



# Qt编程快速入门

DATAGURU专业数据分析社区

# 第十一课：在Qt中执行SQL：数据库编程

## 目录

- 11.1 数据库编程介绍
- 11.2 使用SQL模型类
- 11.3 安装MySQL数据库
- 11.4 小结

## 11.1 数据库编程介绍

- Qt中的Qt SQL模块提供了对数据库的支持，该模块中的众多类基本上可以分为三层：

用户接口层	QSqlQueryModel、QSqlTableModel 和 QSqlRelationalTableModel
SQL 接口层	QSqlDatabase、QSqlQuery、QSqlError、QSqlField、QSqlIndex 和 QSqlRecord
驱动层	QSqlDriver、QSqlDriverCreator、QSqlDriverCreatorBase、QSqlDriverPlugin 和 QSqlResult

- 驱动层为具体的数据库和SQL接口层之间提供了底层的桥梁；SQL接口层提供了对数据库的访问，其中的QSqlDatabase类用来创建连接，QSqlQuery类可以使用SQL语句来实现与数据库交互，其他几个类对该层提供了支持；用户接口层的几个类实现了将数据库中的数据链接到窗口部件上，这些类是使用前一章的模型/视图框架实现的，它们是更高层次的抽象。
- 如果要使用Qt SQL模块中的这些类，需要在项目文件（.pro文件）中添加“QT += sql”这一行代码。

- Qt SQL模块使用数据库驱动插件来和不同的数据库接口进行通信。由于Qt SQL模块的接口是独立于数据库的，所以所有数据库特定的代码都包含在了这些驱动中。Qt默认支持一些驱动：

驱动名称	数据库
QDB2	IBM DB2（7.1 版或者以上版本）
QIBASE	Borland InterBase
QMYSQL	MySQL
QOCI	Oracle Call Interface Driver
QODBC	Open Database Connectivity(ODBC)-微软 SQL Server 和其他 ODBC 兼容数据库
QPSQL	PostgreSQL（7.3 版本或者更高）
QSQLITE2	SQLite 版本 2
QSQLITE	SQLite 版本 3
QTDS	Sybase Adaptive Server 注：从 Qt 4.7 开始已经过时

- 这里要重点提一下SQLite数据库，它是一款轻型的文件型数据库，无需数据库服务器，主要应用于嵌入式领域，支持跨平台，而且Qt对它提供了很好的默认支持。

# 示例：遍历输出驱动列表

```
#include <QApplication>
#include <QSqlDatabase>
#include <QDebug>
#include <QStringList>

int main(int argc, char *argv[])
{
    QApplication a(argc, argv);
    qDebug() << "Available drivers:";
    QStringList drivers = QSqlDatabase::drivers();
    foreach(QString driver, drivers)
        qDebug() << driver;
    return a.exec();
}
```

- 要想使用QSqlQuery或者QSqlQueryModel来访问数据库，那么先要创建并打开一个或者多个数据库连接。数据库连接使用连接名来定义，而不是使用数据库名，可以向相同的数据库创建多个连接。QSqlDatabase也支持默认连接的概念，默认连接就是一个没有命名的连接。在使用QSqlQuery或者QSqlQueryModel的成员函数时需要指定一个连接名作为参数，如果没有指定，那么就会使用默认连接。如果在应用程序中只需要有一个数据库连接，那么使用默认连接是很方便的。示例：

原型：QSqlDatabase QSqlDatabase::addDatabase(const QString &type, const QString &connectionName = QLatin1String(defaultConnection))

```
QSqlDatabase db = QSqlDatabase::addDatabase("QMYSQL");  
db.setHostName("bigblue");  
db.setDatabaseName("flightdb");  
db.setUserName("acarlson");  
db.setPassword("1uTbSbAs");  
bool ok = db.open();
```

- 下面的示例代码中创建了两个名为 “first” 和 “second” 的连接：

```
QSqlDatabase firstDB = QSqlDatabase::addDatabase("QMYSQL", "first");
```

```
QSqlDatabase secondDB = QSqlDatabase::addDatabase("QMYSQL", "second");
```

- 创建完连接后，可以在任何地方使用QSqlDatabase::database()静态函数通过连接名称获取指向数据库连接的指针，如果调用该函数时没有指明连接名称，那么会返回默认连接，例如：

```
QSqlDatabase defaultDB = QSqlDatabase::database();
```

```
QSqlDatabase firstDB = QSqlDatabase::database("first");
```

```
QSqlDatabase secondDB = QSqlDatabase::database("second");
```

