_MODIFIER_WITH_DISTINCT_IN_ITEM

  Message: %s in combination with [%s], in [%s], is not supported.

  Description: The support for this combination of different functionalities is absent in HeatWave.

  Workaround: Remove GROUP BY modifier.

- Error number: 1095; Symbol: HW_ER_UNSUPPORTED_NONPRIMITIVE_GBY_MODIFIER_WITH_DISTINCT

  Message: %s in combination with [%s], is not supported.

  Description: The support for this combination of different functionalities is absent in HeatWave.

Workaround: Remove GROUP BY modifier if using HLL or DISTINCT.

- Error number: 1096; Symbol: HW_ER_DISTINCT_KEY_OUT_OF_BOUNDS

  Message: DISTINCT key, [%s], is out of bounds.

  Description: Problem occured during logical transformation of query.

  Workaround: Retry the query. If the issue persists, try rewriting the query to work around the error.
- Error number: 1097; Symbol: HW_ER_UNSUPPORTED_INTERVAL_TYPE

  Message: The INTERVAL type, [%s], is not supported.

  Description: The specific functionality of INTERVAL is not available in HeatWave.

  Workaround: Use supported precision for time INTERVAL.
- Error number: 1098; Symbol: HW_ER_NO_LOADED_COLS_FROM_TABLE

  Message: No columns loaded to HeatWave from table, `%s`.`%s`.

  Description: HeatWave required columns from table to be loaded.

  Workaround: Load all columns of table to HeatWave.
- Error number: 1099; Symbol: HW_ER_SELECT_EXPR_NOT_PROJECTED

  Message: %s : expr, [%s], cannot be projected.

  Description: Problem occured during logical transformation of query.

  Workaround: Retry the query. If the issue persists, try rewriting the query to work around the error.
- Error number: 1100; Symbol: HW_ER_ITEM_LIST_RESIZE_FAILURE

  Message: More columns found than expected and resize failed - %s.

  Description: Problem occured during logical transformation of query.

  Workaround: Modify the projection columns and retry the query.
- Error number: 1101; Symbol: HW_ER_CANNOT_FIND_OBY_KEY_AFTER_SET

  Message: Could not find key for ORDER BY, [%s], after SET operation.

  Description: Problem occured during logical transformation of query.

  Workaround: Retry the query. Make sure the ORDER BY keys are correctly defined.
- Error number: 1102; Symbol: HW_ER_EMPTY_PROJ_KEYS_FOR_OBY_AFTER_SET

  Message: SET operation children, do not project keys for ORDER BY, [%s], after SET operation.

  Description: Problem occured during logical transformation of query.

  Workaround: Retry the query. Make sure the ORDER BY keys are correctly defined.
- Error number: 1103; Symbol: HW_ER_CHARSET_CONVERSION_FAILED

  Message: Conversion charset between SET operation columns failed.

  Description: Problem occured during logical transformation of query.

Workaround: Use same charset for columns.

- Error number: 1104; Symbol: HW_ER_INCOMPATIBLE_DATA_TYPES_FOR_SET_OPERATION

  Message: SET operation between expressions with data types, [%s, %s], is not supported.

  Description: Problem occured during logical transformation of query.

  Workaround: Use same data type for both arguments.

- Error number: 1105; Symbol: HW_ER_INCOMPATIBLE_ENCODINGS_FOR_SET_OPERATION

  Message: SET operation between differently encoded DICT columns, is not supported.

  Description: SET operations are supported on similarly DICT encoded columns.

  Workaround: Use same DICT encoding for columns.

- Error number: 1106; Symbol: HW_ER_CANT_CREATE_TABLE_SCAN_QKRN

  Message: Could not create TableScan operation - %s.

  Description: Problem occured during logical transformation of query.

  Workaround: Retry the query. If the issue persists, try rewriting the query to work around the error.

- Error number: 1107; Symbol: HW_ER_INTERSECT_DISTINCT_TRANSLATION

  Message: Error during translation of INTERSECT DISTINCT.

  Description: Problem occured during logical transformation of query.

  Workaround: Retry the query. If the issue persists, try rewriting the query to work around the error.

- Error number: 1108; Symbol: HW_ER_EXCEPT_DISTINCT_TRANSLATION

  Message: Error during translation of EXCEPT DISTINCT.

  Description: Problem occured during logical transformation of query.

  Workaround: Retry the query. If the issue persists, try rewriting the query to work around the error.

- Error number: 1109; Symbol: HW_ER_INTERSECT_ALL_TRANSLATION

  Message: Error during translation of INTERSECT ALL.

  Description: Problem occured during logical transformation of query.

  Workaround: Retry the query. If the issue persists, try rewriting the query to work around the error.

- Error number: 1110; Symbol: HW_ER_INVALID_ENCODING

  Message: Invalid encoding found for %s operation, used in qkrn node with id, [%lu].

  Description: Problem occured during logical transformation of query.

  Workaround: Use supported encoding for the columns and operations.

- Error number: 1111; Symbol: HW_ER_EXCEPT_ALL_TRANSLATION

  Message: Error during translation of EXCEPT ALL.

  Description: Problem occured during logical transformation of query.

Workaround: Retry the query. If the issue persists, try rewriting the query to work around the error.

- Error number: 1112; Symbol: HW_ER_UNSUPPORTED_ENCODING_TYPE

  Message: Encoding type, [%d], is not supported for comparison.

  Description: The specific functionality is not available in HeatWave.

  Workaround: Use supported encoding for the columns and operations.

- Error number: 1113; Symbol: HW_ER_UNSUPPORTED_EXPR_FOR_IN_OPERATION

  Message: Multi-value binary expression, [%s], with NULL left operand is not supported.

  Description: The specific functionality is not available in HeatWave.

  Workaround: Use a column in place of a constant.

- Error number: 1114; Symbol: HW_ER_UNSUPPORTED_MULTIVAL_BINARY_EXPR

  Message: Multi-value binary expression, [%s], is not supported - %s.

  Description: The specific functionality is not available in HeatWave.

  Workaround: Use a column in place of a constant.

- Error number: 1115; Symbol: HW_ER_INVALID_ENUM_IN_BINARY_EXPR

  Message: ENUM, [%s], in binary expression, [%s], is not supported.

  Description: The specific functionality is not available in HeatWave.

  Workaround: Cast ENUM type to a supported type.

- Error number: 1116; Symbol: HW_ER_UNABLE_TO_GENERATE_ARG_FOR_FUNC

  Message: Failed to generate left side argument for function, [%s].

  Description: Problem occured during logical transformation of query.

  Workaround: Check and resolve any other error. Retry the query.

- Error number: 1117; Symbol: HW_ER_GENERATING_PRECISION_CONST

  Message: Failed to generate precision const for function, [%s] with non-zero precision, [%f].

  Description: Problem occured during logical transformation of query.

  Workaround: Use arguments with similar data type. Use data types with similar precision.

- Error number: 1118; Symbol: HW_ER_CTAS_BINLOG_FORMAT_STMT

  Message: `CREATE TABLE ... AS SELECT` or `INSERT INTO ... SELECT` statements using HeatWave tables are not supported when binlog_format is set to STATEMENT.

  Description: STATEMENT based replication blocks `CREATE TABLE ... AS SELECT` or `INSERT INTO ... SELECT` statements using HeatWave tables.

  Workaround: binlog_format needs to be set to ROW/MIXED for `CREATE TABLE ... AS SELECT` or `INSERT INTO ... SELECT` statements using HeatWave tables

- Error number: 1119; Symbol: HW_ER_INCOMPATIBLE_DATA_TYPES_IN_BINARY_EXPR

  Message: Type, [%s], is not supported in binary expression, [%s].

  Description: The specific functionality is not available in HeatWave.

  Workaround: Cast left or right hand side of binary expression to a supported data type.

- Error number: 1120; Symbol: HW_ER_UNSUPPORTED_DTYPE_CONVERSION

  Message: GREATEST/LEAST function, [%s], where result data type is not STRING and at least one argument data type is STRING column or operator, is not supported.

  Description: The specific functionality is not available in HeatWave.

  Workaround: Use supported data types for arguments in operators.

- Error number: 1121; Symbol: HW_ER_CANT_EXTRACT_EXPR_FROM_DERIVED_TBL

  Message: Error when extracting expression from derived table, in [%s].

  Description: Problem occured during logical transformation of query.

  Workaround: Retry the query. If the issue persists, try rewriting the query to work around the error.

- Error number: 1122; Symbol: HW_ER_UNSUPPORTED_FUNC_OVER_ENUM

  Message: Function, [%s], is not supported with ENUM columns.

  Description: The specific functionality is not available in HeatWave.

  Workaround: Use non-enum column with function.

- Error number: 1123; Symbol: HW_ER_UNSUPPORTED_MULTIARG_BINARY_EXPR

  Message: Binary multi-argument expression, with constant lhs, is not supported in, [%s].

  Description: The specific functionality is not available in HeatWave.

  Workaround: The left hand side of expression should not be constant. A NULL column can be used instead.

- Error number: 1124; Symbol: HW_ER_UNSUPPORTED_NATIVE_IN_WITH_DIFFERENT_PRECISIONS

  Message: Native-in on numbers with different precisions, is not supported in, [%s].

  Description: The specific functionality is not available in HeatWave.

  Workaround: Use same precision for numerical arguments in the IN list. Use an explicit cast operator with the same precision for floats or decimals arguments.

- Error number: 1126; Symbol: HW_ER_UNSUPPORTED_COND_IN_CASE

  Message: The CASE expression, [%s], returns a constant value. This is not advised and is not supported. It is recommended to eliminate this expression.

  Description: The specific functionality is not available in HeatWave.

  Workaround: Simplify the expression to avoid constant return value. Modify this expression to make the outcome non-constant. Try constructing an expression that returns a column or replacing constants with a column.

- Error number: 1127; Symbol: HW_ER_TOO_MANY_FUNC_ARGS

  Message: HeatWave supports only up to %u function arguments. The function is [%s], and number of arguments is %u.

  Description: HeatWave supports only limited number of function arguments.

  Workaround: Reduce the number of arguments to function within supported limit. Refer to HeatWave documentation for usage of built-in functions.

- Error number: 1128; Symbol: HW_ER_UNSUPPORTED_STRING_OUTPUT_FUNC

  Message: The function, [%s], results in a string output, which is not supported.

  Description: The specific functionality is not available in HeatWave.

  Workaround: Remove any user defined function. Or, simplify the string expression.

- Error number: 1129; Symbol: HW_ER_UNSUPPORTED_FUNC_WITHOUT_ARGS

  Message: Function, [%s], without arguments, is not supported.

  Description: HeatWave require arguments for this function.

  Workaround: Provide the function with a minimum of one argument. Refer to HeatWave documentation for the function usage.

- Error number: 1130; Symbol: HW_ER_UNSUPPORTED_EQUALITY_BETWEEN_NULL_ITEMS

  Message: Equality on 2 NULL items, [%s], is not supported.

  Description: The specific functionality is not available in HeatWave.

  Workaround: Use minimum one non-NULL item for comparison.

- Error number: 1131; Symbol: HW_ER_INCOMPATIBLE_DATA_TYPES_IN_JOIN_COND

  Message: Incompatible data types, ('%s','%s'), in join condition, [%s], is not supported.

  Description: The specific functionality is not available in HeatWave.

  Workaround: Define the join condition with same data type. Alternatively use cast operator to convert into same data type.

- Error number: 1132; Symbol: HW_ER_UNSUPPORTED_NUM_OF_ARGS_IN_IF

  Message: IF, [%s] is not supported. IF expression is currently only supported with exactly 3 arguments in HeatWave.

  Description: The specific functionality is not available in HeatWave.

  Workaround: For three or more arguments, use nested IF. Alternatively use a SWITCH CASE.

- Error number: 1133; Symbol: HW_ER_UNSUPPORTED_CASE_COND_OVER_CONSTS

  Message: A CASE conditioned over constants cannot be evaluated. It is recommended to eliminate this CASE expression, [%s].

  Description: The specific functionality is not available in HeatWave.

  Workaround: Simplify CASE expression for evaluation. Or remove CASE expression.

- Error number: 1134; Symbol: HW_ER_UNSUPPORTED_ARG_TYPE_FOR_LIKE

Message: The argument type in, [%s], is not supported.

Description: The specific functionality is not available in HeatWave.

Workaround: Use a varlen input to LIKE clause.

- Error number: 1135; Symbol: HW_ER_INTERPRETING_ESCAPE_CHARACTER

  Message: Error interpreting the escape character in expr, [%s].

  Description: Escape character could be incorrect in expression.

  Workaround: Double-check the escape character in expression.

- Error number: 1136; Symbol: HW_ER_UNSUPPORTED_LIKE_EXPR

  Message: LIKE expression, [%s] is not supported.

  Description: The specific functionality is not available in HeatWave.

  Workaround: Try some alternative in pattern string. E.g. eliminate '_' or use '%' at the end .

- Error number: 1137; Symbol: HW_ER_UNSUPPORTED_STORED_PROCEDURES

  Message: Statements that reference stored functions are not supported in secondary engines.

  Description: The specific functionality is not available in HeatWave.

  Workaround: Do not use stored functions.

- Error number: 1138; Symbol: HW_ER_UNSUPPORTED_CONVERSION_FUNC

  Message: Only decimal to binary conversion is supported in function, [%s].

  Description: The specific functionality is not available in HeatWave.

  Workaround: Use this function for converting decimals to binary. Refer to HeatWave documentation for function usage and argument data types and supported encodings.

- Error number: 1139; Symbol: HW_ER_UNSUPPORTED_FUNC_WITH_NUM_ARGS

  Message: Function, [%s], with %u arguments is not supported.

  Description: HeatWave supports functions with finite number of arguments.

  Workaround: Use supported number of arguments in function. Refer to HeatWave documentation for function usage and argument data types.

- Error number: 1140; Symbol: HW_ER_COLS_MARKED_NOT_SECONDARY

  Message: You have marked one or more read columns as NOT SECONDARY, which is not supported.

  Description: HeatWave secondary storage engine execution is not supported, if some query columns are not loaded in secondary engine.

  Workaround: Remove NOT SECONDARY attribute from the column definition, then reload the table.

- Error number: 1141; Symbol: HW_ER_UNSUPPORTED_WHERE_CLAUSE

  Message: WHERE clause, [%s], is not supported.

  Description: The specific functionality is not available in HeatWave.

Workaround: Modify the condition in WHERE clause and use suuported data types for arguments.

- Error number: 1142; Symbol: HW_ER_AUTOCOMMIT_TURNED_OFF

  Message: You cannot be offload queries to HeatWave secondary engine because autocommit is disabled.

  Description: HeatWave secondary storage engine execution is not supported, if autocommit option is turned off.

  Workaround: Use "SET autocommit = 0;" to enable auto-commit.

