# 第19篇 (数据库) 利用`QSqlQuery`类执行SQL语句

代码地址：https://github.com/Lornatang/QtStartQuicklyTutorial/tree/main/SQL02


一、创建数据库连接
二、操作结果集
三、在SQL语句中使用变量
四、批处理操作
五、事务操作


## 一、创建数据库连接

前面我们是在主函数中创建数据库连接，然后打开并使用。实际中为了明了方便，一般将数据库连接单独放在一个头文件中。下面来看一个例子。

1．新建Qt Gui应用，项目名称为`myquery`，基类为`QMainWindow`，类名为`MainWindow`。完成后打开`myquery.pro`并将第一行代码更改为：

```
QT       += coregui sql
```

然后保存该文件。

2．向项目中添加新的C++头文件，名称为`connection.`h，然后打开该文件，更改如下：

```
#ifndef CONNECTION_H
#define CONNECTION_H
#include <QMessageBox>#include <QSqlDatabase>#include <QSqlQuery>static bool createConnect
ion()
{
    QSqlDatabase db = QSqlDatabase::addDatabase("QSQLITE");
    db.setDatabaseName(":memory:");
    if (!db.open()) {
       QMessageBox::critical(0, qApp->tr("Cannot open database"),
           qApp->tr("Unable to establisha database connection."
                    ), QMessageBox::Cancel);
       return false;
    }
    QSqlQuery query;
    query.exec("create table student (id int primary key, "
               "name varchar(20))");
    query.exec("insert into student values(0, 'first')");
    query.exec("insert into student values(1, 'second')");
    query.exec("insert into student values(2, 'third')");
    query.exec("insert into student values(3, 'fourth')");
    query.exec("insert into student values(4, 'fifth')");
    return true;
}
#endif // CONNECTION_H
```

在这个头文件中我们添加了一个建立连接的函数，使用这个头文件的目的就是要简化主函数中的内容。这里先创建了一个SQLite数据库的默认连接，设置数据库名称时使用了`“:memory:”`，表明这个是建立在内存中的数据库，也就是说该数据库只在程序运行期间有效，等程序运行结束时就会将其销毁。当然，大家也可以将其改为一个具体的数据库名称，比如`“my.db”`，这样就会在项目目录中创建该数据库文件了。下面使用`open()`函数将数据库打开，如果打开失败，则弹出提示对话框。最后使用`QSqlQuery`创建了一个`student`表，并插入了包含`id`和`name`两个属性的五条记录，如下图所示。其中，`id`属性是`int`类型的，`“primary key”`表明该属性是主键，它不能为空，而且不能有重复的值；而`name`属性是`varchar`类型的，并且不大于20个字符。这里使用的SQL语句都要包含在双引号中，如果一行写不完，那么分行后，每一行都要使用两个双引号引起来。

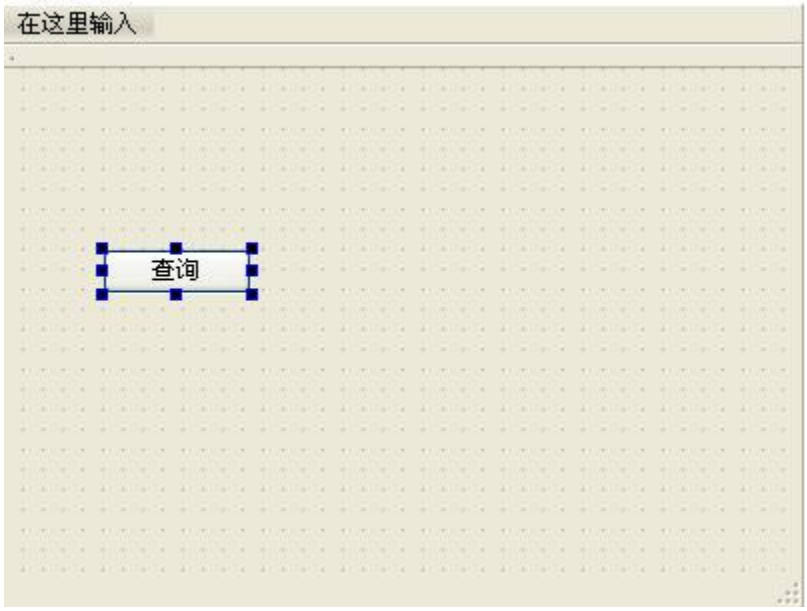
| id | name |
| --- | --- |
| 0 | first |
| 1 | second |
| 2 | third |
| 3 | fourth |
| 4 | fifth |

