QSqlTableModel en action

Par Traducteur : Louis du Verdier - Qt Developer Network

Date de publication : 24 avril 2011

Dernière mise à jour : 30 mai 2011

Pour un programmeur qui démarrait sa vie avec PHP et les technologies Web, QSqlTableModel est une magnifique classe pour l'interaction avec les tables d'une base de données. Elle est très facile d'usage pour insérer, supprimer ou actualiser une entrée.

Commentez

I - L'article original

Le Qt Developer Network est un réseau de développeurs utilisant Qt afin de partager leur savoir sur ce framework. **Vous pouvez le consulter en anglais.**

Cet article est la traduction de **QSqlTableModel in action**.

II - Une histoire de pointeur

Par habitude dans tous mes projets, je crée un pointeur de fonction (ou peu importe le nom que ça a) :

```
QSqlTableModel* lessons::lessonsTable()
{
  QSqlTableModel *model = new QSqlTableModel;
  model->setTable("lessons");
  model->setEditStrategy(QSqlTableModel::OnFieldChange);
  model->select();
  model->setHeaderData(0, Qt::Horizontal, "ID");
  model->setHeaderData(1, Qt::Horizontal, "Name");
  model->setHeaderData(2, Qt::Horizontal, "Teacher");
  return model;
}
```

Ce code est tiré d'un de mes projets qui a gagné la médaille d'argent dans un concours international d'informatique. D'ailleurs, regardons le code. Dans la deuxième ligne :

```
model->setTable("lessons");
```

Nous avons une fonction appelée setTable. La documentation officielle dit ceci à son propos :

Définit à tableName la table de la base de données sur laquelle le modèle opère. Ne sélectionne pas de données depuis la table, mais rapporte son information de champ.

J'adore la documentation de Qt, vraiment. Après cette fonction, à la ligne quatre, nous utilisons la fonction select, pour sélectionner les données dans la table.

```
model->setEditStrategy(QSqlTableModel::OnFieldChange);
```

La fonction dans la ligne ci-dessus définit la stratégie d'édition des valeurs de la base de données à l'une des stratégies variées d'édition de Qt. Mais attention, cela va annuler tout changement en attente.

Avec Qt, nous avons trois stratégies d'édition :

1. QSqlTableModel::OnFieldChange ;
2. QSqlTableModel::OnRowChange ;
3. QSqlTableModel::OnManualSubmit.

Et elles sont très bien expliquées dans la documentation de Qt. Mais je compte donner des exemples plus tard.

Dans la documentation de référence de la classe QSqlTableModel de Qt 4.6, il y a une ligne de la sorte :

```
model->removeColumn(0);
```

Mais je préfère cacher la colonne plutôt que de la retirer parce que, si vous la retirez, lorsque vous allez tenter d'éditer les entrées dans la table par une vue, ça va dire : « QSqlQuery::value: not positioned on a valid record« . Enfin, continuons.

```
model->setHeaderData(0, Qt::Horizontal, "ID");
model->setHeaderData(1, Qt::Horizontal, "Name");
model->setHeaderData(2, Qt::Horizontal, "Teacher");
```

Ces trois lignes vous aident à faire des interfaces plus explicatives par la définition d'un sous-titre. C'est pratique quand vous voulez utiliser une vue, par exemple QTableView.

Maintenant, mettons la table à l'intérieur d'une vue. J'ai choisi QTableView :

```
this->tableModel = lsn.lessonsTable();
ui->tableView_lessons->setModel(this->tableModel);
ui->tableView_lessons->setColumnHidden(0, true);
```

lsn est une classe que j'ai créée qui a la fonction que j'ai écrite plus haut. Comme mentionné précédemment, j'ai préféré cacher la colonne que je ne voulais pas montrer, je l'ai fait dans la troisième ligne.

III - Remerciements

Merci à **Thibaut Cuvelier** et à **Claude Leloup** pour leur relecture !