- Error number: 1143; Symbol: HW_ER_QUERY_TABLE_READ_OR_WRITE_LOCKED

  Message: Query is run in a prelocked table mode, at least one of the query tables are read or write locked.

  Description: HeatWave secondary storage engine execution is not supported, if tables are read or write locked.

  Workaround: Check if you have a trigger that has a forced secondary engine statement. Run the query after unlocking the locked tables.

- Error number: 1144; Symbol: HW_ER_QUERY_PART_OF_MULTISTMT_TRANSACTION

  Message: Query is part of a multi-statement transaction, which is not supported in secondary engines.

  Description: HeatWave secondary storage engine execution is not supported, if the query is part of a multi-statement transaction.

  Workaround: Break multi-statement transaction into separate single transactions.

- Error number: 1145; Symbol: HW_ER_ILLEGAL_ARGS_IN_REGULAR_EXPR

  Message: Illegal argument, [%s], to a regular expression, [%s].

  Description: Arguments of regular expression might be incorrect.

  Workaround: Use non-empty input for regular expression. E.g. remove if any string ('') argument.

- Error number: 1146; Symbol: HW_ER_UNSUPPORTED_OUTPUT_DATA_TYPE_FOR_FUNC

  Message: Unexpected data type as output of function, [%s].

  Description: Function with this output data type is not supported.

  Workaround: Use supported data type if function output is an argument to another function.

- Error number: 1147; Symbol: HW_ER_COULD_NOT_CONSTRUCT_OPERATOR

  Message: Error while constructing operator for function, [%s].

  Description: Problem occured during logical transformation of query.

  Workaround: Check for any errors or warnings related to operators, expressions, data types, etc. and fix them. Use SHOW WARNINGS for warnings or SHOW ERRORS for erros. Resolve the issue and retry the query.

- Error number: 1148; Symbol: HW_ER_UNSUPPORTED_COND_EXPR

  Message: Conditional expression, [%s] is not supported.

Description: The specific functionality is not available in HeatWave.

Workaround: Use only AND or OR in this expression.

- Error number: 1149; Symbol: HW_ER_OR_CONDITION

  Message: There is an error while processing the OR [%s] condition.

  Description: A problem occurred during the logical transformation of the query.

  Workaround: Verify that the condition is well-formed. Simplify and resolve the problem with condition and retry the query.

- Error number: 1150; Symbol: HW_ER_UNSUPPORTED_COUNT_NULL

  Message: COUNT(NULL) as an input argument, [%s], is not supported.

  Description: The specific functionality is not available in HeatWave.

  Workaround: Remove COUNT(NULL) from expression. Alternatively, replace NULL with non-NULL argument.

- Error number: 1152; Symbol: HW_ER_UNSUPPORTED_BINARY_EXPR_WITH_AGG_FUNC

  Message: Binary expression involving aggregate function, [%s], is not supported.

  Description: The specific functionality is not available in HeatWave.

  Workaround: Use constant or column instead of aggregate function.

- Error number: 1153; Symbol: HW_ER_UNSUPPORTED_PROJ_OF_NON_NULL_FROM_FTAG

  Message: Projecting of non-nullable item as part of FTAG is not allowed in, [%s].

  Description: A problem occurred during the logical transformation of the query.

  Workaround: Either make the item nullable or use a item which is nullable.

- Error number: 1154; Symbol: HW_ER_UNSUPPORTED_EXPR_TYPE

  Message: Expression, [%s], of type, [%s], is not supported in HeatWave.

  Description: The specific functionality is not available in HeatWave.

  Workaround: For valid values in other data types, use casting function to convert the expression's data type into supported one.

- Error number: 1155; Symbol: HW_ER_UNSUPPORTED_OPERATOR_WITH_ARG_IN_FUNC

  Message: Operator, [%s], is not supported with argument, [%s], in function, [%s].

  Description: The specific functionality is not available in HeatWave.

  Workaround: Use supported type for operator. Refer to HeatWave documentation for function and operators usage and argument data types.

- Error number: 1156; Symbol: HW_ER_UNSUPPORTED_COND_EXPR_WITH_GBY_OR_DISTINCT

  Message: GROUP BY or DISTINCT cannot handle conditional expression, [%s], which evaluates to a constant, [%s].

  Description: The specific functionality is not available in HeatWave.

Workaround: Modify this expression to result in a non-constant or write an expression that returns a column or try replacing constants with a column.

- Error number: 1157; Symbol: HW_ER_UNSUPPORTED_BETWEEN_OVER_ENUM

  Message: BETWEEN, [%s], over ENUM, [%s], is not supported with, [%s, %s].

  Description: The specific functionality is not available in HeatWave.

  Workaround: Use non-ENUM for BETWEEN clause.

- Error number: 1158; Symbol: HW_ER_UNSUPPORTED_ARG_TYPES_FOR_FUNC

  Message: Unsupported argument types, ('%s, '%s'), for function, [%s].

  Description: The specific functionality is not available in HeatWave.

  Workaround: Use supported data types for arguments. Refer to HeatWave documentation for function usage and argument data types.

- Error number: 1159; Symbol: HW_ER_UNSUPPORTED_FUNC_ON_NON_VECTOR_TYPE

  Message: Function, [%s], requires a vector type column.

  Description: The specific functionality is not available in HeatWave.

  Workaround: Use vector type column as argument of the function. Refer to HeatWave documentation for function usage and argument data types.

- Error number: 1160; Symbol: HW_ER_UNSUPPORTED_EXPR_ITEM

  Message: Expr, [%s], is not supported.

  Description: The specific functionality is not available in HeatWave.

  Workaround: Use supported type for expression. Refer to HeatWave documentation for function, operator, expression usage and argument data types.

- Error number: 1161; Symbol: HW_ER_UNSUPPORTED_CUSTOM_AGGREGATION

  Message: Custom aggregation function, [%s], is not supported.

  Description: The support for custom aggregation function is not available in HeatWave.

  Workaround: Use the existing aggregation functions to express the functionality of a user-defined function.

- Error number: 1162; Symbol: HW_ER_UNSUPPORTED_DISTANCE_METRIC

  Message: DISTANCE metric, [%s] is not supported (supported metrics are: DOT, COSINE, EUCLIDEAN).

  Description: The specific functionality is not available in HeatWave.

  Workaround: Use supported DISTANCE metric. Refer to HeatWave documentation for function, operator, expression usage and argument data types.

- Error number: 1163; Symbol: HW_ER_NO_HW_PLAN_GENERATED

  Message: All plans were rejected by HeatWave secondary engine.

  Description: Query offload to HeatWave secondary engine failed as no suitable HeatWave query plan was generated.

  Workaround: Retry the query with use_secondary_engine session variable set to ON or contact support for further assistance.

- Error number: 1164; Symbol: HW_ER_PARTITION_NOT_LOADED

  Message: Accessed partition has not been loaded - %s.

  Description: The execution of query by HeatWave needs the partition to be loaded.

  Workaround: Load the partition into HeatWave.

- Error number: 1165; Symbol: HW_ER_UNSUPPORTED_LOAD_POOL

  Message: Schema %s.%s is in SNAPSHOT pool, explicit partition selection requires TRANSACTIONAL pool.

  Description: A partition selective scan cannot be offloaded if the table is not loaded to TRANSACTIONAL pool.

  Workaround: Set the table pool to TRANSACTIONAL and reload the table.

- Error number: 1166; Symbol: HW_ER_TABLE_NOT_PARTITIONED

  Message: Schema %s.%s is not logically partitioned.

  Description: A partition selective scan cannot be offloaded if the table is not logically partitioned.

  Workaround: Verify that the logical partitions are properly defined and are loaded in HeatWave.

- Error number: 1167; Symbol: HW_ER_COLUMN_NOT_LOADED

  Message: Column `%s`.`%s`.`%s` is not loaded to HeatWave.

  Description: All columns from the table accessed by the query must be loaded to HeatWave.

  Workaround: Reload the table.

- Error number: 1168; Symbol: HW_ER_UNSUPPORTED_TYPE_IN_OBY

  Message: Expr, [%s], in ORDER BY list is not supported.

  Description: The expression in the ORDER BY list might be incorrect.

  Workaround: Check and resolve any other expression related error.

- Error number: 1169; Symbol: HW_ER_UNSUPPORTED_TYPE_IN_ON_CLAUSE

  Message: ON clause, [%s], is not supported.

  Description: The specific functionality of ON clause is not available in HeatWave.

  Workaround: Either remove the ON clause or simplify the expression.

- Error number: 1170; Symbol: HW_ER_UNSUPPORTED_GBY_TRANSFORMATION

  Message: Error during GROUP BY transformation - %s.

Description: A problem occurred during the logical transformation of the query.

Workaround: Retry the query. If the issue persists, try rewriting the query to work around the error.

- Error number: 1171; Symbol: HW_ER_TOO_MANY_CUBE_ARGS

  Message: CUBE with distinct aggregate functions and GROUP_CONCAT with ORDER BY is supported for a maximum of five elements in the GROUP BY list.

  Description: HeatWave supports finite number of arguments in CUBE argument.

  Workaround: Limit the number of keys to five in GROUP BY clause.

- Error number: 1172; Symbol: HW_ER_REQUESTED_ABORT

  Message: Query is aborted/interrupted.

  Description: Query was aborted during logical transformation.

  Workaround: Retry the query. If the issue persists, try rewriting the query to work around the error.

- Error number: 1173; Symbol: HW_ER_UNSUPPORTED_SESSION_PROTOCOL

  Message: The session protocol is not supported.

  Description: The specific functionality is not available in HeatWave.

  Workaround: Do not use unsupported protocol.

- Error number: 1174; Symbol: HW_ER_QUERY_EXEC_TASK_ABORTED

  Message: Error during HeatWave query execution - %s (task [%s] has been aborted).

  Description: Problem occured during execution of query.

  Workaround: Retry the query. If the issue persists, try rewriting the query to work around the error.

- Error number: 1175; Symbol: HW_ER_UNSUPPORTED_RECURSIVE_COMMON_TABLE_EXPRESSIONS

  Message: Recursive common table expressions are not supported in HeatWave.

  Description: The specific functionality is not available in HeatWave.

  Workaround: Transform a common table expression into a non-recursive structure.

- Error number: 1176; Symbol: HW_ER_UNSUPPORTED_ARITHMETIC_AGGREGATION_ON_VARCHAR_COLS

  Message: Arithmetic aggregation on VARCHAR columns is not supported in [%s].

  Description: The specific functionality is not available in HeatWave.

  Workaround: Use supported aggregation on VARCHAR columns.

- Error number: 1177; Symbol: HW_ER_UNSUPPORTED_FUNC_OVER_DICT_ENCODED_COLS

  Message: Function, [%s], on dictionary encoded columns, is not supported.

  Description: The specific functionality is not available in HeatWave.

  Workaround: Use varlen encoded column in function. Refer to HeatWave documentation for function usage and argument data types.

- Error number: 1178; Symbol: HW_ER_BOOTSTRAP_NOT_ON

  Message: Query cannot be executed in HeatWave when rapid_bootstrap is not ON.

  Description: HeatWave query execution is disabled when the system variable rapid_bootstrap is not set to ON.

  Workaround: Retry the query with rapid_bootstrap system variable set to ON.

- Error number: 1179; Symbol: HW_ER_UNSUPPORTED_OPERATOR_WITH_ARGS_IN_FUNC

  Message: Operator, [%s], is not supported on arguments, ['%s', '%s'], in function, [%s].

  Description: The specific functionality is not available in HeatWave.

  Workaround: Refer to HeatWave documentation for operator and function usage and argument data types.

- Error number: 1180; Symbol: HW_ER_THREAD_STACK_OVERRUN_SECONDARY

  Message: Preventing potential stack overflow in %s

  Description: Unexpected internal stackoverflow while processing the query.

  Workaround: An internal error occurred due to stackoverflow while processing the query, please contact support to discuss and resolve the issue.

- Error number: 1181; Symbol: HW_ER_UNSUPPORTED_ARG_IN_FUNC

  Message: Unsupported argument, [%s], in function, [%s].

  Description: An unsupported argument was passed to a function in HeatWave.

  Workaround: Refer to HeatWave documentation for operator and function usage and argument data types.

- Error number: 1182; Symbol: HW_ER_EXPORT_TO_JSON_FORMAT

  Message: Error during export to JSON format - %s.

  Description: An error was encountered during JSON export in HeatWave.

  Workaround: Refer to HeatWave documentation for supported types for JSON export.

- Error number: 1183; Symbol: HW_ER_UNSUPPORTED_ITEM_FOR_EXPORT

  Message: Unsupported item for export - %s.

  Description: An error was encountered during export in HeatWave.

  Workaround: Refer to HeatWave documentation for supported types for export.

- Error number: 1184; Symbol: HW_ER_UNSUPPORTED_AI_MODE_OPERATION

  Message: Unsupported AI Mode Operation - %s.

  Description: Unsupported operation found while running query in HeatWave AI mode.

  Workaround: Refer to HeatWave documentation for supported queries in AI mode.

11.1.3 MySQL HeatWave AutoML Error Messages

Each error message includes an error number, SQLSTATE value, and message string, as described in Error Message Sources and Elements.

- Error number: `ML001016`; SQLSTATE: `HY000`

  Message: Only classification, regression, and forecasting tasks are supported.

  Example: `ERROR HY000: ML001016: Only classification, regression, and forecasting tasks are supported.`

  Check the `task` option in the `ML_TRAIN` call to ensure that it is specified correctly.

- Error number: `ML001031`; SQLSTATE: `HY000`

  Message: Running as a classification task. % classes have less than % samples per class, and cannot be trained on. Maybe it should be trained as a regression task instead of a classification task. Or the task ran on the default setting - classification, due to an incorrect JSON task argument.

  Example: `ERROR HY000: ML001031: Running as a classification task. 189 classes have less than 5 samples per class, and cannot be trained on. Maybe it should be trained as a regression task instead of a classification task. Or the task ran on the default setting - classification, due to an incorrect JSON task argument.`

  If a classification model is intended, add more samples to the data to increase the minority class count; that is, add more rows with the under-represented target column value. If a classification model was not intended, run `ML_TRAIN` with the regression task option.

- Error number: `ML001051`; SQLSTATE: `HY000`

  Message: One or more rows contain all NaN values. Imputation is not possible on such rows.

  Example: `ERROR HY000: ML001051: One or more rows contain all NaN values. Imputation is not possible on such rows.`

  MySQL does not support NaN values. Replace with NULL.

- Error number: `ML001052`; SQLSTATE: `HY000`

  Message: All columns are dropped. They are constant, mostly unique, or have a lot of missing values!

  Example: `ERROR HY000: ML001052: All columns are dropped. They are constant, mostly unique, or have a lot of missing values!`

  `ML_TRAIN` ignores columns with certain characteristics such as columns missing more than 20% of values and columns containing the same single value. See Section 6.5.1, “Prepare Data”.

- Error number: `ML001053`; SQLSTATE: `HY000`

  Message: Unlabeled samples detected in the training data. (Values in target column can not be NULL).