需要注意，代码中的`query`没有进行任何指定就可以操作前面打开的数据库，这是因为现在只有一个数据库连接，它就是默认连接，这时候所有的操作都是针对该连接的。但是如果要同时操作多个数据库连接，就需要进行指定了，这方面内容可以参考《Qt Creator快速入门》的第17章。

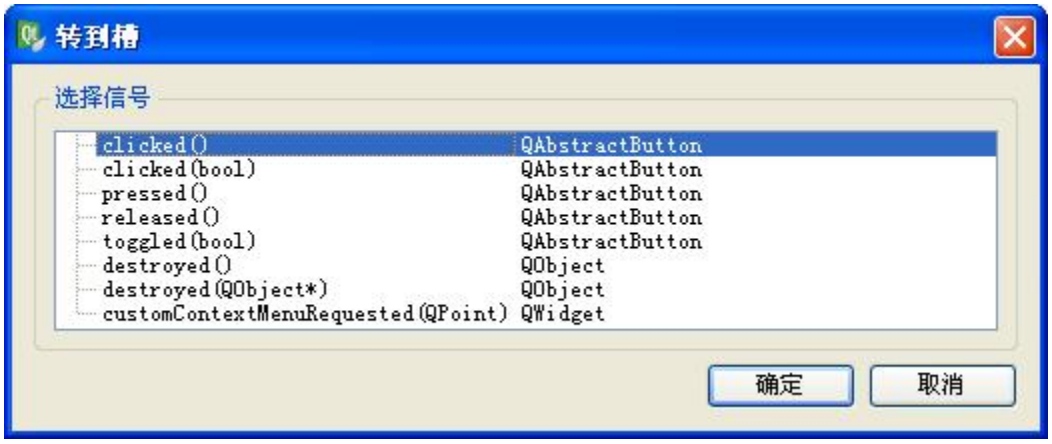
3．下面我们到`main.cpp`中调用连接函数。

```
#include "mainwindow.h"#include <QApplication>#include "connection.h"int main(int argc, ch
ar *argv[])
{
    QApplication a(argc, argv);

    if (!createConnection())
           return 1;

    MainWindow w;
    w.show();

    return a.exec();
}
```

4．我们往界面上添加一个按钮来实现查询操作。双击`mainwindow.ui`文件进入设计模式。然后将一个`Push Button`拖到界面上，并修改其显示文本为“查询”。效果如下图所示。




5．在查询按钮上点击鼠标右键，选择“转到槽”，然后选择`clicked()`单击信号槽并点击确定，如下图所示。




6．将槽的内容更改如下：

```
void MainWindow::on_pushButton_clicked()
{
    QSqlQuery query;
    query.exec("select * from student");
    while(query.next())
    {
       qDebug() << query.value(0).toInt()
                                    << query.value(1).toString();
    }
}
```

7．在`mainwindow.cpp`文件中添加头文件：

```
#include <QSqlQuery>#include <QDebug>
```

8．运行程序，然后按下查询按钮，在应用程序输出窗口将会输出结果，效果如下图所示。


# 二、操作结果集

在前面的程序中，我们使用`query.exec("select * from student");`查询出表中所有的内容。其中的SQL语句`“select * from student”`中`“*”`号表明查询表中记录的所有属性。而当`query.exec("select * from student");`这条语句执行完后，我们便获得了相应的执行结果，因为获得的结果可能不止一条记录，所以称之为结果集。

