CARNEGIE MELLON UNIVERSITY
COMPUTER SCIENCE DEPARTMENT
15-445/645 – DATABASE SYSTEMS (FALL 2023)
PROF. ANDY PAVLO AND JIGNESH PATEL

Homework #5 (by Fernando Lisboa) – Solutions

Due: **Sunday Dec 3, 2023 @ 11:59pm**

IMPORTANT:
- **Upload this PDF** with your answers to **Gradescope by 11:59pm on Sunday Dec 3, 2023**.
- **Plagiarism**: Homework may be discussed with other students, but all homework is to be completed **individually**.
- **You have to use this PDF for all of your answers.**

For your information:
- Graded out of **100** points; **4** questions total

Revision : 2023/12/03 22:57

Question	Points	Score
Write-Ahead Logging	25	
Two-Phase Commit	18	
Distributed Query Plan	27	
Miscellaneous	30	
Total:	100	

Question 1: Write-Ahead Logging............................[25 points]

Consider a DBMS using write-ahead logging with physical log records with the STEAL and NO-FORCE buffer pool management policy. Assume the DBMS executes a non-fuzzy checkpoint where all dirty pages are written to disk.

Its transaction recovery log contains log records of the following form:

```
<txnId, objectId, beforeValue, afterValue>
```

The log also contains checkpoint, transaction begin, and transaction commit records.

The database contains three objects (i.e., A, B, and C).

The DBMS sees records as in Figure 1 in the WAL on disk after a crash.

Assume the DBMS uses ARIES as described in class to recover from failures.

LSN	WAL Record
1	`<T1 BEGIN>`
2	`<T1, A, 12, 55>`
3	`<T1, B, 42, 8>`
4	`<T2 BEGIN>`
5	`<T2, A, 55, 62>`
6	`<T1 COMMIT>`
7	`<T2, C, 0, 35>`
8	`<T3 BEGIN>`
9	`<T3, B, 8, 81>`
10	`<T2, A, 62, 67>`
11	<**CHECKPOINT**>
12	`<T3, B, 81, 45>`
13	`<T2, A, 67, 13>`
14	`<T3 COMMIT>`
15	`<T2, C, 35, 66>`

Figure 1: WAL

(a) **[6 points]** What are the values of A, B, and C in the database stored on disk before the DBMS recovers the state of the database?

- ☐ A=67, B=81, C=35
- ☐ A=13, B=45, C=66
- ☐ A=13, B=45, C=Not possible to determine
- ☐ A=55, B=8, C=35
- ☐ A=Not possible to determine, B:42, C=0
- ☐ A=12, B=42, C=Not possible to determine
- ☐ A=Not possible to determine, B=Not possible to determine, C:66
- ☐ A=Not possible to determine, B=81, C=Not possible to determine
- ☐ A=13, B=Not possible to determine C=35

Question 1 continues...

■ **A,B,C=Not possible to determine**

Solution: The checkpoint flushed everything to disk, but then the data objects A,B and C were all modified by transactions after the checkpoint.

Since we are using STEAL, any dirty page could be written to disk, so therefore we don't know the contents of the database on disk at the crash.

(b) **[3 points]** What should be the correct action on T1 when recovering the database from WAL?

- ☐ redo all of T1's changes
- ☐ undo all of T1's changes
- ■ **do nothing to T1**

Solution: T1 committed before the checkpoint. All of its changes were written to disk. There is nothing to redo or undo.

(c) **[3 points]** What should be the correct action on T2 when recovering the database from WAL?

- ☐ redo all of T2's changes
- ■ **undo all of T2's changes**
- ☐ do nothing to T2

Solution: T2 never committed. All of its changes should only be undone.

(d) **[3 points]** What should be the correct action on T3 when recovering the database from WAL?

- ■ **redo all of T3's changes**
- ☐ undo all of T3's changes
- ☐ do nothing to T3

Solution: T3 committed after the checkpoint, so that means the DBMS has to redo all of its changes.

(e) **[10 points]** Assume that the DBMS flushes all dirty pages when the recovery process finishes. What are the values of A, B, and C after the DBMS recovers the state of the database from the WAL in Figure 1?

- ☐ A=33, B=6, C=42
- ☐ A=13, B=66, C=55
- ☐ A=13, B=45, C=66
- ☐ A=13, B=7, C=67
- ☐ A=12, B=42, C=0

Question 1 continues…

- ☐ A=33, B=42, C=100
- ■ **A=55, B=45, C=0**
- ☐ A=13, B=42, C=67
- ☐ A=33, B=20, C=43
- ☐ A=55, B=45, C=66
- ☐ A=55, B=12, C=66
- ☐ Not possible to determine

Solution: A = 55 (Undo changes by T2; rollback to value committed by T1)
B = 45 (Redo changes by T3; rollback to afterValue of last T3 update)
C = 0 (Undo changes by T2; rollback to beforeValue of earliest T2 update)

Question 2: Two-Phase Commit .... [18 points]

Consider a distributed transaction T operating under the two-phase commit protocol with the ***early acknowledgement optimization***. Let N_0 be the *coordinator* node, and N_1, N_2, N_3 be the *participant* nodes. Let the C be the client application issuing the transaction request. Assume that the client communicates directly with the coordinator node.