  Example: `ERROR HY000: ML001053: Unlabeled samples detected in the training data. (Values in target column can not be NULL).`

  Training data must be labeled. See Section 6.5.1, “Prepare Data”.

- Error number: `ML003000`; SQLSTATE: `HY000`

Message: Number of offloaded datasets has reached the limit!

Example: `ERROR HY000: ML003000: Number of offloaded datasets has reached the limit!`

- Error number: `ML003011`; SQLSTATE: `HY000`

  Message: Columns of provided data need to match those used for training. Provided - ['%', '%', '%'] vs Trained - ['%', '%'].

  Example: `ERROR HY000: ML003011: Columns of provided data need to match those used for training. Provided - ['petal length', 'petal width', 'sepal length', 'sepal width'] vs Trained - ['petal length', 'sepal length', 'sepal width'].`

  The input data columns do not match the columns of training dataset used to train the model. Compare the input data to the training data to identify the discrepancy.

- Error number: `ML003012`; SQLSTATE: `HY000`

  Message: The table (%) is NULL or has not been loaded.

  Example: `ERROR HY000: ML003012: The table (mlcorpus.iris_train) is NULL or has not been loaded.`

  There is no data in the specified table.

- Error number: `ML003014`; SQLSTATE: `HY000`

  Message: The size of model generated is larger than the maximum allowed.

  Example: `ERROR HY000: ML003014: The size of model generated is larger than the maximum allowed.`

  Models greater than 4 GB in size are not supported.

- Error number: `ML003015`; SQLSTATE: `HY000`

  Message: The input column types do not match the column types of dataset which the model was trained on. ['%', '%'] vs ['%', '%'].

  Example: `ERROR HY000: ML003015: The input column types do not match the column types of dataset which the model was trained on. ['numerical', 'numerical', 'categorical', 'numerical'] vs ['numerical', 'numerical', 'numerical', 'numerical'].`

- Error number: `ML003016`; SQLSTATE: `HY000`

  Message: Missing argument \"row_json\" in input JSON -> dict_keys(['%', '%']).

  Example: `ERROR HY000: ML003016: Missing argument \"row_json\" in input JSON -> dict_keys(['operation', 'user_name', 'table_name', 'schema_name', 'model_handle']).`

- Error number: `ML003017`; SQLSTATE: `HY000`

  Message: The corresponding value of row_json should be a string!

  Example: `ERROR HY000: ML003017: The corresponding value of row_json should be a string!`

- Error number: `ML003018`; SQLSTATE: `HY000`

Message: The corresponding value of row_json is NOT a valid JSON!

Example: `ERROR HY000: ML003018: The corresponding value of row_json is NOT a valid JSON!`

- Error number: `ML003019`; SQLSTATE: `HY000`

  Message: Invalid data for the metric (%). Score could not be computed.

  Example: `ERROR HY000: ML003019: Invalid data for the metric (roc_auc). Score could not be computed.`

  The scoring metric is legal and supported, but the data provided is not suitable to calculate such a score. For example: ROC_AUC for multi-class classification. Try a different scoring metric.

- Error number: `ML003020`; SQLSTATE: `HY000`

  Message: Unsupported scoring function (%) for current task (%).

  Example: `ERROR HY000: ML003020: Unsupported scoring function (accuracy) for current task (regression).`

  The scoring metric is legal and supported, but the task provided is not suitable to calculate such a score; for example: Using the `accuracy` metric for a `regression` model.

- Error number: `ML003021`; SQLSTATE: `HY000`

  Message: Cannot train a regression task with a non-numeric target column.

  Example: `ERROR HY000: ML003021: Cannot train a regression task with a non-numeric target column.`

  `ML_TRAIN` was run with the regression task type on a training dataset with a non-numeric target column. Regression models require a numeric target column.

- Error number: `ML003022`; SQLSTATE: `HY000`

  Message: At least 2 target classes are needed for classification task

  Example: `ERROR HY000: ML003022: At least 2 target classes are needed for classification task.`

  `ML_TRAIN` was run with the classification task type on a training dataset where the target column did not have at least two possible values.

- Error number: `ML003023`; SQLSTATE: `3877 (HY000)`

  Message: Unknown option given. Allowed options for training are: ['task', 'model_list', 'exclude_model_list', 'optimization_metric', 'exclude_column_list', 'datetime_index', 'endogenous_variables', 'exogenous_variables', 'positive_class', 'users', 'items', 'user_columns', 'item_columns'].

  Example: `ERROR 3877 (HY000): ML003023: Unknown option given. Allowed options for training are: ['task', 'model_list', 'exclude_model_list', 'optimization_metric', 'exclude_column_list', 'datetime_index', 'endogenous_variables', 'exogenous_variables', 'positive_class', 'users', 'items', 'user_columns', 'item_columns'].`

  The `ML_TRAIN` call specified an unknown option.

- Error number: `ML003024`; SQLSTATE: `HY000`

Message: Not enough available memory, unloading any RAPID tables will help to free up memory.

Example: `ERROR HY000: ML003024: Not enough available memory, unloading any RAPID tables will help to free up memory.`

There is not enough memory on the MySQL HeatWave Cluster to perform the operation. Try unloading data that was loaded for analytics to free up space.

The recommended node shape for MySQL HeatWave AutoML functionality is MySQL HeatWave.256GB. The MySQL HeatWave.16GB node shape might not have enough memory to train the model with large data sets. If this error message appears with the smaller node shape and MySQL HeatWave AutoML, use the larger shape instead.

- Error number: `ML003027`; SQLSTATE: `3877 (HY000)`

  Message: JSON attribute (item_columns) must be in JSON_ARRAY type.

  Example: `ERROR 3877 (HY000): ML003027: JSON attribute (item_columns) must be in JSON_ARRAY type.`

  Specify the `item_columns` JSON attribute as a JSON array.

- Error number: `ML003027`; SQLSTATE: `3877 (HY000)`

  Message: JSON attribute (user_columns) must be in JSON_ARRAY type.

  Example: `ERROR 3877 (HY000): ML003027: JSON attribute (user_columns) must be in JSON_ARRAY type.`

  Specify the `user_columns` JSON attribute as a JSON array.

- Error number: `ML003039`; SQLSTATE: `HY000`

  Message: Not all user specified columns are present in the input table - missing columns are {%}.

  Example: `ERROR HY000: ML003039: Not all user specified columns are present in the input table - missing columns are {C4}.`

  The syntax includes a column that is not available.

- Error number: `ML003047`; SQLSTATE: `HY000`

  Message: All columns cannot be excluded. User provided exclude_column_list is ['%', '%'].

  Example: `ERROR HY000: ML003047: All columns cannot be excluded. User provided exclude_column_list is ['C0', 'C1', 'C2', 'C3'].`

  The syntax includes an `exclude_column_list` that attempts to exclude too many columns.

- Error number: `ML003048`; SQLSTATE: `HY000`

  Message: exclude_column_list JSON attribute must be of JSON_ARRAY type.

  Example: `ERROR HY000: ML003048: exclude_column_list JSON attribute must be of JSON_ARRAY type.`

  The syntax includes a malformed `JSON_ARRAY` for the `exclude_column_list`.

- Error number: `ML003048`; SQLSTATE: `HY000`

  Message: include_column_list JSON attribute must be of JSON_ARRAY type.

  Example: `ERROR HY000: ML003048: include_column_list JSON attribute must be of JSON_ARRAY type.`

  The syntax includes a malformed `JSON_ARRAY` for the `include_column_list`.

- Error number: `ML003049`; SQLSTATE: `HY000`

  Message: One or more columns in include_column_list ([%]) does not exist. Existing columns are (['%', '%']).

  Example: `ERROR HY000: ML003049: One or more columns in include_column_list ([C15]) does not exist. Existing columns are (['C0', 'C1', 'C2', 'C3']).`

  The syntax includes an `include_column_list` that expects a column that does not exist.

- Error number: `ML003050`; SQLSTATE: `HY000`

  Message: include_column_list must be a subset of exogenous_variables for forecasting task.

  Example: `ERROR HY000: ML003050: include_column_list must be a subset of exogenous_variables for forecasting task.`

  The syntax for a forecasting task includes an `include_column_list` that expects one or more columns that are not defined by `exogenous_variables`.

- Error number: `ML003052`; SQLSTATE: `HY000`

  Message: Target column provided % is one of the independent variables used to train the model [%, %, %].

  Example: `ERROR HY000: ML003052: Target column provided LSTAT is one of the independent variables used to train the model [RM, RAD, LSTAT].`

  The syntax defines a `target_column_name` that is one of the independent variables used to train the model.

- Error number: `ML003053`; SQLSTATE: `HY000`

  Message: datetime_index must be specified by the user for forecasting task and must be a column in the training table.

  Example: `ERROR HY000: ML003053: datetime_index must be specified by the user for forecasting task and must be a column in the training table.`

  The syntax for a forecasting task must include `datetime_index`, and this must be a column in the training table.

- Error number: `ML003054`; SQLSTATE: `HY000`

  Message: endogenous_variables must be specified by the user for forecasting task and must be column(s) in the training table.

  Example: `ERROR HY000: ML003054: endogenous_variables must be specified by the user for forecasting task and must be column(s) in the training table.`

  The syntax for a forecasting task must include the `endogenous_variables` option, and these must be a column or columns in the training table.

- Error number: `ML003055`; SQLSTATE: `HY000`

Message: endogenous_variables / exogenous_variables option must be of JSON_ARRAY type.

Example: `ERROR HY000: ML003055: endogenous_variables / exogenous_variables option must be of JSON_ARRAY type.`

The syntax for a forecasting task includes `endogenous_variables` or `exogenous_variables` that do not have valid `JSON` format.

- Error number: `ML003056`; SQLSTATE: `HY000`

  Message: exclude_column_list cannot contain any of endogenous or exogenous variables for forecasting task.

  Example: `ERROR HY000: ML003056: exclude_column_list cannot contain any of endogenous or exogenous variables for forecasting task.`

  The syntax for a forecasting task includes `exclude_column_list` that contains columns that are also in `endogenous_variables` or `exogenous_variables`.

- Error number: `ML003057`; SQLSTATE: `HY000`

  Message: endogenous and exogenous variables may not have any common columns for forecasting task.

  Example: `ERROR HY000: ML003057: endogenous and exogenous variables may not have any common columns for forecasting task.`

  The syntax for a forecasting task includes `endogenous_variables` and `exogenous_variables`, and they have one or more columns in common.

- Error number: `ML003058`; SQLSTATE: `HY000`

  Message: Can not train a forecasting task with non-numeric endogenous_variables column(s).

  Example: `ERROR HY000: ML003058: Can not train a forecasting task with non-numeric endogenous_variables column(s).`

  The syntax for a forecasting task includes `endogenous_variables` and some of the columns are not defined as numeric.

- Error number: `ML003059`; SQLSTATE: `HY000`

  Message: User provided list of models ['ThetaForecaster', 'ETSForecaster', 'SARIMAXForecaster', 'ExpSmoothForecaster'] does not include any supported models for the task. Supported models for the given task and table are ['DynFactorForecaster', 'VARMAXForecaster'].

  Example: `ERROR HY000: ML003059: User provided list of models ['ThetaForecaster', 'ETSForecaster', 'SARIMAXForecaster', 'ExpSmoothForecaster'] does not include any supported models for the task. Supported models for the given task and table are ['DynFactorForecaster', 'VARMAXForecaster'].`

  The syntax for a forecasting task includes multivariate `endogenous_variables`, but the provided models only support univariate `endogenous_variables`.

- Error number: `ML003060`; SQLSTATE: `HY000`

  Message:: endogenous_variables may not contain repeated column names ['%1', '%2', '%1'].

  Example: `ERROR HY000: ML003060: endogenous_variables may not contain repeated column names ['wind', 'solar', 'wind'].`

The syntax for a forecasting task includes `endogenous_variables` with a repeated column.

- Error number: `ML003061`; SQLSTATE: `HY000`

  Message: exogenous_variables may not contain repeated column names ['consumption', 'wind_solar', 'consumption'].

  Example: `ERROR HY000: ML003061: exogenous_variables may not contain repeated column names ['consumption', 'wind_solar', 'consumption'].`

  The syntax for a forecasting task includes `exogenous_variables` with a repeated column.

- Error number: `ML003062`; SQLSTATE: `HY000`

  Message: endogenous_variables argument must not be NULL.

  Example: `ERROR HY000: ML003062: endogenous_variables argument must not be NULL.`

  The syntax for a forecasting task includes `endogenous_variables` with a NULL argument.

- Error number: `ML003063`; SQLSTATE: `HY000`

  Message: exogenous_variables argument must not be NULL when provided by user.

  Example: `ERROR HY000: ML003063: exogenous_variables argument must not be NULL when provided by user.`

  The syntax for a forecasting task includes user provided `exogenous_variables` with a NULL argument.

- Error number: `ML003064`; SQLSTATE: `HY000`

  Message: Cannot exclude all models.

  Example: `ERROR HY000: ML003064: Cannot exclude all models.`

  The syntax for a forecasting task must include at least one model.

- Error number: `ML003065`; SQLSTATE: `HY000`

  Message: Prediction table cannot have overlapping datetime_index with train table when exogenous_variables are used. It can only forecast into future.

  Example: `ERROR HY000: ML003065: Prediction table cannot have overlapping datetime_index with train table when exogenous_variables are used. It can only forecast into future.`

  The syntax for a forecasting task includes `exogenous_variables` and the prediction table contains values in the `datetime_index` column that overlap with values in the `datetime_index` column in the training table.

- Error number: `ML003066`; SQLSTATE: `HY000`

  Message: datetime_index for test table must not have missing dates after the last date in training table. Please ensure test table starts on or before 2034-01-01 00:00:00. Currently, start date in the test table is 2036-01-01 00:00:00.

  Example: `ERROR HY000: ML003066: datetime_index for test table must not have missing dates after the last date in training table. Please ensure test table starts on or before 2034-01-01 00:00:00. Currently, start date in the test table is 2036-01-01 00:00:00.`

The syntax for a forecasting task includes a prediction table that contains values in the `datetime_index` column that leave a gap to the values in the `datetime_index` column in the training table.

- Error number: `ML003067`; SQLSTATE: `HY000`

  Message: datetime_index for forecasting task must be between year 1678 and 2261.

  Example: `ERROR HY000: ML003067: datetime_index for forecasting task must be between year 1678 and 2261.`

  The syntax for a forecasting task includes values in a `datetime_index` column that are outside the date range from 1678 to 2261.

- Error number: `ML003068`; SQLSTATE: `HY000`

  Message: Last date of datetime_index in the training table 2151-01-01 00:00:00 plus the length of the table 135 must be between year 1678 and 2261.