- 要移除一个数据库连接，需要先使用QSqlDatabase::close()关闭数据库，然后使用静态函数QSqlDatabase::removeDatabase()移除该连接。



- QSqlQuery类提供了一个接口，用于执行SQL语句和浏览查询的结果集。要执行一个SQL语句，只需要简单的创建一个QSqlQuery对象，然后调用QSqlQuery::exec()函数即可。例如：

```
QSqlQuery query;
```

```
query.exec("select * from student");
```

- QSqlQuery提供了对结果集的访问，可以一次访问一条记录。当执行完exec()函数后，QSqlQuery的内部指针会位于第一条记录前面的位置。必须调用一次QSqlQuery::next()函数来使其前进到第一条记录，然后可以重复使用next()函数来访问其他的记录，直到该函数的返回值为false，例如可以使用以下代码来遍历一个结果集：

```
while(query.next())
```

```
{ qDebug() << query.value(0).toInt() << query.value(1).toString(); }
```

- 在QSqlQuery类中提供了多个函数来实现在结果集中进行定位，比如next()定位到下一条记录，previous()定位到前一条记录，first()定位的第一条记录，last()定位到最后一条记录，seek(n)定位到第n条记录。当前行的索引可以使用at()返回；record()函数可以返回当前指向的记录；如果数据库支持，那么可以使用size()来返回结果集中的总行数。



- 插入一条记录：`query2.exec("insert into student (id, name) values (100, 'ChenYun')");`
- 如果想在同一时间插入多条记录，那么一个有效的方法就是将查询语句和真实的值分离，这可以使用占位符来完成。Qt支持两种占位符：名称绑定和位置绑定。

名称绑定：

```
query2.prepare("insert into student (id, name) values (:id, :name)");  
int idValue = 100;  
QString nameValue = "ChenYun";  
query2.bindValue(":id", idValue);  
query2.bindValue(":name", nameValue);  
query2.exec();
```

位置绑定：

```
query2.prepare("insert into student (id, name) values (?, ?)");  
int idValue = 100;  
QString nameValue = "ChenYun";  
query2.addBindValue(idValue);  
query2.addBindValue(nameValue);  
query2.exec();
```

- 当要插入多条记录时，只需要调用 QSqlQuery::prepare() 一次，然后使用多次 bindValue() 或者 addBindValue() 函数来绑定需要的数据，最后调用一次 exec() 函数就可以了。其实，进行多条数据插入时，还可以使用批处理进行：

```
query2.prepare("insert into student (id, name) values (?, ?)");
```

```
QVariantList ids;
```

```
ids << 20 << 21 << 22;
```

```
query2.addBindValue(ids);
```

```
QVariantList names;
```

```
names << "xiaoming" << "xiaoliang" << "xiaogang";
```

```
query2.addBindValue(names);
```

```
if(!query2.execBatch()) qDebug() << query2.lastError();
```

- 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()提交或者QSqlDatabase::rollback()回滚。当使用事务时必须在创建查询以前就开始事务。