The following messages have been sent:

time	message
1	C to N_0: "**`REQUEST:COMMIT`**"
2	N_0 to N_2: "**`Phase1:PREPARE`**"
3	N_2 to N_0: "**`OK`**"
4	N_0 to N_1: "**`Phase1:PREPARE`**"
5	N_0 to N_3: "**`Phase1:PREPARE`**"
6	N_1 to N_0: "**`OK`**"
7	N_3 to N_0: "**`OK`**"

Figure 2: Two-Phase Commit messages for transaction T for parts (a), (b)

(a) **[6 points]** Who should send a message next at time 8 in Figure 2? Select *all* the possible answers.

- ☐ C
- ■ N_0
- ☐ N_1
- ☐ N_2
- ☐ N_3
- ☐ It is not possible to determine

Solution: At the time when all participant nodes have responded with "**`OK`**", T is considered to be committed. N_0 must notify the participant nodes N_1, N_2, and N_3 of the decision to commit. At this time N_0 should also notify the client application since the protocol is running with the early acknowledgement optimization.

(b) **[6 points]** To whom? Again, select *all* the possible answers.

- ■ C
- ☐ N_0
- ■ N_1
- ■ N_2
- ■ N_3
- ☐ It is not possible to determine

Solution: At the time when all participant nodes have responded with "**`OK`**", T is considered to be committed. N_0 must notify the participant nodes N_1, N_2, and N_3 of the decision to commit. At this time N_0 should also notify the client application since the protocol is running with the early acknowledgement optimization.

Question 2 continues…

(c) **[6 points]** Now suppose that the two-phase commit protocol is incorrectly implemented in our system, and we instead have the message log shown in Figure 3 (below). C, N_0, N_1, N_2, and N_3 all play the same roles as they did in the previous parts.

At what timestep(s) is the two-phase commit protocol violated? Select *all* that apply.

☐ 2
☐ 3
☐ 4
☐ 5
☐ 6
☐ 7
[✓ **8**]
[✓ **9**]

Solution: After receiving the "**ABORT**" from N_2 at time 5, the coordinator (N_0) will mark the transaction as aborted, and it will not proceed to the commit phase with nodes N_1 and N_3 at times 8 and 9. It should instead notify all of the participants of the decision to abort the transaction.

time	message
1	C to N_0: "**REQUEST:COMMIT**"
2	N_0 to N_2: "**Phase1:PREPARE**"
3	N_0 to N_1: "**Phase1:PREPARE**"
4	N_0 to N_3: "**Phase1:PREPARE**"
5	N_2 to N_0: "**ABORT**"
6	N_1 to N_0: "**OK**"
7	N_3 to N_0: "**OK**"
8	N_0 to N_1: "**Phase2:COMMIT**"
9	N_0 to N_3: "**Phase2:COMMIT**"

Figure 3: Incorrect Two-Phase Commit messages for transaction T for part (c)

Homework #5 continues…

Question 3: Distributed Query Plan . . . . [27 points]

The CMUDB group is working on a brand new shared-nothing distributed database system called BusTub***. They are developing the distributed query engine.

Given the following schema:

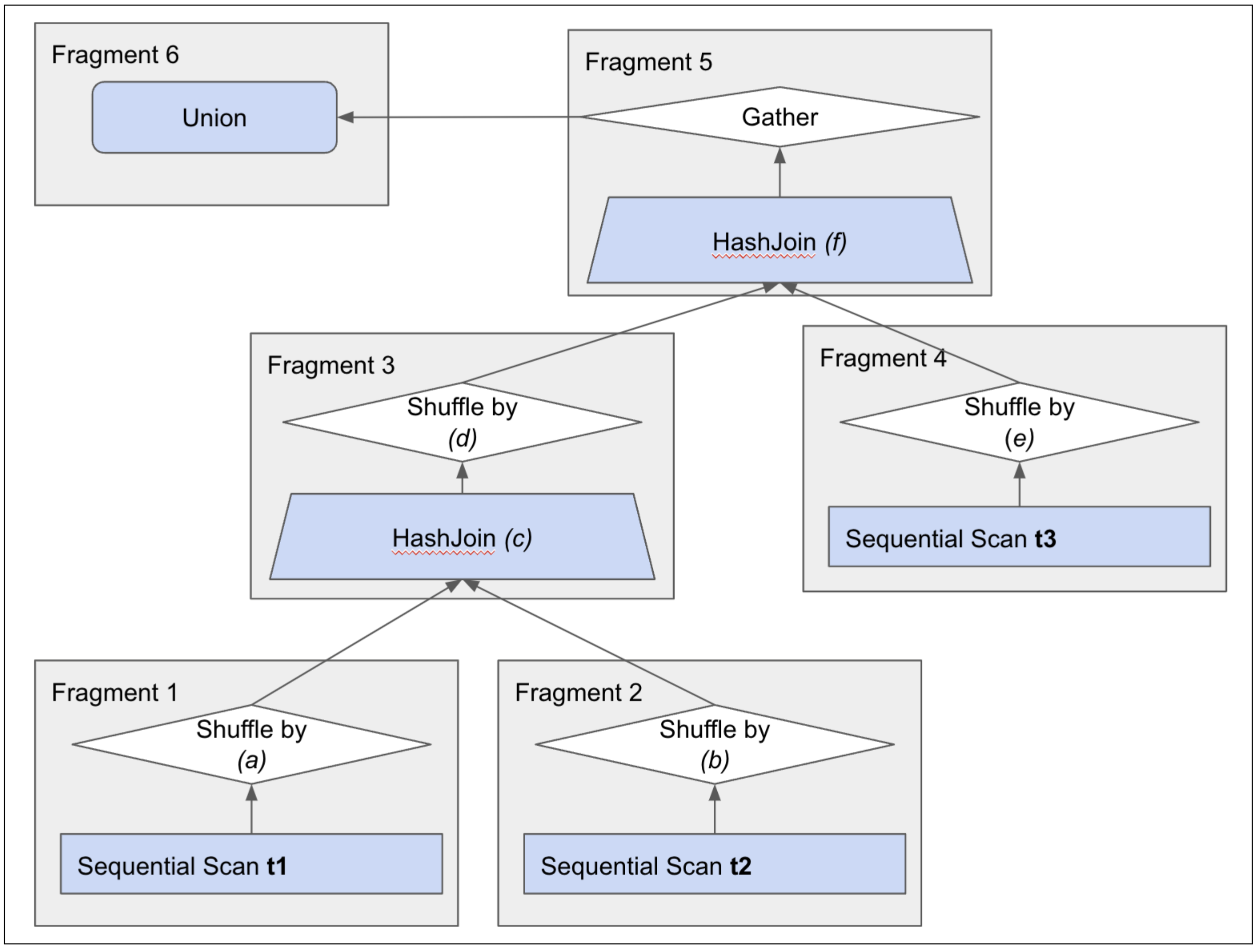
```
CREATE TABLE t1(PARTITION KEY v1 int, v2 int);
CREATE TABLE t2(PARTITION KEY v3 int, v4 int);
CREATE TABLE t3(PARTITION KEY v5 int, v6 int);
```

The database system partitions the tables by key range. That is to say, each node in the system manages rows of the table within a non-overlapping range of keys.

Given the following query:

```
SELECT * FROM (t1 INNER JOIN t2 ON v1 = v3)
    INNER JOIN t3 ON v4 = v5;
```

Assume that the query optimizer doesn't know the ranges of data stored on each node and only knows the tables are sharded by some keys, and it generates the following plan:



Question 3 continues. . .

Note that this query plan is missing some expressions in the Shuffle and HashJoin operators, corresponding to the following question parts. Fill in the missing parts to this query plan.

(a) **[3 points]** In the Shuffle from fragment 1 to fragment 3, which column should be used to shuffle the outputted rows of the **t1** scan?

■ **v1**
☐ v2
☐ v3
☐ v4
☐ v5
☐ v6

Solution: The rows should be shuffled by their join key. For **t1** this is **v1**.

(b) **[3 points]** In the Shuffle from fragment 2 to fragment 3, which column should be used to shuffle the outputted rows of the **t2** scan?

☐ v1
☐ v2
■ **v3**
☐ v4
☐ v5
☐ v6

Solution: The rows should be shuffled by their join key in the next fragment. For **t2** this is **v3**.

(c) **[2 points]** How are tables **t1** and **t2** being joined in the fragment 3 HashJoin?

☐ $v1 = v2$
☐ $v2 = v3$
■ $v1 = v3$
☐ $v2 = v4$
☐ $v3 = v5$
☐ $v4 = v5$

Solution: Tables **t1** and **t2** are being joined on $v1 = v3$ in the query.

(d) **[3 points]** In the Shuffle from fragment 3 to fragment 5, which column should be used to shuffle the outputted rows of the HashJoin?