  Example: `ERROR HY000: ML003068: Last date of datetime_index in the training table 2151-01-01 00:00:00 plus the length of the table 135 must be between year 1678 and 2261.`

  The syntax for a forecasting task includes a prediction table that has too many rows, and the values in the `datetime_index` column would be outside the date range from 1678 to 2261.

- Error number: `ML003070`; SQLSTATE: `3877 (HY000)`

  Message: For recommendation tasks both user and item column names should be provided.

  Example: `ERROR 3877 (HY000): ML003070: For recommendation tasks both user and item column names should be provided.`

- Error number: `ML003071`; SQLSTATE: `HY000`

  Message: contamination must be numeric value greater than 0 and less than 0.5.

  Example: `ERROR HY000: ML003071: contamination must be numeric value greater than 0 and less than 0.5.`

- Error number: `ML003071`; SQLSTATE: `3877 (HY000)`

  Message: item_columns can not contain repeated column names ['C4', 'C4'].

  Example: `ERROR 3877 (HY000): ML003071: item_columns can not contain repeated column names ['C4', 'C4'].`

- Error number: `ML003071`; SQLSTATE: `3877 (HY000)`

  Message: user_columns can not contain repeated column names ['C4', 'C4'].

  Example: `ERROR 3877 (HY000): ML003071: user_columns can not contain repeated column names ['C4', 'C4'].`

- Error number: `ML003072`; SQLSTATE: `HY000`

  Message: Can not use more than one threshold method.

  Example: `ERROR HY000: ML003072: Can not use more than one threshold method.`

- Error number: `ML003072`; SQLSTATE: `3877 (HY000)`

  Message: Target column C3 can not be specified as a user or item column.

  Example: `ERROR 3877 (HY000): ML003072: Target column C3 can not be specified as a user or item column.`

- Error number: `ML003073`; SQLSTATE: `HY000`

  Message: topk must be an integer value between 1 and length of the table, inclusively (1 <= topk <= 20).

  Example: `ERROR HY000: ML003073: topk must be an integer value between 1 and length of the table, inclusively (1 <= topk <= 20).`

- Error number: `ML003073`; SQLSTATE: `3877 (HY000)`

  Message: The users and items columns should be different.

  Example: `ERROR 3877 (HY000): ML003073: The users and items columns should be different.`

- Error number: `ML003074`; SQLSTATE: `HY000`

  Message: threshold must be a numeric value between 0 and 1, inclusively (0 <= threshold <= 1).

  Example: `ERROR HY000: ML003074: threshold must be a numeric value between 0 and 1, inclusively (0 <= threshold <= 1).`

- Error number: `ML003074`; SQLSTATE: `3877 (HY000)`

  Message: Unsupported ML Operation for recommendation task.

  Example: `ERROR 3877 (HY000): ML003074: Unsupported ML Operation for recommendation task.`

- Error number: `ML003075`; SQLSTATE: `HY000`

  Message: Unknown option given. This scoring metric only allows for these options: ['topk'].

  Example: `ERROR HY000: ML003075: Unknown option given. This scoring metric only allows for these options: ['topk'].`

- Error number: `ML003075`; SQLSTATE: `3877 (HY000)`

  Message: Unknown option given. Allowed options for recommendations are ['recommend', 'top'].

  Example: `ERROR 3877 (HY000): ML003075: Unknown option given. Allowed options for recommendations are ['recommend', 'top'].`

- Error number: `ML003076`; SQLSTATE: `HY000`

  Message: ML_EXPLAIN, ML_EXPLAIN_ROW and ML_EXPLAIN_TABLE are not supported for anomaly_detection task.

  Example: `ERROR HY000: ML003076: ML_EXPLAIN, ML_EXPLAIN_ROW and ML_EXPLAIN_TABLE are not supported for anomaly_detection task.`

- Error number: `ML003076`; SQLSTATE: `3877 (HY000)`

  Message: The recommend option should be provided when a value for topk is assigned.

  Example: `ERROR 3877 (HY000): ML003076: The recommend option should be provided when a value for topk is assigned.`

- Error number: `ML003077`; SQLSTATE: `HY000`

  Message: topk must be provided as an option when metric is set as precision_at_k.

  Example: `ERROR HY000: ML003077: topk must be provided as an option when metric is set as precision_at_k.`

- Error number: `ML003077`; SQLSTATE: `3877 (HY000)`

  Message: Unknown recommend value given. Allowed values for recommend are ['ratings', 'items', 'users'].

  Example: `ERROR 3877 (HY000): ML003077: Unknown recommend value given. Allowed values for recommend are ['ratings', 'items', 'users'].`

- Error number: `ML003078`; SQLSTATE: `HY000`

  Message: anomaly_detection only allows 0 (normal) and 1 (anomaly) for labels in target column with any metric used, and they have to be integer values.

  Example: `ERROR HY000: ML003078: anomaly_detection only allows 0 (normal) and 1 (anomaly) for labels in target column with any metric used, and they have to be integer values.`

- Error number: `ML003078`; SQLSTATE: `3877 (HY000)`

  Message: Should not provide a value for topk when the recommend option is set to ratings.

  Example: `ERROR 3877 (HY000): ML003078: Should not provide a value for topk when the recommend option is set to ratings.`

- Error number: `ML003079`; SQLSTATE: `3877 (HY000)`

  Message: Provided value for option topk is not a strictly positive integer.

  Example: `ERROR 3877 (HY000): ML003079: Provided value for option topk is not a strictly positive integer.`

- Error number: `ML003080`; SQLSTATE: `3877 (HY000)`

  Message: One or more rows contains NULL or empty values. Please provide inputs without NULL or empty values for recommendation.

  Example: `ERROR 3877 (HY000): ML003080: One or more rows contains NULL or empty values. Please provide inputs without NULL or empty values for recommendation.`

- Error number: `ML003081`; SQLSTATE: `3877 (HY000)`

  Message: Options should be NULL. Options are currently not supported for this task classification.

  Example: `ERROR 3877 (HY000): ML003081: options should be NULL. Options are currently not supported for this task classification.`

- Error number: `ML003082`; SQLSTATE: `3877 (HY000)`

  Message: All supported models are excluded, but at least one model should be included.

  Example: `ERROR 3877 (HY000): ML003082: All supported models are excluded, but at least one model should be included.`

- Error number: `ML003083`; SQLSTATE: `HY000`

  Message: Both user column name ['C3'] and item column name C0 must be provided as string.

  Example: `ERROR HY000: ML003083: Both user column name ['C3'] and item column name C0 must be provided as string.`

- Error number: `ML003105`; SQLSTATE: `3877 (HY000)`

  Message: Cannot recommend users to a user not present in the training table.

  Example: `ERROR: 3877 (HY000): ML003105: Cannot recommend users to a user not present in the training table.`

- Error number: `ML003106`; SQLSTATE: `3877 (HY000)`

  Message: Cannot recommend items to an item not present in the training table.

  Example: `ERROR 3877 (HY000): ML003106: Cannot recommend items to an item not present in the training table.`

- Error number: `ML003107`; SQLSTATE: `3877 (HY000)`

  Message: Users to users recommendation is not supported, please retrain your model.

  Example: `ERROR 3877 (HY000): ML003107: Users to users recommendation is not supported, please retrain your model.`

- Error number: `ML003108`; SQLSTATE: `3877 (HY000)`

  Message: Items to items recommendation is not supported, please retrain your model.

  Example: `ERROR 3877 (HY000): ML003108: Items to items recommendation is not supported, please retrain your model.`

- Error number: `ML003109`; SQLSTATE: `HY000`

  Message: Invalid Model format.

  Example: `HY000: ML003109: Invalid Model format.`

- Error number: `ML003111`; SQLSTATE: `HY000`

  Message: Unknown option given. Allowed options are ['batch_size'].

  Example: `ERROR HY000: ML003111: Unknown option given. Allowed options are ['batch_size'].`

- Error number: `ML003112`; SQLSTATE: `HY000`

  Message: NULL values are not supported for text columns.

  Example: `ERROR HY000: ML003112: NULL values are not supported for text columns.`

- Error number: `ML003114`; SQLSTATE: `HY000`

Message: Error while parsing text. One of the text columns only contains stop words like the, is, and, a, an, in, has, etc.

Example: `ERROR HY000: ML003114: Error while parsing text. One of the text columns only contains stop words like the, is, and, a, an, in, has, etc.`

- Error number: `ML003115`; SQLSTATE: `HY000`

  Message: Empty input table after applying threshold.

  Example: `ERROR HY000: ML003115: Empty input table after applying threshold.`

- Error number: `ML003116`; SQLSTATE: `HY000`

  Message: The feedback_threshold option can only be set for implicit feedback.

  Example: `ERROR HY000: ML003116: The feedback_threshold option can only be set for implicit feedback.`

- Error number: `ML003117`; SQLSTATE: `HY000`

  Message: The remove_seen option can only be used with the following recommendation ['items', 'users', 'users_to_items', 'items_to_users'].

  Example: `ERROR HY000: ML003117: The remove_seen option can only be used with the following recommendation ['items', 'users', 'users_to_items', 'items_to_users'].`

- Error number: `ML003118`; SQLSTATE: `HY000`

  Message: The remove_seen option must be set to either True or False. Provided *`input`*.

  Example: `ERROR HY000: ML003118: The remove_seen option must be set to either True or False. Provided input.`

- Error number: `ML003119`; SQLSTATE: `HY000`

  Message: The feedback option must either be set to explicit or implicit. Provided *`input`*.

  Example: `ERROR HY000: ML003119: The feedback option must either be set to explicit or implicit. Provided input.`

- Error number: `ML003120`; SQLSTATE: `HY000`

  Message: The input table needs to contain strictly more than one unique item.

  Example: `ERROR HY000: ML003120: The input table needs to contain strictly more than one unique item.`

- Error number: `ML003121`; SQLSTATE: `HY000`

  Message: The input table needs to contain at least one unknown or negative rating.

  Example: `ERROR HY000: ML003121: The input table needs to contain at least one unknown or negative rating.`

- Error number: `ML003122`; SQLSTATE: `HY000`

  Message: The feedback_threshold option must be numeric.

  Example: `ERROR HY000: ML003122: The feedback_threshold option must be numeric.`

- Error number: `ML003123`; SQLSTATE: `HY000`

  Message: User and item columns should contain strings.

  Example: `ERROR HY000: ML003123: User and item columns should contain strings.`

- Error number: `ML003124`; SQLSTATE: `HY000`

  Message: Calculation for precision_at_k metric could not complete because there are no recommended items.

  Example: `ERROR HY000: ML003124: Calculation for precision_at_k metric could not complete because there are no recommended items.`

- Error number: `ML004002`; SQLSTATE: `HY000`

  Message: Output format of onnx model is not supported (output_name={%},output_shape={%},output_type={%}).

  Example: `HY000: ML004002: Output format of onnx model is not supported (output_name={%},output_shape={%},output_type={%}).`

- Error number: `ML004003`; SQLSTATE: `HY000`

  Message: This ONNX model only supports fixed batch size=%.

  Example: `HY000: ML004003: This ONNX model only supports fixed batch size=%.`

- Error number: `ML004005`; SQLSTATE: `HY000`

  Message: The type % in data_types_map is not supported.

  Example: `HY000: ML004005: The type % in data_types_map is not supported.`

- Error number: `ML004006`; SQLSTATE: `HY000`

  Message: ML_SCORE is not supported for an onnx model that does not support batch inference.

  Example: `HY000: ML004006: ML_SCORE is not supported for an onnx model that does not support batch inference.`

- Error number: `ML004007`; SQLSTATE: `HY000`

  Message: ML_EXPLAIN is not supported for an onnx model that does not support batch inference.

  Example: `HY000: ML004007: ML_EXPLAIN is not supported for an onnx model that does not support batch inference.`

- Error number: `ML004008`; SQLSTATE: `HY000`

  Message: onnx model input type=% is not supported! Providing the appropriate types map using 'data_types_map' in model_metadata may resolve the issue.

  Example: `HY000: ML004008: onnx model input type=% is not supported! Providing the appropriate types map using 'data_types_map' in model_metadata may resolve the issue.`

- Error number: `ML004009`; SQLSTATE: `HY000`

  Message: Input format of onnx model is not supported (onnx_input_name={%}, expected_input_shape={%}, expected_input_type={%}, data_shape={%}).

  Example: `HY000: ML004009: Input format of onnx model is not supported (onnx_input_name={%}, expected_input_shape={%}, expected_input_type={%}, data_shape={%}).`

- Error number: `ML004010`; SQLSTATE: `HY000`

  Message: Output being sparse tensor with batch size > 1 is not supported.

  Example: `HY000: ML004010: Output being sparse tensor with batch size > 1 is not supported.`

- Error number: `ML004010`; SQLSTATE: `3877 (HY000)`

  Message: Received data exceeds maximum allowed length 943718400.

  Example: `ERROR 3877 (HY000): ML004010: Received data exceeds maximum allowed length 943718400.`

- Error number: `ML004011`; SQLSTATE: `HY000`

  Message: predictions_name=% is not valid.

  Example: `HY000: ML004011: predictions_name=% is not valid.`

- Error number: `ML004012`; SQLSTATE: `HY000`

  Message: prediction_probabilities_name=% is not valid.

  Example: `HY000: ML004012: prediction_probabilities_name=% is not valid.`

- Error number: `ML004013`; SQLSTATE: `HY000`

  Message: predictions_name should be provided when task=regression and onnx model generates more than one output.

  Example: `HY000: ML004013: predictions_name should be provided when task=regression and onnx model generates more than one output.`

- Error number: `ML004014`; SQLSTATE: `HY000`

  Message: Missing expected JSON key (%)

  Example: `ERROR HY000: ML004014: Missing expected JSON key (schema_name).`

- Error number: `ML004014`; SQLSTATE: `HY000`

  Message: Incorrect labels_map. labels_map should include the key %

  Example: `HY000: ML004014: Incorrect labels_map. labels_map should include the key %`

- Error number: `ML004015`; SQLSTATE: `HY000`

  Message: Expected JSON string type value for key (%)

  Example: `ERROR HY000: ML004015: Expected JSON string type value for key (schema_name).`

- Error number: `ML004015`; SQLSTATE: `HY000`

Message: When task=classification, if the user does not provide prediction_probabilities_name for the onnx model, ML_EXPLAIN method=% will not be supported.

Example: `HY000: ML004015: When task=classification, if the user does not provide prediction_probabilities_name for the onnx model, ML_EXPLAIN method=% will not be supported.` % can be "shap", "fast_shap" or "partial_dependence"

- Error number: `ML004016`; SQLSTATE: `HY000`

  Message: Invalid base64-encoded ONNX string.

  Example: `HY000: ML004016: Invalid base64-encoded ONNX string.`

- Error number: `ML004017`; SQLSTATE: `HY000`

  Message: Invalid ONNX model.

  Example: `HY000: ML004017: Invalid ONNX model.`

- Error number: `ML004018`; SQLSTATE: `HY000`

  Message: Parsing JSON arg: Invalid value. failed!