结果集其实就是查询到的所有记录的集合，在`QSqlQuery`类中提供了多个函数来操作这个集合，需要注意这个集合中的记录是从0开始编号的。最常用的操作有：

- `seek(int n)` ：`query`指向结果集的第n条记录；
- `first()` ：`query`指向结果集的第一条记录；
- `last()` ：`query`指向结果集的最后一条记录；
- `next()` ：`query`指向下一条记录，每执行一次该函数，便指向相邻的下一条记录；
- `previous()` ：`query`指向上一条记录，每执行一次该函数，便指向相邻的上一条记录；
- `record()` ：获得现在指向的记录；
- `value(int n)` ：获得属性的值。其中`n`表示你查询的第n个属性，比方上面我们使用`“select * from student`”就相当于`“select id, name from student”`，那么`value(0)`返回`id`属性的值，`value(1)`返回`name`属性的值。该函数返回`QVariant`类型的数据，关于该类型与其他类型的对应关系，可以在帮助中查看QVariant。
- `at()` ：获得现在`query`指向的记录在结果集中的编号。

需要特别注意，刚执行完`query.exec("select *from student");`这行代码时，`query`是指向结果集以外的，我们可以利用`query.next()`使得`query`指向结果集的第一条记录。当然我们也可以利用`seek(0)`函数或者`first()`函数使`query`指向结果集的第一条记录。但是为了节省内存开销，推荐的方法是，在`query.exec("select * from student");`这行代码前加上`query.setForwardOnly(true);`这条代码，此后只能使用`next()`和`seek()`函数。

下面我们通过例子来演示一下这些函数的使用。将槽更改如下：

```
void MainWindow::on_pushButton_clicked()
{
    QSqlQuery query;
    query.exec("select * from student");
    qDebug() << "exec next() :";
    //开始就先执行一次next()函数，那么query指向结果集的第一条记录
    if(query.next())
    {
       //获取query所指向的记录在结果集中的编号
       int rowNum = query.at();
       //获取每条记录中属性（即列）的个数
       int columnNum = query.record().count();
       //获取"name"属性所在列的编号，列从左向右编号，最左边的编号为0
       int fieldNo = query.record().indexOf("name");
       //获取id属性的值，并转换为int型
```

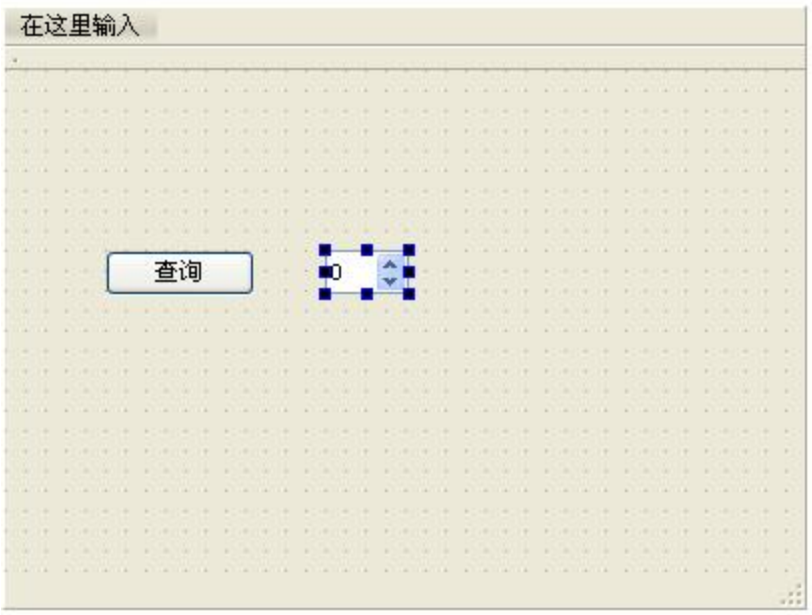
```
        int id = query.value(0).toInt();
        //获取name属性的值
        QString name = query.value(fieldNo).toString();
        //将结果输出
        qDebug() << "rowNum is : " << rowNum
                 << " id is : " << id
                 << " name is : " << name
                 << " columnNum is : " << columnNum;
    }
//定位到结果集中编号为2的记录，即第三条记录，因为第一条记录的编号为0
    qDebug() << "exec seek(2) :";
    if(query.seek(2))
    {
        qDebug() << "rowNum is : " << query.at()
                 << " id is : " << query.value(0).toInt()
                 << " name is : " << query.value(1).toString();
    }
    //定位到结果集中最后一条记录
    qDebug() << "exec last() :";
    if(query.last())
    {
        qDebug() << "rowNum is : " << query.at()
                 << " id is : " << query.value(0).toInt()
                 << " name is : " << query.value(1).toString();
    }
}
```

