第20篇 (数据库) SQL表格模型 `QSqlTableModel`

代码地址：https://github.com/Lornatang/QtStartQuicklyTutorial/tree/main/SQL03

一、创建数据库

1．新建Qt Gui应用，项目名称为`tableModel`，基类`QMainWindow`，类名`MainWindow`。

2．完成后打开`tableModel.pro`文件，将第一行代码更改为：

```
QT       += coregui sql
```

然后保存文件。

3．向项目中添加新的C++头文件，名称为`connection.h`。完成后将其内容更改如下：

```
#ifndef CONNECTION_H
#define CONNECTION_H
#include <QSqlDatabase>#include <QSqlQuery>static bool createConnection()
{
    QSqlDatabase db = QSqlDatabase::addDatabase("QSQLITE");
    db.setDatabaseName("database.db");
    if(!db.open()) return false;
    QSqlQuery query;
    query.exec(QString(
      "create table student (id int primary key, name vchar)"));
    query.exec(QString("insert into student values (0,'刘明')"));
    query.exec(QString("insert into student values (1,'陈刚')"));
    query.exec(QString("insert into student values (2,'王红')"));
    return true;
}
#endif // CONNECTION_H
```

这里因为语句中使用了中文，所以使用了`QString()`进行编码转换，这个还需要在`main()`函数中设置编码。

4．下面将`main.cpp`文件更改如下：

```
#include "mainwindow.h"#include <QApplication>#include "connection.h"#include <QTextCodec>
int main(int argc, char *argv[])
{
    QApplication a(argc, argv);
    QTextCodec::setCodecForTr(QTextCodec::codecForName("utf8"));
    QTextCodec::setCodecForCStrings(QTextCodec::codecForLocale());
    if(!createConnection())
       return 1;
    MainWindow w;
    w.show();

    return a.exec();
}
```

这里的`setCodecForCStrings()`就是用来设置字符串编码的。

5．下面进入设计模式，向窗口上拖入`Label`、`Push Button`、`Line Edit`和`Table View`等部件，进行界面设计，效果如下图所示。

6．完成后到`mainwindow.h`文件中，先包含头文件：

```
#include <QSqlTableModel>
```

然后添加私有对象声明：

```
QSqlTableModel *model;
```

7．到`mainwindow.cpp`，在构造函数添加如下代码：

```
model = new QSqlTableModel(this);
model->setTable("student");
model->setEditStrategy(QSqlTableModel::OnManualSubmit);
model->select(); //选取整个表的所有行
//不显示name属性列,如果这时添加记录，则该属性的值添加不上
// model->removeColumn(1);
ui->tableView->setModel(model);
//使其不可编辑
//ui->tableView->setEditTriggers(QAbstractItemView::NoEditTriggers);
```

这里创建一个`QSqlTableModel`后，只需使用`setTable()`来为其指定数据库表，然后使用`select()`函数进行查询，调用这两个函数就等价于执行了`“select * from student”`这个SQL

语句。这里还可以使用`setFilter()`来指定查询时的条件，在后面会看到这个函数的使用。在使用该模型以前，一般还要设置其编辑策略，它由`QSqlTableModel::EditStrategy`枚举变量定义，一共有三个值，如下图所示。用来说明当数据库中的值被编辑后，什么情况下提交修改。

常量	描述
QSqlTableModel::OnFieldChange	所有对模型的改变都会立即应用到数据库
QSqlTableModel::OnRowChange	对一条记录的改变会在用户选择另一条记录时被应用
QSqlTableModel::OnManualSubmit	所有的改变都会在模型中进行缓存，直到调用 submitAll()或者 revertAll()函数

运行程序，效果如下图所示。

可以看到，这个模型已经完全脱离了SQL语句，我们只需要执行`select()`函数就能查询整张表。上面有两行代码被注释掉了，你可以取消注释，测试一下它们的作用。

二、修改操作

1．我们进入“提交修改”按钮的单击信号槽，更改如下：

```
void MainWindow::on_pushButton_3_clicked()
{
    model->database().transaction(); //开始事务操作
    if (model->submitAll()) {
       model->database().commit(); //提交
    } else {
       model->database().rollback(); //回滚
       QMessageBox::warning(this, tr("tableModel"),
                            tr("数据库错误: %1")
                            .arg(model->lastError().text()));
    }
}
```

这里用到了事务操作，真正起提交操作的是`model->submitAll()`一句，它提交所有更改。

2．进入“撤销修改”按钮的单击信号槽，更改如下：

```
void MainWindow::on_pushButton_4_clicked()
{
    model->revertAll();
}
```