  Example: `ERROR HY000: ML004018: Parsing JSON arg: Invalid value. failed!`

- Error number: `ML004018`; SQLSTATE: `HY000`

  Message: There are issues in running inference session for the onnx model. This might have happened due to inference on inputs with incorrect names, shapes or types.

  Example: `HY000: ML004018: There are issues in running inference session for the onnx model. This might have happened due to inference on inputs with incorrect names, shapes or types.`

- Error number: `ML004019`; SQLSTATE: `HY000`

  Message: Expected JSON object type value for key (%)

  Example: `ERROR HY000: ML004019: Expected JSON object type value for key (JSON root).`

- Error number: `ML004019`; SQLSTATE: `HY000`

  Message: The computed predictions do not have the right format. This might have happened because the provided predictions_name is not correct.

  Example: `HY000: ML004019: The computed predictions do not have the right format. This might have happened because the provided predictions_name is not correct.`

- Error number: `ML004020`; SQLSTATE: `HY000`

  Message: Operation was interrupted by the user.

  Example: `ERROR HY000: ML004020: Operation was interrupted by the user.`

  If a user-initiated interruption, `Ctrl-C`, is detected during the first phase of MySQL HeatWave AutoML model and table load where a MySQL parallel scan is used in the MySQL HeatWave plugin to read data as of MySQL database and send it to the MySQL HeatWave Cluster, error messaging is handled by the MySQL parallel scan function and directed to `ERROR 1317 (70100): Query execution was interrupted.`. The `ERROR 1317 (70100)` message is reported to the client instead of the `ML004020` error message.

- Error number: `ML004020`; SQLSTATE: `HY000`

  Message: The computed prediction probabilities do not have the right format. This might have happened because the provided prediction_probabilities_name is not correct.

  Example: `HY000: ML004020: The computed prediction probabilities do not have the right format. This might have happened because the provided prediction_probabilities_name is not correct.`

- Error number: `ML004021`; SQLSTATE: `HY000`

  Message: The onnx model and dataset do not match. The onnx model's input=% is not a column in the dataset.

  Example: `HY000: ML004021: The onnx model and dataset do not match. The onnx model's input=% is not a column in the dataset.`

- Error number: `ML004022`; SQLSTATE: `HY000`

  Message: The user does not have access privileges to %.

  Example: `ERROR HY000: ML004022: The user does not have access privileges to ml.foo.`

- Error number: `ML004022`; SQLSTATE: `HY000`

  Message: Labels in y_true and y_pred should be of the same type. Got y_true=% and y_pred=YYY. Make sure that the predictions provided by the classifier coincides with the true labels.

  Example: `HY000: ML004022: Labels in y_true and y_pred should be of the same type. Got y_true=% and y_pred=YYY. Make sure that the predictions provided by the classifier coincides with the true labels.`

- Error number: `ML004026`; SQLSTATE: `HY000`

  Message: A column (%) with an unsupported column type (%) detected!

  Example: `ERROR HY000: ML004026: A column (D1) with an unsupported column type (BINARY) detected!`

- Error number: `ML004051`; SQLSTATE: `HY000`

  Message: Invalid operation.

  Example: `ERROR HY000: ML004051: Invalid operation.`

- Error number: `ML004999`; SQLSTATE: `HY000`

  Message: Error during Machine Learning.

  Example: `ERROR HY000: ML004999: Error during Machine Learning.`

- Error number: `ML006006`; SQLSTATE: `45000`

  Message: target_column_name should be NULL or empty.

  Example: `ERROR 45000: ML006006: target_column_name should be NULL or empty.`

- Error number: `ML006017`; SQLSTATE: `45000`

  Message: model_handle already exists in the Model Catalog.

  Example: `45000: ML006017: model_handle already exists in the Model Catalog.`

- Error number: `ML006020`; SQLSTATE: `45000`

  Message: model_metadata should be a JSON object.

  Example: `45000: ML006020: model_metadata should be a JSON object.`

- Error number: `ML006021`; SQLSTATE: `45000`

  Message: contamination has to be passed with anomaly_detection task.

  Example: `ERROR 45000: ML006021: contamination has to be passed with anomaly_detection task.`

- Error number: `ML006022`; SQLSTATE: `45000`

  Message: Unsupported task.

  Example: `ERROR 45000: ML006022: Unsupported task.`

- Error number: `ML006023`; SQLSTATE: `45000`

  Message: "No model object found" will be raised.

  Example: `45000: ML006023: "No model object found" will be raised.`

- Error number: `ML006027`; SQLSTATE: `1644 (45000)`

  Message: Received results exceed `max_allowed_packet`. Please increase it or lower input options value to reduce result size.

  Example: `ERROR 1644 (45000): ML006027: Received results exceed `max_allowed_packet`. Please increase it or lower input options value to reduce result size.`

- Error number: `ML006029`; SQLSTATE: `45000`

  Message: model_handle is not Ready.

  Example: `45000: ML006029: model_handle is not Ready.`

- Error number: `ML006030`; SQLSTATE: `45000`

  Message: onnx_inputs_info must be a json object.

  Example: `ERROR 45000: ML006030: onnx_inputs_info must be a json object.`

- Error number: `ML006031`; SQLSTATE: `45000`

  Message: Unsupported format.

  Example: `45000: ML006031: Unsupported format.`

- Error number: `ML006031`; SQLSTATE: `45000`

  Message: onnx_outputs_info must be a json object.

  Example: `ERROR 45000: ML006031: onnx_outputs_info must be a json object.`

- Error number: `ML006032`; SQLSTATE: `45000`

  Message: data_types_map must be a json object.

  Example: `ERROR 45000: ML006032: data_types_map must be a json object.`

- Error number: `ML006033`; SQLSTATE: `45000`

  Message: labels_map must be a json object.

  Example: `ERROR 45000: ML006033: labels_map must be a json object.`

- Error number: `ML006034`; SQLSTATE: `45000`

  Message: onnx_outputs_info must be provided for task=classification.

  Example: `ERROR 45000: ML006034: onnx_outputs_info must be provided for task=classification.`

- Error number: `ML006035`; SQLSTATE: `45000`

  Message: onnx_outputs_info must only be provided for classification and regression tasks.

  Example: `ERROR 45000: ML006035: onnx_outputs_info must only be provided for classification and regression tasks.`

- Error number: `ML006036`; SQLSTATE: `45000`

  Message: % is not a valid key in onnx_inputs_info.

  Example: `ERROR 45000: ML006036: % is not a valid key in onnx_inputs_info.`

- Error number: `ML006037`; SQLSTATE: `45000`

  Message: % is not a valid key in onnx_outputs_info.

  Example: `ERROR 45000: ML006037: % is not a valid key in onnx_outputs_info.`

- Error number: `ML006038`; SQLSTATE: `45000`

  Message: For task=classification, at least one of predictions_name and prediction_probabilities_name must be provided.

  Example: `ERROR 45000: ML006038: For task=classification, at least one of predictions_name and prediction_probabilities_name must be provided.`

- Error number: `ML006039`; SQLSTATE: `45000`

  Message: prediction_probabilities_name must only be provided for task=classification.

  Example: `ERROR 45000: ML006039: prediction_probabilities_name must only be provided for task=classification.`

- Error number: `ML006040`; SQLSTATE: `45000`

  Message: predictions_name must not be an empty string.

  Example: `ERROR 45000: ML006040: predictions_name must not be an empty string.`

- Error number: `ML006041`; SQLSTATE: `45000`

  Message: prediction_probabilities_name must not be an empty string.

  Example: `ERROR 45000: ML006041: prediction_probabilities_name must not be an empty string.`

- Error number: `ML006042`; SQLSTATE: `45000`

  Message: labels_map must only be provided for task=classification.

Example: `ERROR 45000: ML006042: labels_map must only be provided for task=classification.`

- Error number: `ML006043`; SQLSTATE: `45000`

  Message: When labels_map is provided, prediction_probabilities_name must also be provided.

  Example: `ERROR 45000: ML006043: When labels_map is provided, prediction_probabilities_name must also be provided.`

- Error number: `ML006044`; SQLSTATE: `45000`

  Message: When labels_map is provided, predictions_name must not be provided.

  Example: `ERROR 45000: ML006044: When labels_map is provided, predictions_name must not be provided.`

- Error number: `ML006045`; SQLSTATE: `45000`

  Message: ML_SCORE is not supported for a % task.

  Example: `ERROR 45000: ML006045: ML_SCORE is not supported for a % task.`

- Error number: `ML006046`; SQLSTATE: `45000`

  Message: ML_EXPLAIN is not supported for a % task.

  Example: `ERROR 45000: ML006046: ML_EXPLAIN is not supported for a % task.`

- Error number: `ML006047`; SQLSTATE: `45000`

  Message: onnx_inputs_info must only be provided when format='ONNX'.

  Example: `ERROR 45000: ML006047: onnx_inputs_info must only be provided when format='ONNX'.`

- Error number: `ML006048`; SQLSTATE: `45000`

  Message: onnx_outputs_info must only be provided when format='ONNX'.

  Example: `ERROR 45000: ML006048: onnx_outputs_info must only be provided when format='ONNX'.`

- Error number: `ML006049`; SQLSTATE: `45000`

  Message: The length of a key provided in onnx_inputs_info should not be greater than 32 characters.

  Example: `ERROR 45000: ML006049: The length of a key provided in onnx_inputs_info should not be greater than 32 characters.`

- Error number: `ML006050`; SQLSTATE: `45000`

  Message: The length of a key provided in onnx_outputs_info should not be greater than 32 characters.

  Example: `ERROR 45000: ML006050: The length of a key provided in onnx_outputs_info should not be greater than 32 characters.`

- Error number: `ML006051`; SQLSTATE: `45000`

  Message: Invalid ONNX model.

  Example: `ERROR 45000: ML006051: Invalid ONNX model.`

- Error number: `ML006052`; SQLSTATE: `45000`

  Message: Input table is empty. Please provide a table with at least one row.

  Example: `ERROR 45000: ML006052: Input table is empty. Please provide a table with at least one row.`

- Error number: `ML006053`; SQLSTATE: `45000`

  Message: Insufficient access rights. Grant user with correct privileges (SELECT, DROP, CREATE, INSERT, ALTER) on input schema.

  Example: `ERROR 45000: ML006053: Insufficient access rights. Grant user with correct privileges (SELECT, DROP, CREATE, INSERT, ALTER) on input schema.`

- Error number: `ML006054`; SQLSTATE: `45000`

  Message: Input table already contains a column named `_4aad19ca6e_pk_id`. Please provide an input table without such column.

  Example: `ERROR 45000: ML006054: Input table already contains a column named `_4aad19ca6e_pk_id`. Please provide an input table without such column.`

- Error number: `ML006055`; SQLSTATE: `45000`

  Message: Options must be a JSON_OBJECT.

  Example: `ERROR 45000: ML006055: Options must be a JSON_OBJECT.`

- Error number: `ML006056`; SQLSTATE: `45000`

  Message: batch_size must be an integer between 1 and %.

  Example: `ERROR 45000: ML006056: batch_size must be an integer between 1 and %.`

- Error number: `ML006070`; SQLSTATE: `45000`

  Message: model_list is currently not supported for anomaly_detection.

  Example: `ERROR 45000: ML006070: model_list is currently not supported for anomaly_detection.`

- Error number: `ML006071`; SQLSTATE: `45000`

  Message: exclude_model_list is currently not supported for anomaly_detection.

  Example: `ERROR 45000: ML006071: exclude_model_list is currently not supported for anomaly_detection.`

- Error number: `ML006072`; SQLSTATE: `45000`

  Message: optimization_metric is currently not supported for anomaly_detection.

  Example: `ERROR 45000: ML006072: optimization_metric is currently not supported for anomaly_detection.`

- Message: Error while releasing memory from HeatWave ML. Please restart the cluster.

  Example: `ERROR HY000: Error while releasing memory from HeatWave ML. Please restart the cluster.`

  To resolve this error, restart the MySQL HeatWave Cluster. To learn more, see the following:

  - Starting, Stopping, or Restarting a MySQL HeatWave Cluster in the *MySQL HeatWave on OCI Service Guide*.
  - Starting, Stopping, or Restarting a MySQL HeatWave Cluster in the *MySQL HeatWave on AWS Service Guide*
  - Restarting a Database in the *MySQL HeatWave for Azure Service Guide*.

- Message: Failure while processing command for allocating memory to HeatWave ML.

  Example: `ERROR HY000: Failure while processing command for allocating memory to HeatWave ML.`

  To resolve this error, restart the MySQL HeatWave Cluster. To learn more, see the following:

  - Starting, Stopping, or Restarting a MySQL HeatWave Cluster in the *MySQL HeatWave on OCI Service Guide*.
  - Starting, Stopping, or Restarting a MySQL HeatWave Cluster in the *MySQL HeatWave on AWS Service Guide*
  - Restarting a Database in the *MySQL HeatWave for Azure Service Guide*.

- Message: Failed to allocate memory to HeatWave ML. Please consider reducing the memory usage of the HeatWave cluster.

  Example: `ERROR HY000: Failed to allocate memory to HeatWave ML. Please consider reducing the memory usage of the HeatWave cluster.`

  To resolve this error, free up memory in the cluster by removing data that is no longer needed.

11.1.4 MySQL HeatWave GenAI Issues

This section describes some commonly encountered issues and errors for MySQL HeatWave GenAI and their workarounds.

- *Issue:* When you try to verify whether the vector embeddings were correctly loaded, if you see a message which indicates that the vector embeddings or table did not load in HeatWave, then it could be due one of the following reasons:

  - The task that loads the vector embeddings into the vector store table might still be running.

    Workaround: Check the task status by using the query that was printed by the `VECTOR_STORE_LOAD` routine:

    ```
    SELECT * from mysql_task_management.task_status where id = TaskID;
    ```

    Or, to see the log messages, check the task logs table:

    ```
    SELECT * from mysql_task_management.task_log where task_id = TaskID;
    ```

    Replace *`TaskID`* with the ID for the task which was printed by the `VECTOR_STORE_LOAD` routine.

  - The folder you are trying to load might contain unsupported format files or the file that you are trying to load might be of an unsupported format.

    Workaround: The supported file formats are: PDF, TXT, PPT, HTML, and DOC.

    If you find unsupported format files, then try one of the following:

    - Delete the files with unsupported formats from the folder, and run the `VECTOR_STORE_LOAD` command again to load the vector embeddings into the vector store table again.

    - Move the files with supported formats to another folder, create a new PAR and run the `VECTOR_STORE_LOAD` command with the new PAR to load the vector embeddings into the vector store table again.

- *Issue:* the `VECTOR_STORE_LOAD` command fails unexpectedly.

  Workaround: Ensure that you use the `--sqlc` flag when you connect to your database system:

  ```
  mysqlsh -uAdmin -pPassword -hPrivateIP --sqlc
  ```

  Replace the following:

  - *`Admin`*: the admin name.
  - *`Password`*: the database system password.
  - *`PrivateIP`*: the private IP address of the database system.