最后在`mainwindow.cpp`中添加`#include <QSqlRecord>`头文件包含，运行程序，点击查询按钮，输出结果如下图所示。

```
exec next() :
rowNum is :  0  id is :  0  name is :  "first"  columnNum is :  2
exec seek(2) :
rowNum is :  2  id is :  2  name is :  "third"
exec last() :
rowNum is :  4  id is :  4  name is :  "fifth"
```

# 三、在SQL语句中使用变量

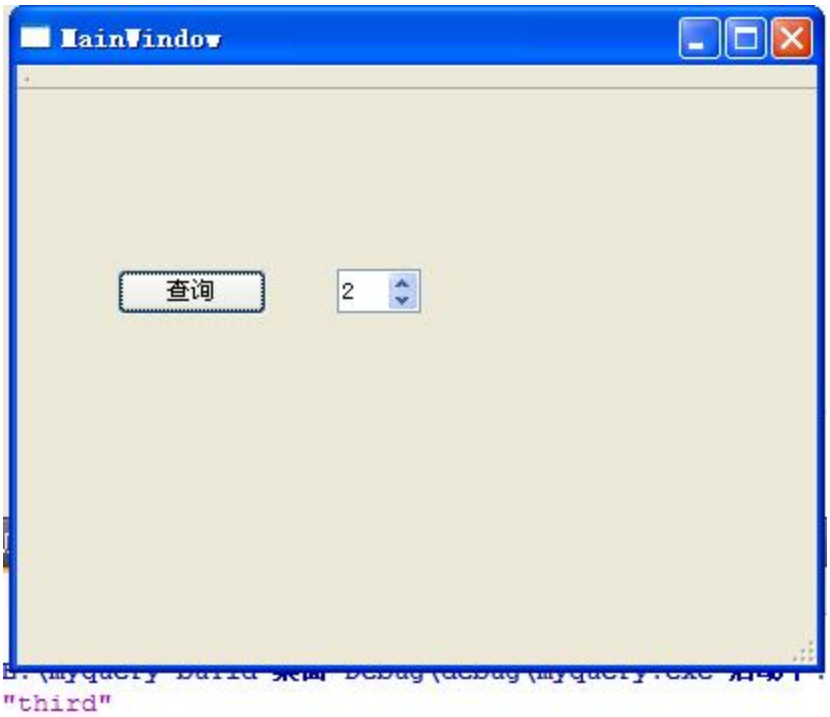
1．我们先来看一个例子。首先在设计模式往界面上添加一个`Spin Box`部件，如下图所示。




2．将查询按钮槽里面的内容更改如下：

```
void MainWindow::on_pushButton_clicked()
{
    QSqlQuery query;
    int id = ui->spinBox->value();
    query.exec(QString("select name from student where id =%1")
               .arg(id));
    query.next();
    QString name = query.value(0).toString();
    qDebug() << name;
}
```

这里使用了`QString`类的`arg()`函数实现了在SQL语句中使用变量，我们运行程序，更改`Spin Box`的值，然后点击查询按钮，效果如下图所示。




3．其实在`QSqlQuery`类中提供了数据绑定同样可以实现在SQL语句中使用变量，虽然它也是通过占位符来实现的，不过使用它形式上更明了一些。下面先来看一个例子，将查询按钮槽更改如下：

```
void MainWindow::on_pushButton_clicked()
{
    QSqlQuery query;
    query.prepare("insert into student (id, name) "
                  "values (:id, :name)");
    query.bindValue(0, 5);
    query.bindValue(1, "sixth");
    query.exec();
    query.exec("select * from student");
    query.last();
    int id = query.value(0).toInt();
    QString name = query.value(1).toString();
    qDebug() << id << name;
}
```

这里在`student`表的最后又添加了一条记录。然后我们先使用了`prepare()`函数，在其中利用了`":id"`和`":name"`来代替具体的数据，而后又利用`bindValue()`函数给`id`和`name`两个属性赋值，这称为绑定操作。其中编号0和1分别代表`":id"`和`":name"`，就是说按照`prepare()`函数中出现的属性从左到右编号，最左边是0 。