☐ v1
☐ v2
☐ v3
■ **v4**
☐ v5
☐ v6

Solution: The rows should be shuffled by their join key in the next fragment. For the output of the inner join of tables **t1** and **t2**, this is **v4**.

Question 3 continues…

(e) **[3 points]** In the Shuffle from fragment 4 to fragment 5, which column should be used to shuffle the outputted rows of the **t3** scan?

☐ v1
☐ v2
☐ v3
☐ v4
■ **v5**
☐ v6

Solution: The rows should be shuffled by their join key in the next fragment. For **t3**, this is **v5**.

(f) **[3 points]** How are the input tables being joined in the fragment 5 HashJoin?

☐ $v1 = v2$
☐ $v2 = v3$
☐ $v1 = v3$
☐ $v2 = v4$
☐ $v3 = v5$
■ $v4 = v5$

Solution: Tables (**t1** INNER JOIN **t2**) and **t3** are being joined on $v4 = v5$ in the query.

(g) **[5 points]** Assume there are 3 nodes in the system and the data ranges in each node are as follows:

	$t1.v1$	$t2.v3$	$t3.v5$
$Node\ 1$	0 - 999	0 - 999	0 - 999
$Node\ 2$	1000 - 1999	1000 - 1999	1000 - 1999
$Node\ 3$	2000 - 2999	2000 - 2999	2000 - 2999

Table 1: Data distribution for table $t1$ to $t3$

Assume that $t2.v4$ also has all values across 0-2999, except that they are uniformly randomly distributed across the three nodes (i.e., the table is not partitioned by v4 and so any value of v4 could be in any node).

Given the data ranges on each node and our query plan from earlier **assume that our system has generated the best possible execution schedule** (i.e., the optimal scheduling of scan/join partitions on nodes within fragments which minimizes the worst case amount of data transferred over the network).

What is the worst case number of rows that need to be shuffled from fragments 1 and 2 to fragment 3 across the network in this schedule?

■ **0**
☐ 1000

Question 3 continues…

☐ 2000
☐ 3000

Solution: Since we know that all equal values of $t1.v1$ and $t2.v3$ are on the same node, they can be locally joined at each node without needing to shuffle any data from the scan fragment to the HashJoin fragment over the network. That is, we can scan and join 0-999 on Node 1, 1000-1999 on Node 2, and 2000-2999 on Node 3 without needing to transfer any data between nodes.

(h) **[5 points]** With same assumptions as the previous part, what is the worst case number of rows that need to be shuffled from fragments 3 and 4 to fragment 5 across the network in this schedule?

☐ 0
☐ 1000
☐ 2000
[✓ **3000**]

Solution: Fragment 5 joins on $t2.v4$ and $t3.v5$. Since $t2.v4$ is not a partition key, each value can be in any one of the 3 compute nodes. In the worst case, none of the equal $t2.v4$ and $t3.v5$ values are in the same node and so they must all be shuffled to the same node. There are 3000 possible values.

Question 4: Miscellaneous . . . . [30 points]

(a) **[3 points]** Both PAXOS and Two-Phase Commit protocols can be used to implement distributed transactions.

■ **True**
☐ False

(b) **[3 points]** A DBMS implementing transactions via shadow paging never has to undo or redo any actions of a transaction.

■ **True**
■ **False**

Solution: The wording to this question was ambiguous as to whether or not cleaning up the shadow pages on recovery was considered undoing the actions of a transaction, and so both answers were accepted.

(c) **[3 points]** In reference to recovery algorithms that use a write-ahead log (WAL). Under NO-STEAL + FORCE policy, a DBMS will need to redo the changes of a committed transaction during recovery.

☐ True
■ **False**

(d) **[3 points]** In reference to recovery algorithms that use a write-ahead log (WAL). Under NO-STEAL + FORCE policy, a DBMS will have to undo the changes of an aborted transaction during recovery.

☐ True
■ **False**

(e) **[3 points]** Fuzzy checkpoints need to block the execution of all transactions while a consistent snapshot is written to disk.

☐ True
■ **False**

(f) **[3 points]** For a DBMS that uses ARIES, all updated pages must be flushed to disk for a transaction to commit.

☐ True
■ **False**

(g) **[3 points]** During the undo phase of ARIES, all transactions that committed after the last checkpoint are undone.

Question 4 continues…

☐ True
[✓ **False**]

(h) **[3 points]** In ARIES, only transactions that commit will have an associated "TXN-END" record in the log.

☐ True
[✓ **False**]

(i) **[3 points]** With consistent hashing, if a node fails then all keys must be reshuffled among the remaining nodes.

☐ True
[✓ **False**]

(j) **[3 points]** In a system with strong consistency requirements, it is best for the DBMS to implement active-passive replication with asynchronous replication and continuous log streaming.

☐ True
[✓ **False**]