  If you are still not able to ingest files using `VECTOR_STORE_LOAD` routine, then try Ingest Files Using Auto Parallel Load.

11.1.5 MySQL HeatWave Lakehouse Error Messages

See MySQL 26.7 Error Message Reference for Lakehouse error messages.

As of MySQL 9.2.2, a detailed summary of warnings is available when loading data to Lakehouse. This includes warnings related to the schema inference stage of loading data.

The summaries include the following:

- Warnings for `Lakehouse Load` and `Lakehouse Schema Inference`.
- The number of warnings.
- The code of each warning with a description.
- If warning messages were filtered, the number of warnings that were not recorded due to the filters.

Refer to the following samples of warning summaries.

A warning summary for `Lakehouse Load` that includes the number of filtered warnings that are not recorded for the summary.

```
Warning 6434    [WARNINGS SUMMARY] Lakehouse Load had 27 warning(s) out of which 27 were not recorded (due
Warning 6434    [WARNINGS SUMMARY] 22 warnings with code: 6053(ER_LH_WARN_EXCEEDS_MAX_TRUNCATING)
Warning 6434    [WARNINGS SUMMARY] 5 warnings with code: 6052(ER_LH_WARN_EXCEEDS_MIN_TRUNCATING)
```

A warning summary for `Lakehouse Schema Inference` with Autopilot that includes the number of filtered warnings that are not recorded for the summary.

```
{"message": "[WARNINGS SUMMARY] Lakehouse Schema Inference had 18 warning(s) out of which 1 were not record
{"message": "[WARNINGS SUMMARY] 16 warnings with code: 6062(ER_LH_COLUMN_MISMATCH_ERR)", "table_name": "lin
{"message": "[WARNINGS SUMMARY] 1 warnings with code: 6095(ER_LH_WARN_INFER_SKIPPED_LINES)", "table_name":
{"message": "[WARNINGS SUMMARY] 1 warnings with code: 6063(ER_LH_COLUMN_MAX_ERR)", "table_name": "lineitem"
```

As of MySQL 9.1.2, the following details are included in Lakehouse error and warning messages related to data files in the Object Storage bucket to help you understand them better:

- Byte offset text message is added to the existing error/warning messages in `csv` or JSON file formats.
- Error/warning position details are added to JSON data type related errors/warnings in `csv`, JSON, `parquet` or AVRO file formats.

- Row number and character position details are added to warning messages(Non strict mode) in `csv` and JSON file formats.
- Row number is added to warning messages(Non strict mode) in `parquet` or AVRO file formats.

Refer to the following sample error messages, for more information.

- Error message - Byte offset (17) is added to `csv` or JSON error message.

```
ERROR HY000: Column 1 of {object: "mtrtest/datalake_integer.test_3.tbl" in
namespace: "OCI_OBJECT_STORE_SOURCE_NAMESPACE", bucket:
"OCI_OBJECT_STORE_SOURCE_BUCKET"} at byte offset 17 : Invalid integer value.
```

- Warning message - Row number (13), character position (1), and byte offset (66) are added to `csv` or JSON warning message.

```
Warning 6057 Row 13, char 1, column 1 of {object: "mtrtest/datalake_int_basic/0.tbl" in
namespace: "OCI_OBJECT_STORE_SOURCE_NAMESPACE", bucket:
"OCI_OBJECT_STORE_SOURCE_BUCKET"} at byte offset 66 : Integer value was truncated.
```

- Warning message - Position (11), character (12), are added to `csv`, JSON, `parquet` or AVRO warning message.

```
Warning 6121 Row 20, column 3 of {object:
"datalake_avro_json/datalake_json_error.avro.tbl" in namespace:
"OCI_OBJECT_STORE_SOURCE_NAMESPACE", bucket: "OCI_OBJECT_STORE_SOURCE_BUCKET"} :
Invalid escape character in string at position 11(char 12) in this column.
```

Some Lakehouse error and warning messages include the URL to the external object. The URL can contain up to 1024 characters, and exceed the error message limit of 512 characters. As of MySQL 9.0.1-u1, the truncated message will include a MySQL command to access the full error message. For example:

- Error message:

```
ERROR HY000: Message was truncated, use
<SELECT data FROM performance_schema.error_log
WHERE thread_id=15 AND data LIKE 'ERR_LH_test.t1_long_name_6045%'
ORDER BY LOGGED DESC>
to obtain more detailed information about the warning/error
```

- Warning message:

```
Warning 6045 Message was truncated, use
<SELECT data FROM performance_schema.error_log
WHERE thread_id=15 AND data LIKE 'WARN_LH_test.t1_long_name_6045_0%'
ORDER BY LOGGED DESC>
to obtain more detailed information about the warning/error
```

As of MySQL 9.0.1-u1, it is possible to filter out Lakehouse warning messages from the console, MySQL Shell and error log following a load command. The console **Total Warnings** will include both the displayed and the filtered warnings count. This does not filter error messages.

Use the `lakehouse_filter_warning_codes_list` session variable to filter warning messages by error code. For example:

```
mysql> SET @@session.lakehouse_filter_warning_codes_list = '6017,5099,6018';
```

Use the `lakehouse_filter_warning_modes_list` session variable to filter warning messages by mode. For example:

```
mysql> SET @@session.lakehouse_filter_warning_modes_list =
       'NUMERIC_TRUNCATION,STRING_TRUNCATION,EMPTY_FILES,DUPLICATE_FILES';
```

The supported modes are:

- Mode: `AVRO_FILE_BLOCK_HEADER` filters the following warning messages:

    - ER_LH_AVRO_CANNOT_PARSE_HEADER
    - ER_LH_AVRO_HEADER_METADATA_ERR
    - ER_LH_FORMAT_HEADER_NO_MAGIC_BYTES
    - ER_LH_AVRO_HEADER_NO_SCHEMA
    - ER_LH_AVRO_INVALID_BLOCK_RECORD_COUNT
    - ER_LH_AVRO_INVALID_BLOCK_SIZE
    - ER_LH_AVRO_NO_CODEC_IN_HEADER
- Mode: DUPLICATE_FILES filters the following warning messages:
    - ER_LH_DUPLICATE_FILE
- Mode: EMPTY_FILES filters the following warning messages:
    - ER_LH_EMPTY_FILE
- Mode: EMPTY_MISSING_COLUMNS filters the following warning messages:
    - ER_LH_COLUMN_MISMATCH_ERR
    - ER_LH_COLUMN_MAX_ERR
    - ER_LH_WARN_COL_MISSING_NOT_NULLABLE
    - ER_LH_COL_IS_EMPTY_WARN
- Mode: INFER_SKIPPED filters the following warning messages:
    - ER_LH_INFER_FILE_HAS_NO_DATA
    - ER_LH_WARN_INFER_SKIPPED_FILES
    - ER_LH_WARN_INFER_SKIPPED_LINES
- Mode: MISSING_FILES filters the following warning messages:
    - ER_LH_NO_FILES_FOUND
- Mode: NULL_COLUMNS filters the following warning messages:
    - ER_LH_COL_NOT_NULLABLE
- Mode: NUMERIC_PARSING for the int, decimal, float, and double data types filters the following warning messages:
    - ER_LH_BAD_VALUE
    - ER_LH_DECIMAL_OOM_ERR
    - ER_LH_DECIMAL_UNKNOWN_ERR
    - ER_LH_PARQUET_DECIMAL_CONVERSION_ERR
    - ER_LH_REAL_IS_NAN
- Mode: NUMERIC_TRUNCATION for the int, decimal, float, and double data types filters the following warning messages:

    - ER_LH_DECIMAL_PRECISION_EXCEEDS_SCHEMA
    - ER_LH_EXCEEDS_MAX
    - ER_LH_EXCEEDS_MIN
    - ER_LH_OUT_OF_RANGE
    - ER_LH_WARN_DECIMAL_ROUNDING
    - ER_LH_WARN_EXCEEDS_MIN_TRUNCATING
    - ER_LH_WARN_EXCEEDS_MAX_TRUNCATING
    - ER_LH_WARN_TRUNCATED
- Mode: STRING_PARSING for the char, enum, text, and varchar data types filters the following warning messages:
    - ER_LH_BAD_VALUE
    - ER_LH_CANNOT_CONVERT_STRING
- Mode: STRING_TRUNCATION for the char, text, and varchar data types filters the following warning messages:
    - ER_LH_STRING_TOO_LONG
- Mode: TEMPORAL_PARSING for the date, datetime, time, and timestamp data types filters the following warning messages:
    - ER_LH_BAD_VALUE
    - ER_LH_DATETIME_FORMAT
- Mode: TEMPORAL_TRUNCATION for the date, datetime, time, and timestamp data types filters the following warning messages:
    - ER_LH_WARN_TRUNCATED
- Mode: PARQUET_SCHEMA filters the following warning messages:
    - ER_LH_PARQUET_CANNOT_LOCATE_SCHEMA
    - ER_LH_PARQUET_SCHEMA_MISMATCH

11.2 Limitations

The sections in this topic list the limitations of different MySQL HeatWave features.

11.2.1 MySQL HeatWave Limitations

This section lists functionality that is not supported by MySQL HeatWave. It is not an exhaustive list with respect to data types, functions, operators, and SQL modes. For data types, functions, operators, and SQL modes that are supported by MySQL HeatWave, see Section 4.2.1, “Supported Data Types for MySQL HeatWave”, Section 5.3, “Supported Functions and Operators”, and Section 5.2, “Supported SQL Modes”. If a particular data type, function, operator, or SQL mode does not appear in those tables and lists, it should be considered unsupported.

11.2.1.1 Data Type Limitations

- Review Column Limits for data types.

- Spatial data types are unsupported. See Spatial Data Types.
- `VARBINARY` is unsupported.
- Before MySQL 9.6.1, `BINARY` is unsupported. As of MySQL 9.6.1, it is supported.
- `BINARY` has the following limitations:
  - It is not supported for MySQL HeatWave Lakehouse.
  - Does not support dictionary encoding.
- Decimal values with a precision greater than 18 in expression operators are unsupported, with the exception of the following:
  - Arithmetic operators, see: Section 5.3.2, “Arithmetic Operators”.
  - `CAST()`
  - `COALESCE()`
  - `CASE`
  - `IF()`
  - `NULLIF()`
  - `ABS()`
  - `CEILING()`
  - `FLOOR()`
  - `ROUND()`
  - `TRUNCATE()`
  - `GREATEST()`
  - `LEAST()`
- The following `ENUM` type columns are unsupported:
  - `ENUM` type columns as part of a `UNION`, `EXCEPT`, `EXCEPT ALL`, `INTERSECT`, or `INTERSECT ALL` `SELECT` list or as a `JOIN` key, except when used inside a supported expression.
  - `ENUM` type columns as part of a non-top level `UNION ALL SELECT` list or as a `JOIN` key, except when used inside a supported expression.

- `ENUM` type support is limited to:
  - Comparison with string or numeric constants, and other numeric, non-temporal expressions (numeric columns, constants, and functions with a numeric result).
  - Comparison operators (<, <=, <=>, =, >=, >, and `BETWEEN`) with numeric arguments.
  - Comparison operators (=, <=>, and <>) with string constants.
  - *`enum_col`* `IS [NOT] {NULL|TRUE|FALSE}`
  - The `IN()` function in combination with numeric arguments (constants, functions, or columns) and string constants.
  - `COUNT()`, `SUM()`, and `AVG()` aggregation functions on `ENUM` columns. The functions operate on the numeric index value, not the associated string value.
  - `CAST(`*`enum_col`* `AS {[N]CHAR [(X)]|SIGNED|UNSIGNED|FLOAT|DOUBLE|DECIMAL [(M,N)]})`. The numeric index value is cast, not the associated string value.
  - `CAST(`*`enum_col`*`) AS {[N]CHAR}` is supported only in the `SELECT` list and when it is not nested in another expression.
- You cannot run queries for materialized views that project `ENUM` type columns.
- Temporal types are supported only with strict SQL mode.

11.2.1.2 Bulk Ingest Data to MySQL Server Limitations

- MySQL HeatWave on OCI does not support `LOAD DATA` with `ALGORITHM=BULK`.
- MySQL HeatWave on AWS does support `LOAD DATA` with `ALGORITHM=BULK`, but does not support the `INFILE` and `URL` clauses.
- For versions earlier than MySQL 9.4.0, tables must have a primary key, and the primary key must not have a prefix index. For MySQL 9.4.0 and later, primary keys are not required.
- `LOAD DATA` with `ALGORITHM=BULK` has the following limitations:
  - It locks the target table exclusively and does not allow other operations on the table.
  - It does not support automatic rounding or truncation of the input data. It fails if the input data requires rounding or truncation in order to be loaded.
  - It does not support temporary tables.
  - It is atomic but not transactional. It commits any transaction that is already running. On failure the `LOAD DATA` statement is completely rolled back.
  - It cannot execute when the target table is explicitly locked by a `LOCK TABLES` statement.
- The target table for `LOAD DATA` with `ALGORITHM=BULK` has the following limitations:
  - It must be empty. The state of the table should be as though it has been freshly created. If the table has instantly added/dropped column, call TRUNCATE before calling `LOAD DATA` with `ALGORITHM=BULK`.
  - It must not be partitioned.
  - It must not contain secondary indexes. As of MySQL 9.2.0, `LOAD DATA` with `ALGORITHM=BULK` does support secondary indexes.
  - It must be in a file_per_tablespace, and must not be in a shared tablespace.

- It must have the default row format, `ROW_FORMAT=DYNAMIC`. Use `ALTER TABLE` to make any changes to the table after `LOAD DATA` with `ALGORITHM=BULK`.
- It must not contain virtual or stored generated columns.
- It must not contain foreign keys.
- It must not contain `CHECK` constraints.
- It must not contain triggers.
- It is not replicated to other nodes.
- It must not use a secondary engine. Set the secondary engine after `LOAD DATA` with `ALGORITHM=BULK`. See: Section 4.2.6.2, “Define the Secondary Engine”.

11.2.1.3 Change Propagation Limitations

- Cascading changes triggered by a foreign key constraint.