3．在`mainwindow.cpp`文件中包含头文件：

```
#include <QMessageBox>#include <QSqlError>
```

4．现在运行程序，我们将“陈刚”改为“李强”，如果我们点击“撤销修改”，那么它就会重新改为“陈刚”，而当我们点击“提交修改”后它就会保存到数据库，此时再点击“撤销修改”就修改不回来了。

可以看到，这个模型可以将所有修改先保存到`model`中，只有当我们执行提交修改后，才会真正写入数据库。当然这也是因为我们在最开始设置了它的保存策略：

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

这里的`OnManualSubmit`表明我们要提交修改才能使其生效。

三、查询操作

1．进入“查询”按钮的单击信号槽，更改如下：

```
void MainWindow::on_pushButton_clicked()
{
    QString name = ui->lineEdit->text();
    //根据姓名进行筛选
    model->setFilter(QString("name = '%1'").arg(name));
    //显示结果
    model->select();
}
```

使用`setFilter()`函数进行关键字筛选，这个函数是对整个结果集进行查询。

2．进入“显示全表”按钮的单击信号槽，更改如下：

```
void MainWindow::on_pushButton_2_clicked()
{
    model->setTable("student");    //重新关联表
    model->select();    //这样才能再次显示整个表的内容
}
```

为了再次显示整个表的内容，我们需要再次关联这个表。

3．下面运行程序，输入一个姓名，点击“查询”按钮后，就可以显示该记录了。再点击“显示全表”按钮则返回。如下图所示。

四、排序操作

分别进入“按id升序排序”和“按id降序排序”按钮的单击信号槽，更改如下：

```
// 升序
void MainWindow::on_pushButton_7_clicked()
{
    model->setSort(0, Qt::AscendingOrder); //id属性即第0列，升序排列
    model->select();
}
// 降序
void MainWindow::on_pushButton_8_clicked()
{
    model->setSort(0, Qt::DescendingOrder);
    model->select();
}
```

这里使用了`setSort()`函数进行排序，它有两个参数，第一个参数表示按第几个属性排序，表头从左向右，最左边是第0个属性，这里就是id属性。第二个参数是排序方法，有升序和降序两种。运行程序，效果如下图所示。

五、删除操作

我们进入“删除选中行”按钮的单击信号槽，更改如下：

```
void MainWindow::on_pushButton_6_clicked()
{
    //获取选中的行
    int curRow = ui->tableView->currentIndex().row();

    //删除该行
    model->removeRow(curRow);

    int ok = QMessageBox::warning(this,tr("删除当前行!"),tr("你确定"
                                                    "删除当前行吗？"),
                                  QMessageBox::Yes,QMessageBox::No);
    if(ok == QMessageBox::No)
    {
       model->revertAll(); //如果不删除，则撤销
    }
    else model->submitAll(); //否则提交，在数据库中删除该行
}
```

删除行的操作会先保存在`model`中，当我们执行了`submitAll()`函数后才会真正的在数据库中删除该行。这里我们使用了一个警告框来让用户选择是否真得要删除该行。运行程序，效果如下图所示。

我们点击第二行，然后单击“删除选中行”按钮，出现了警告框。这时你会发现，表中的第二行前面出现了一个小感叹号，表明该行已经被修改了，但是还没有真正的在数据库中修

改，这时的数据有个学名叫脏数据(Dirty Data)。当我们按钮“Yes”按钮后数据库中的数据就会被删除，如果按下“No”，那么更改就会取消。

六、插入操作

我们进入“添加记录”按钮的单击信号槽，更改如下：

```
void MainWindow::on_pushButton_5_clicked()
{
    int rowNum = model->rowCount(); //获得表的行数
    int id = 10;
    model->insertRow(rowNum); //添加一行
    model->setData(model->index(rowNum,0),id);
    //model->submitAll(); //可以直接提交
}
```

在表的最后添加一行，因为在`student`表中我们设置了`id`号是主键，所以这里必须使用`setData()`函数给新加的行添加`id`属性的值，不然添加行就不会成功。这里可以直接调用`submitAll()`函数进行提交，也可以利用“提交修改”按钮进行提交。运行程序，效果如下图所示。

按下“添加记录”按钮后，就添加了一行，不过在该行的前面有个星号，如果我们按下“提交修改”按钮，这个星号就会消失。当然，如果我们将上面代码里的提交函数的注释去掉，也就不会有这个星号了。