特别注意，在最后一定要执行`exec()`函数，所做的操作才能被真正执行。运行程序，点击查询按钮，可以看到前面添加的记录的信息。这里的`":id"`和`":name"`，叫做占位符，这是ODBC数据库的表示方法，还有一种Oracle的表示方法就是全部用“？”号。例如：

```
query.prepare("insert into student(id, name) "
                  "values (?, ?)");
query.bindValue(0, 5);
query.bindValue(1, "sixth");
query.exec();
```

也可以利用`addBindValue()`函数，这样就可以省去编号，它是按顺序给属性赋值的，如下：

```
query.prepare("insert into student(id, name) "
                  "values (?, ?)");
query.addBindValue(5);
query.addBindValue("sixth");
query.exec();
```

当用ODBC的表示方法时，我们也可以将编号用实际的占位符代替，如下：

```
query.prepare("insert into student(id, name) "
                      "values (:id, :name)");
query.bindValue(":id", 5);
query.bindValue(":name", "sixth");
query.exec();
```

以上各种形式的表示方式效果是一样的。

4．下面我们演示一下通过绑定操作在SQL语句中使用变量。更改槽函数如下：

```
void MainWindow::on_pushButton_clicked()
{
    QSqlQuery query;
    query.prepare("select name from student where id = ?");
    int id = ui->spinBox->value();
    query.addBindValue(id);
    query.exec();
    query.next();
    qDebug() << query.value(0).toString();
}
```

运行程序，可以实现通过`Spin Box`的值来进行查询。

# 四、批处理操作

当要进行多条记录的操作时，我们就可以利用绑定进行批处理。将槽更改如下：

```
void MainWindow::on_pushButton_clicked()
{
    QSqlQuery q;
    q.prepare("insert into student values (?, ?)");
    QVariantList ints;
    ints << 10 << 11 << 12 << 13;
    q.addBindValue(ints);
    QVariantList names;
    // 最后一个是空字符串，应与前面的格式相同
        names << "xiaoming" << "xiaoliang"
                      << "xiaogang" << QVariant(QVariant::String);
    q.addBindValue(names);
    if (!q.execBatch()) //进行批处理，如果出错就输出错误
       qDebug() << q.lastError();
    //下面输出整张表
    QSqlQuery query;
    query.exec("select * from student");
    while(query.next())
    {
       int id = query.value(0).toInt();
       QString name = query.value(1).toString();
       qDebug() << id << name;
    }
}
```

然后需要在`mainwindow.cpp`上添加头文件包含：`#include <QSqlError>`。我们在程序中利用列表存储了同一属性的多个值，然后进行了值绑定。最后执行`execBatch()`函数进行批处理。注意程序中利用`QVariant(QVariant::String)`来输入空值`NULL`，因为前面都是`QString`类型的，所以这里要使用`QVariant::String`使格式一致化。 运行程序，效果如下图所示：

```
0 "first"
1 "second"
2 "third"
3 "fourth"
4 "fifth"
10 "xiaoming"
11 "xiaoliang"
12 "xiaogang"
13 ""
```

# 五、事务操作

事务可以保证一个复杂的操作的原子性，就是对于一个数据库操作序列，这些操作要么全部做完，要么一条也不做，它是一个不可分割的工作单位。在Qt中，如果底层的数据库引擎支持事务，那么`QSqlDriver::hasFeature(QSqlDriver::Transactions)`会返回`true`。可以使用`QSqlDatabase::transaction()`来启动一个事务，然后编写一些希望在事务中执行的SQL语句，最后调用`QSqlDatabase::commit()`或者`QSqlDatabase::rollback()`。当使用事务时必须在创建查询以前就开始事务，例如：

```
QSqlDatabase::database().transaction();
QSqlQuery query;
query.exec("SELECT id FROMemployee WHERE name = 'Torild Halvorsen'");
if (query.next()) {
    int employeeId = query.value(0).toInt();
    query.exec("INSERT INTO project(id, name, ownerid) "
               "VALUES (201, 'ManhattanProject', "
               + QString::number(employeeId) + ')');
}
QSqlDatabase::database().commit();
```