11.2.1.4 Functions and Operator Limitations

- As of MySQL 9.2.2, the maximum output width of the following functions is 65,332 bytes:
  - `JSON_ARRAYAGG()`
  - `JSON_OBJECTAGG()`
  - `JSON_ARRAY()`
  - `JSON_OBJECT()`
  - `JSON_EXTRACT()`
- Bit functions and operators.
- `COALESCE()` as a `JOIN` predicate.
- `CONVERT_TZ()` with named time zones before MySQL 8.4.0. Only datetime values are supported. Rewrite queries that use named time zones with equivalent datetime values. For example:

  ```
  mysql> SELECT CONVERT_TZ(O_ORDERDATE, 'UTC','EST') FROM tpch.orders;
  ```

  Rewrite as:

  ```
  mysql> SELECT CONVERT_TZ(O_ORDERDATE, '+00:00','-05:00') FROM tpch.orders;
  ```

- Only `UNIX_TIMESTAMP()` and `FROM_UNIXTIME()` functions support a *`timezone`* value specified as an offset from UTC in the form of *`[H]H:MM`* and prefixed with a + or -. For example:

  ```
  mysql> SET time_zone = timezone;
  ```

  For information about time zone offsets, see MySQL Server Time Zone Support.
- `COUNT(NULL)` in cases where it is used as an input argument for non-aggregate operators.
- All known limitations for `COUNT(DISTINCT)` also apply to `HLL()`.
- Full-text search functions.
- XML, Spatial, and other domain specific functions.
- The following JSON functions:

    - `JSON_TABLE()` queries that include a `LEFT JOIN` clause require a primary key column specified for the base or intermediate table. Without a primary key column, these queries are not offloaded.

- Loadable Functions.
- `GROUP_CONCAT()` with:
    - Dictionary-encoded string columns, `RAPID_COLUMN=ENCODING=SORTED`. See Section 5.8.1.2, “Dictionary Encoding”.
    - As of MySQL 9.0.1-u1, the GROUP_CONCAT() function supports CUBE and ROLLUP non-primitive GROUP BY transformation options.
    - As of MySQL 9.0.1-u1 `COUNT(DISTINCT)` function supports CUBE, ROLLUP and WITH ROLLUP options.
- `COUNT(DISTINCT)` with:
    - As of MySQL 9.0.1-u1 `COUNT(DISTINCT)` function supports CUBE, ROLLUP and WITH ROLLUP options.
- A `CASE` control flow operator or `IF()` function that contains columns not within an aggregation function and not part of the `GROUP BY` key.
- String functions and operators on columns that are not `VARLEN` encoded. See Section 5.8.1.4, “Encoding String Columns”.
- The `AVG()` aggregate function with enumeration and temporal data types.
- The following aggregate functions with the enumeration data type:
    - `STD()`
    - `STDDEV()`
    - `STDDEV_POP()`
    - `STDDEV_SAMP()`
    - `SUM()`
    - `VARIANCE()`
    - `VAR_POP()`
    - `VAR_SAMP()`

    With the exception of `SUM()`, the same aggregate functions within a semi-join predicate due to the nondeterministic nature of floating-point results and potential mismatches. For example, the following use is not supported:

    ```
    mysql> SELECT FROM A WHERE a1 IN (SELECT VAR_POP(b1) FROM B);
    ```

    The same aggregate functions with numeric data types other than those supported by MySQL HeatWave. See Section 4.2.1, “Supported Data Types for MySQL HeatWave”.
- `WEEK(date[,mode])` does not support the `default_week_format` system variable.

    To use the `mode` argument, the *`mode`* value must be defined explicitly.

11.2.1.5 Index Hint and Optimizer Hint Limitations

- Index and optimizer hints. See Index Hints, and Optimizer Hints.

- Semijoin strategies other than `FIRSTMATCH`.

  MySQL attempts to enforce the `FIRSTMATCH` strategy and ignores all other semijoin strategies specified explicitly as subquery optimizer hints. However, MySQL may still select the `DUPSWEEDOUT` semijoin strategy during `JOIN` order optimization, even if an equivalent plan could be offered using the `FIRSTMATCH` strategy. A plan that uses the `DUPSWEEDOUT` semijoin strategy would produce incorrect results if executed on MySQL HeatWave.

  For general information about subquery optimizer hints, see Subquery Optimizer Hints.

11.2.1.6 Join Limitations

- Antijoins, with the exception of supported `IN` and `EXISTS` antijoin variants listed below.
- `EXISTS` semijoins and antijoins are supported in the following variants only:
  - `SELECT ... WHERE ... EXISTS (...)`
  - `SELECT ... WHERE ... EXISTS (...) IS TRUE`
  - `SELECT ... WHERE ... EXISTS (...) IS NOT FALSE`
  - `SELECT ... WHERE ... NOT EXISTS (...) IS FALSE`
  - `SELECT ... WHERE ... NOT EXISTS (...) IS NOT TRUE`

  Depending on transformations and optimizations performed by MySQL, other variants of `EXISTS` semijoins may or may not be offloaded.
- `IN` semijoins and antijoins other than the following variants:
  - `SELECT ... WHERE ... IN (...)`
  - `SELECT ... WHERE ... IN (...) IS TRUE`
  - `SELECT ... WHERE ... NOT IN (...) IS FALSE`

  Depending on transformations and optimizations performed by MySQL, other variants of `IN` semijoins may or may not be offloaded.
- A query with a supported semijoin or antijoin condition may be rejected for offload due to how MySQL optimizes and transforms the query.
- Semijoin and antijoin queries use the best plan found after evaluating the first 10000 possible plans, or after investigating 10000 possible plans since the last valid plan.

  The plan evaluation count is reset to zero after each derived table, after an outer query, and after each subquery. The plan evaluation limit is required because the `DUPSWEEDOUT` join strategy, which is not supported by MySQL HeatWave, may be used as a fallback strategy by MySQL during join order optimization (for related information, see FIRSTMATCH). The plan evaluation limit prevents too much time being spent evaluating plans in cases where MySQL generates numerous plans that use the `DUPSWEEDOUT` semijoin strategy.
- Outer join queries without an equality condition defined for the two tables.
- Some outer join queries with `IN ... EXISTS` sub-queries (semi-joins) in the `ON` clause.

11.2.1.7 Partition Selection Limitations

Before MySQL 9.1.0, MySQL HeatWave does not support explicit partition selection.

As of MySQL 9.1.0, MySQL HeatWave supports InnoDB partitions with the following limitations:

- MySQL HeatWave cannot load partitions with Auto Parallel Load.
- MySQL HeatWave cannot load partitions from a snapshot of the database.
- MySQL HeatWave cannot load partitions from a table that contains dictionary-encoded columns.
- MySQL HeatWave can unload up to 1,000 partitions in a single statement. Use additional statements to unload more than 1,000 partitions.
- MySQL HeatWave cannot load or unload partitions during the recovery process.

11.2.1.8 Variable Limitations

- The `sql_select_limit` as a global variable.

  It is only supported as a session variable.

- `time_zone` and `timestamp` variable settings are not passed to MySQL HeatWave when queries are offloaded. MySQL 8.4.0-u2 removes this limitation.

11.2.1.9 Other Limitations

- Most non-default DB System SQL modes.

  For a list of supported SQL modes, see Section 5.2, “Supported SQL Modes”.

- The `gb18030_chinese_ci` character set and collation.
- The `WITH ROLLUP` modifier in `GROUP BY` clauses in the following cases:
  - In queries that contain distinct aggregations.
  - In queries that contain duplicate `GROUP BY` keys.
- Cursors inside stored programs are not supported before MySQL 9.0.0, see: Cursors.
- `UNION ALL` queries with an `ORDER BY` or `LIMIT` clause, between dictionary-encoded columns, or between `ENUM` columns.

  `EXCEPT`, `EXCEPT ALL`, `INTERSECT`, `INTERSECT ALL`, and `UNION` queries with or without an `ORDER BY` or `LIMIT` clause, between dictionary-encoded columns, or between `ENUM` columns.

  `EXCEPT`, `EXCEPT ALL`, `INTERSECT`, `INTERSECT ALL`, `UNION` and `UNION ALL` subqueries with or without an `ORDER BY` or `LIMIT` clause, between dictionary-encoded columns, between `ENUM` columns, or specified in an `IN` or `EXISTS` clause.

- Comparison predicates, `GROUP BY`, `JOIN`, and so on, if the key column is `DOUBLE PRECISION`.
- Queries with an impossible `WHERE` condition (queries known to have an empty result set).

  For example, the following query is not offloaded:

  ```
  mysql> SELECT AVG(c1) AS value FROM t1 WHERE c1 IS NULL;
  ```

- Primary keys with column prefixes.
- Virtual generated columns.
- Queries that are executed as part of a trigger.
- Queries that call a stored function are not supported before MySQL 9.5.0.

  As of MySQL 9.5.0, queries that call a stored function are supported with the following limitations:

  - The query runs with forced secondary engine only:

```
SET use_secondary_engine=forced;
```

- The function definition must not contain `IF` or `ELSE` statements.
- The function definition must not contain `CREATE`, `INSERT`, `ALTER`, `UPDATE`, or `DELETE` statements.
- In the function definition, the `SQL SECURITY` characteristic must be set to `INVOKER`. It must not be set to `DEFINER` For more information, see The SQL SECURITY Characteristic.
- The function must use the same `sql_mode` variable setting as the current session.
- The function must use the same Connection Character Set and Collation System Variables variables setting as the current session.
- Condition handlers are not supported on the secondary engine. For more information, see Condition Handling.
- Stored function must not be a part of a prepared statement. For more information, see Prepared Statements.
- Only the following statements must be used in the function definition:
  - `SET` statement: the `SET` statement must use only local variables. The set value can either be an expression or a subquery.
  - `SELECT ... INTO` statement: only local variable can be used.
  - `RETURN` statement: The `RETURN` statement must be followed by an expression that does not contain a subquery. If you need to use a subquery, set a local variable to use the subquery.

Stored functions for MySQL HeatWave can be defined as shown below:

```
CREATE [DEFINER = username] FUNCTION
 [IF NOT EXISTS] function_name ([func_parameter[,...]])
 RETURNS datatype [characteristic ...] routine_body
func_parameter:
 param_name datatype
datatype:
 Any valid MySQL data type
characteristic:  {
 COMMENT 'string'
 | LANGUAGE SQL
 | [NOT] DETERMINISTIC
 | { CONTAINS SQL | READS SQL DATA} }
SQL SECURITY INVOKER
routine_body:
 SQL routine with supported statements
```

For example:

```
CREATE FUNCTION get_earliest_departure(from_airport_id SMALLINT)
RETURNS DATETIME
SQL SECURITY INVOKER
DETERMINISTIC
BEGIN
    DECLARE earliest_departure DATETIME;

    SELECT MIN(departure)
      INTO earliest_departure
      FROM flight
     WHERE `from` = from_airport_id;

    RETURN earliest_departure;
END
```

- Queries that are executed as part of a stored program.
- Queries that are part of a multi-statement transaction.
- Materialized views are not supported for versions earlier than MySQL 9.5.0. See Section 5.4.7, “Query Views” for information on non-materialized views. As of MySQL 9.5.0, materialized views are supported. See Section 5.4.8, “Query Materalized Views”.

  You cannot run queries for materialized views that project `DICTIONARY` encoded columns.
- Partial query offload for regular `SELECT` queries.

  If all elements of the query are supported, the entire query is offloaded; otherwise, the query is executed on the DB System by default.

  MySQL HeatWave supports `CREATE TABLE ... SELECT` and `INSERT ... SELECT` statements where only the `SELECT` portion of the operation is offloaded to MySQL HeatWave. See Section 5.4, “Run Queries”.
- Named time zones are not supported before MySQL 8.4.0.
- Row widths in intermediate and final query results that exceed 4MB in size.

  A query that exceeds this row width limit is not offloaded to MySQL HeatWave for processing.
- Consecutive filter operations on derived tables.

  For example, the following query is not supported:

```
mysql> SELECT * FROM (SELECT * FROM t1 WHERE x < 7) tt1,
          (SELECT * FROM t1 WHERE x < y) tt2
          WHERE  tt1.x > 5 AND tt1.x = tt2.x;
```

  The query uses a filter for table `tt1` in the table scan of table `t1` (`x < 7`) followed by a consecutive filter on table `tt1` (`tt1.x > 5`) in the `WHERE` clause.
- Recursive common table expressions.
- Operations involving `ALTER TABLE` such as loading, unloading, or recovering data when MySQL Server is running in `SUPER_READ_ONLY` mode.

  MySQL Server is placed in `SUPER_READ_ONLY` mode when MySQL Server disk space drops below a set amount for a specific duration. For information about thresholds that control this behavior and how to disable `SUPER_READ_ONLY` mode, see Resolving SUPER_READ_ONLY and OFFLINE_MODE Issue in the *MySQL HeatWave on OCI Service Guide*.

11.2.1.10 Asynchronous Task Limitations

- Asynchronous tasks run as MySQL Events on the server. Therefore, only those specific SQL operations that are supported inside events are supported for creating asynchronous tasks.
- Any SQL operation that depends on the return value of the `SESSION_USER()` or `USER()` function might behave differently when run asynchronously. This is because MySQL Events are executed by a reserved internal user called `event_scheduler`, and the `SESSION_USER()` and `USER()` functions return the value `event_scheduler` instead of the actual session user name.
- When an asynchronous task attempts to obtain a lock on a database resource, but it unable to get it as another transaction holds a conflicting lock, an error is generated and the following message is displayed: `Cannot acquire lock. Try again later.`

11.2.2 MySQL HeatWave Lakehouse Limitations

The following limitations apply to Lakehouse. For MySQL HeatWave limitations, see Section 11.2.1, “MySQL HeatWave Limitations”.

11.2.2.1 Lakehouse Limitations for all File Formats

MySQL HeatWave Lakehouse has the following limitations for all file formats. Review Other Limitations for all MySQL HeatWave query-related limitations.

Unsupported Items

MySQL HeatWave Lakehouse does not support the following:

- DML statements:
  - `INSERT`
  - `UPDATE`
  - `DELETE`
  - `REPLACE`
- The `CREATE TABLESPACE` statement.
- The following options for the `CREATE TABLE` statement:
  - `AUTOEXTEND_SIZE`
  - `AVG_ROW_LENGTH`
  - `CHECKSUM`
  - `COMPRESS`
  - `CONNECTION`
  - `DATADIR`
  - `DELAY_KEY_WRITE`
  - `ENCRYPT`
  - `INDEXDIR`
  - `INSERT_METHOD`
  - `KEY_BLOCK_SIZE`
  - `MAX_ROWS`
  - `MIN_ROWS`
  - `PACK_KEYS`
  - `PASSWORD`
  - `ROW_FORMAT`
  - `STATS_AUTO_RECALC`
  - `STATS_PERSISTENT`
  - `STATS_SAMPLE_PAGES`
  - `UNION`
- The default expression for a column definition for the `CREATE TABLE` statement.

- Creating temporary tables.
- Creating `AUTO_INCREMENT` columns.

  Therefore, Lakehouse is not compatible with `REQUIRE_TABLE_PRIMARY_KEY_CHECK` = `GENERATE`.
- Creating triggers.
- Running `ANALYZE TABLE`.
- Running `ALTER TABLE` statements that construct indexes, `ADD` or `DROP` columns, or add enforced check constraints.
- `SELECT` statements without `RAPID` as the secondary engine.
- Hidden columns.
- Index construction.
- Keys with column prefixes.
- Running `CREATE TABLE` or `ALTER TABLE` statement containing a `STORAGE` clause in the column attributes of a Lakehouse table.
- The use of `CURRENT_TIMESTAMP()`, `CURRENT_TIMESTAMP` or `NOW()` as a default value for a timestamp column. Also an `UPDATE` statement with `CURRENT_TIMESTAMP()`, `CURRENT_TIMESTAMP` or `NOW()`. Enable `explicit_defaults_for_timestamp` to use `ALTER TABLE` and `CREATE TABLE` statements with Lakehouse tables that have a timestamp column.
- A zone map on a JSON column.
- Before MySQL 9.6.1, stored generated columns in the JSON file format, unstructured format, Delta Lake tables, or the `VECTOR` data type. As of MySQL 9.6.1, stored generated columns are supported for the JSON file format and Delta Lake tables.
- Stored generated columns over `NOT SECONDARY` columns.
- Stored generated columns over stored generated columns.

Other Limitations

MySQL HeatWave Lakehouse has the following limitations:

- As of MySQL 9.3.1, Lakehouse tables have the following column width limitations based on data type:
  - `MEDIUMTEXT`, `LONGTEXT`, and `JSON`: 4192192 bytes
  - `TEXT`: 65535 bytes
  - `CHAR`, `VARCHAR`, and `VECTOR`: 65532 bytes

  Note

  - Column width limitations apply after Lakehouse extracts and cleans data from files, and in the case of JSON data after converting it into JSON DOM representation internally.
  - The maximum row width is 4 MB. This includes the size of all the data and column metadata stored in the row. You cannot load a Lakehouse table that has multiple columns with a combined data size greater than 4 MB. This applies to intermediate relations and final query results as well.

  Before MySQL 9.3.1, the maximum column width is 65532 bytes.

- Do not create Lakehouse tables on the source DB in a replicated DB System if any of the replicas are outside MySQL HeatWave on OCI or MySQL HeatWave on AWS. This causes replication errors.
- Before MySQL 8.4.0-u2, a replication channel might fail if a MySQL HeatWave Cluster is added to a replica of a DB System, and later manually stopped.
- It is not possible to dump external tables using the MySQL Shell export utilities, such as `dumpInstance()`. External Lakehouse tables are not replicated to DB System storage and cannot be exported. To export DB System data from a Lakehouse enabled database, exclude the external tables with an `excludeTables` option.
- It is not possible to restore a backup from a Lakehouse enabled DB System to a standalone DB System.
- A Lakehouse enabled DB System can support a maximum of 512 nodes.
- Before MySQL 8.4.0, Lakehouse does not enforce any specified constraints on primary key, unique key, foreign key, and `CHECK` constraints. MySQL 8.4.0 removes this limitation for primary key and unique key constraints.
- Before MySQL 9.0.1-u1, the limit for the Lakehouse error message count is 100. As of MySQL 9.0.1-u1, Lakehouse supports `max_error_count`.
- The `lakehouse_filter_warning_codes_list` session variable has a limit of 50 codes and 250 characters.
- You can create up to 16 stored generated columns.
- The output of a stored generated column using `const` can be up to 1024 bytes.
- Specifying a metadata column as a `NOT SECONDARY` column generates an error.
- Specifying `SECONDARY LOAD` or `SECONDARY UNLOAD` for a partitioning column in an `ALTER TABLE` statement generates an error.
- Queries that access partitioned external tables when `use_secondary_engine=off` or when the table cannot be offloaded to MySQL HeatWave generates an error.

11.2.2.2 Lakehouse Limitations for the Avro File Format

MySQL HeatWave Lakehouse has the following limitations for Avro files.

- Avro file headers must be less than 1MB.
- The data in an Avro block must be no larger than 64MiB before applying any compression.
- Lakehouse only supports uncompressed Avro blocks, or Avro blocks compressed with Snappy or Deflate algorithms. Lakehouse does not support other compression algorithms.
- Lakehouse does not support these data types in Avro files: `Array`, `Map`, and nested records. Lakehouse marks columns with these data types as `NOT SECONDARY`, and does not load them.
- Lakehouse only supports an Avro union between one supported data type and `NULL`.
- The Avro `DECIMAL` data type is limited to the scale and precision supported by MySQL.
- Avro data with `FIXED` values or values in `BYTES` are only supported with the Avro `DECIMAL` data type.

11.2.2.3 Lakehouse Limitations for the CSV File Format

MySQL HeatWave Lakehouse has the following limitations for CSV files.

- MySQL does not recognise NaN values in `CSV` files. Replace all NaN values with NULL.
- Lakehouse does not support CSV files with more than 4MB per line.

- As of MySQL 9.3.2, Lakehouse supports the `VECTOR` data type for CSV and Parquet files. Consider the following limitations:
  - You cannot give the `VECTOR` data type a default value other than `NULL`.
  - Vectors that are in binary or hex format are not supported.
  - You cannot directly convert `VECTOR` to `JSON` or `JSON` to `VECTOR`.
    - To convert `VECTOR` to `JSON`, first convert `VECTOR` to a STRING data type by using the VECTOR_TO_STRING() function. Then, cast this string as `JSON`.

      For example, if `col_1` in `table_1` is a `VECTOR` column that you want to convert to `JSON`, you can do the following:

      ```
      SELECT CAST(VECTOR_TO_STRING(col_1) as JSON) FROM table_1;
      ```

    - To convert `JSON` to `VECTOR`, first convert `JSON` to a STRING data type, and then use the STRING_TO_VECTOR() function to convert to `VECTOR`.

      For example, if `col_1` in `table_1` is a `JSON` column that you want to convert to `VECTOR`, you can do the following:

      ```
      SELECT STRING_TO_VECTOR(CAST(col_1 as NCHAR)) FROM table_1;
      ```

  - If you run `Vector_to_string(col)`as a Vector_to_string() function, it fails if there are any `0x` (default for invalid vectors) values. To avoid errors, filter out these values.

11.2.2.4 Lakehouse Limitations for the JSON File Format

MySQL HeatWave Lakehouse has the following limitations for JSON files.

- As of MySQL 9.3.1, Lakehouse can load up to 4192192 bytes (~4MB) of data per line, and ignores any lines with more than this limit. Before MySQL 9.3.1, this limit is 65532 bytes.

Note

The JSON size is measured after Lakehouse extracts and converts JSON data to JSON DOM representation internally.

- As of MySQL 8.4.0, Lakehouse only supports Newline Delimited JSON.
- Tables created with `json` format must only have a single column that conforms to the `JSON` data type. In addition, stored generated columns or the `_metadatq_filename` column can be part of the table schema. See The JSON Data Type.
- As of MySQL 9.6.1, the JSON format is supported with stored generated columns. In earlier versions, it is not supported.

11.2.2.5 Lakehouse Limitations for the Parquet File Format

MySQL HeatWave Lakehouse has the following limitations for Parquet files.

- NaN values in Parquet files are loaded as NULL.
- As of MySQL 9.3.2, Lakehouse supports the `VECTOR` data type. Consider the following limitations:
  - To load Parquet files with the `VECTOR` data type (from Parquet strings or from Parquet (numeric) Lists), you must load the table manually with `is_strict_mode` set to `false`. For loading simple lists as `JSON`, you can load the data manually or automatically. To learn more, see Vector Data Type and External Table Syntax.

Tip

If you need to manually create a table but do not have the appropriate table definition, you can load the table automatically in `dryrun` mode and review the load script. You also need to convert columns from `JSON` to `VECTOR` as needed. See the example commands below to do this.

```
mysql> SET @input_list = '[{
  "db_name": "data_db",
  "tables": [{
    "table_name": "table_1",
    "engine_attribute": {"dialect": {"format": "parquet" "embed_model_id": "minilm"},
                         "file": [{"par": "https://objectstorage.us-ashburn-1.oraclecloud.com/p/.../n/t
}]}]';
mysql> CALL sys.HEATWAVE_LOAD(CAST(@input_list AS JSON), JSON_OBJECT('mode', 'dryrun'));
mysql> SELECT log->>"$.sql" AS "Load Script" FROM sys.heatwave_autopilot_report WHERE type = "sql" ORDE
```

- You cannot give the `VECTOR` data type a default value other than `NULL`.
- You cannot directly convert `VECTOR` to `JSON` or `JSON` to `VECTOR`.
  - To convert `VECTOR` to `JSON`, first convert `VECTOR` to a STRING data type by using the VECTOR_TO_STRING() function. Then, cast this string as `JSON`.

    For example, if `col_1` in `table_1` is a `VECTOR` column that you want to convert to `JSON`, you can do the following:

    ```
    SELECT CAST(VECTOR_TO_STRING(col_1) as JSON) FROM table_1;
    ```

  - To convert `JSON` to `VECTOR`, first convert `JSON` to a STRING data type, and then use the STRING_TO_VECTOR() function to convert to `VECTOR`.

    For example, if `col_1` in `table_1` is a `JSON` column that you want to convert to `VECTOR`, you can do the following:

    ```
    SELECT STRING_TO_VECTOR(CAST(col_1 as NCHAR)) FROM table_1;
    ```

  - If you run `Vector_to_string(col)` as a Vector_to_string() function, it fails if there are any `0x` (default for invalid vectors) values. To avoid errors, filter out these values.
- As of MySQL 9.1.2, you can load Parquet files with row groups that total up to 10GB with large MySQL HeatWave Cluster shapes.
- Lakehouse does not support the following data types in Parquet files. Lakehouse marks columns with these data types as `NOT SECONDARY`, and does not load them.
  - `BSON`
  - `ENUM`
  - `Interval`
  - `List`
    - Not supported before MySQL 9.3.2. As of MySQL 9.3.2, limited support is available for Parquet Lists in Lakehouse. Simple (non-nested) lists of the following Parquet data types are now

supported: `INT_8`, `INT_16`, `INT_32`, `INT_64`, `UINT_8`, `UINT_16`, `UINT_32`, `UINT_64`, `FLOAT` (32-bit), `DOUBLE`, `STRING`, and `BOOL`.

- `Map`
- `Unknown`
- `UUID`

- Do not use strict SQL mode if the inferred schema differs from the table schema.

11.2.2.6 Limitations for Delta Lake Tables

MySQL HeatWave Lakehouse has the following limitations for delta lake tables.

- Loading delta lake tables is only supported in MySQL HeatWave on OCI.
- Support for delta lake tables is restricted to delta tables with features enabled by default by a Spark client.
- Writing, updating, and creating of delta lake tables in MySQL HeatWave Lakehouse is not supported. Only reading of delta lake tables is supported.
- For partitioned tables, loading columns that are part of the partitioning key is not supported. These columns must be tagged as `NOT SECONDARY`. When loading tables with Auto Parallel Load, these columns are inferred as `NOT SECONDARY`.
- Loading of empty tables is not supported.
- Loading of tables located at the root of an Object Store bucket is not supported.
- Loading tables with deletion vectors or column-mapping properties enabled is not supported.
- The following delta lake data types are not supported:
  - BINARY
  - GEOGRAPHY
  - GEOMETRY
  - ARRAY (with the exception of arrays of simple types, which are inferred as JSON)
  - MAP
  - STRUCT
  - VARIANT
  - OBJECT
- As of MySQL 9.6.1, Delta Lake tables support stored generated columns. In earlier versions, it is not supported.

11.2.3 MySQL HeatWave AutoML Limitations

The following limitations apply to MySQL HeatWave AutoML. For MySQL HeatWave limitations, see: Section 11.2.1, “MySQL HeatWave Limitations”.

Text Handling Limitations

- MySQL HeatWave AutoML only supports datasets in the English language.

- MySQL HeatWave AutoML does not support `TEXT` columns with `NULL` values.
- MySQL HeatWave AutoML does not support target columns (a column with ground truth values) with a `TEXT` data type.
- MySQL HeatWave AutoML does not support recommendation tasks with columns that have a `TEXT` data type.
- For the forecasting task, `endogenous_variables` cannot be in `TEXT`.

Account Name Limitations

- The `ML_TRAIN` routine does not support MySQL user names that contain a period. For example, a user named `'joe.smith'@'%'` cannot run the `ML_TRAIN` routine. The model catalog schema created by the `ML_TRAIN` procedure incorporates the user name in the schema name (e.g., `ML_SCHEMA_joesmith`), and a period is not a permitted schema name character.

Memory Limitations

- The table used to train a model cannot exceed 10 GB, 100 million rows, or 1017 columns.
- As of MySQL 9.0.0, the shape you set for the MySQL HeatWave cluster in the DB system defines the total memory available to train a model and for all loaded models. For imported models, we recommend individual models have a size of 4GB or less. `ML_TRAIN` will fail to run and raises an error if it attempts to train a model larger than this. Before MySQL 9.0.0, the limit was 900 MB. To query all loaded models and relevant model sizes, see ML_MODEL_ACTIVE
- MySQL HeatWave on AWS only supports MySQL HeatWave AutoML with the MySQL HeatWave.256GB node shape. To use MySQL HeatWave machine learning functionality, select that shape when creating a MySQL HeatWave Cluster.

Routine and Query Limitations

- `ML_EXPLAIN_TABLE` and `ML_PREDICT_TABLE` are compute intensive processes, with `ML_EXPLAIN_TABLE` being the most compute intensive. We recommend the following:
  - `ML_EXPLAIN_TABLE`: Before MySQL 9.4.1, use the `batch_size` option to limit operations to batches of 10 to 100 rows by splitting large tables into smaller tables. For tables with over ten columns, we suggest a `batch_size` value of `10`. As of MySQL 9.4.1, the `batch_size` option is deprecated. Limit the input table to a maximum of 100 rows. If the input table has more than ten columns, limit it to ten rows.
  - `ML_PREDICT_TABLE`: Before MySQL 9.4.1, limit operations to batches of rows by splitting large tables into smaller tables by using the `batch_size` option. For MySQL 9.4.1 and later, if `ML_PREDICT_TABLE` takes a long time to complete, manually limit input tables to a maximum of 1,000 rows.
  - See Track Progress for MySQL HeatWave AutoML Routines to learn more.
- `ML_EXPLAIN`, `ML_EXPLAIN_ROW`, and `ML_EXPLAIN_TABLE` routines limit explanations to the 100 most relevant features.
- The `ML_PREDICT_TABLE` `ml_results` column contains the prediction results and the data. This combination must be less than 65,532 characters.
- Concurrent MySQL HeatWave analytics and MySQL HeatWave AutoML queries are not supported. A MySQL HeatWave AutoML query must wait for MySQL HeatWave analytics queries to finish, and vice versa. MySQL HeatWave analytics queries are given priority over MySQL HeatWave AutoML queries.
- The `ML_PREDICT_ROW`, `ML_MODEL_IMPORT`, and `ML_MODEL_EXPORT` routines are not supported with the `TwoTower` recommendation model.

Other Limitations

- If you delete a recommendation model trained with the `TwoTower` model from the model catalog, you need to run a Delete Model API to manage the generated embedding tables.