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

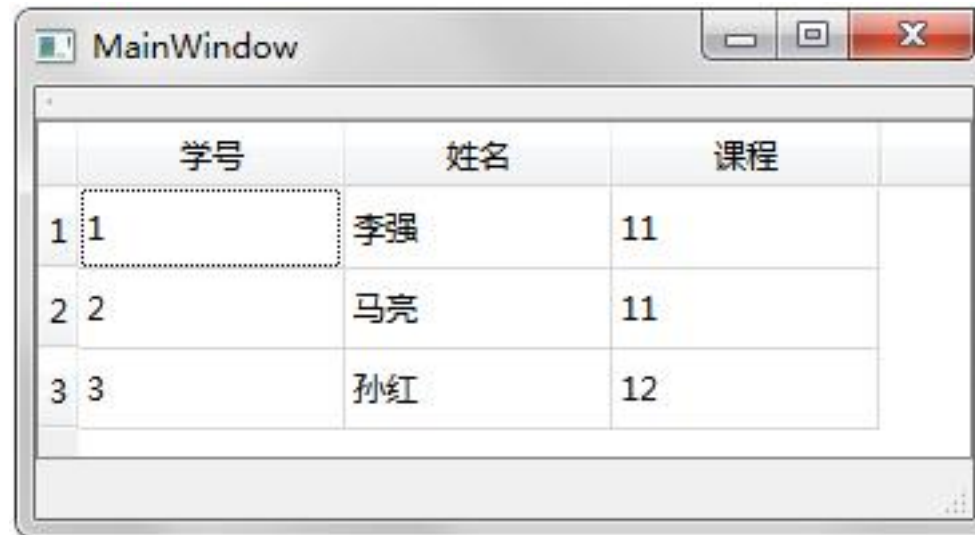
## 11.2 使用SQL模型类

- 除了 QSqlQuery，Qt 还提供了 3 个更高层的类来访问数据库，分别是 QSqlQueryModel、QSqlTableModel 和 QSqlRelationalTableModel。
- 这 3 个类都是从 QAbstractTableModel 派生来的，可以很容易地实现将数据库中的数据在 QListView 和 QTableView 等项视图类中进行显示。使用这些类的另一个好处是，这样可以使编写的代码很容易的适应其他的数据源。例如，如果开始使用了 QSqlTableModel，而后来要改为使用 XML 文件来存储数据，这样需要做的仅是更换一个数据模型。

# SQL查询模型 QSqlQueryModel

```
QSqlQueryModel *model = new QSqlQueryModel(this);  
model->setQuery("select * from student");  
model->setHeaderData(0, Qt::Horizontal, tr("学号"));  
model->setHeaderData(1, Qt::Horizontal, tr("姓名"));  
model->setHeaderData(2, Qt::Horizontal, tr("课程"));
```

```
QTableView *view = new QTableView(this);  
view->setModel(model);  
setCentralWidget(view);
```

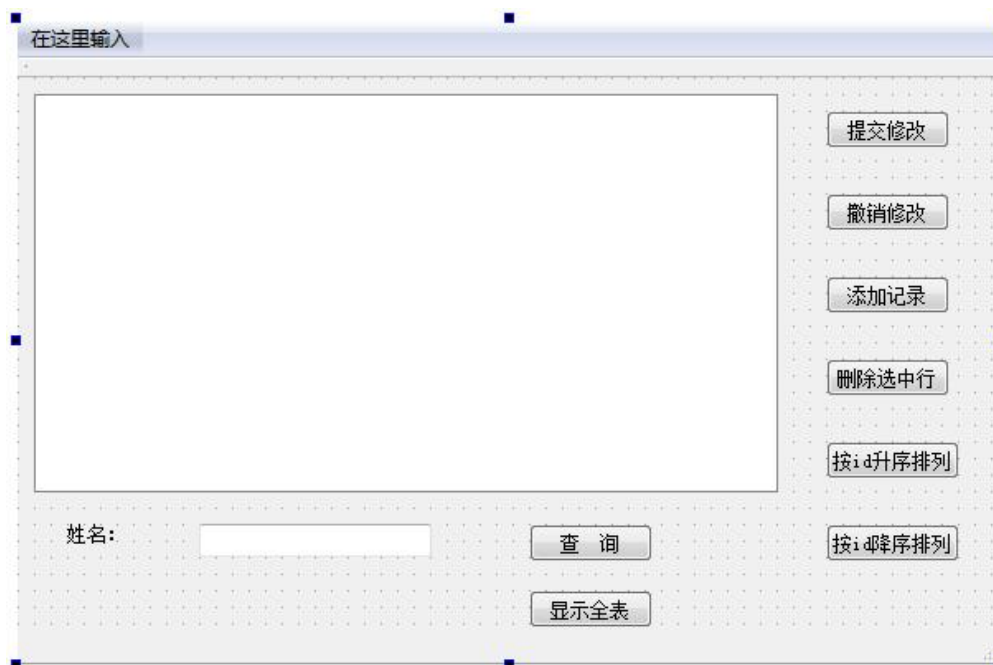


The screenshot shows a Qt application window titled "MainWindow". Inside the window is a QTableView displaying a table with three columns: "学号" (Student ID), "姓名" (Name), and "课程" (Course). The table contains three rows of data. The first row has student ID 1, name 李强, and course 11. The second row has student ID 2, name 马亮, and course 11. The third row has student ID 3, name 孙红, and course 12. The first cell of the first row (1, 1) is selected, indicated by a dashed border.

	学号	姓名	课程
1	1	李强	11
2	2	马亮	11
3	3	孙红	12

这里先创建了QSqlQueryModel对象，然后使用setQuery()来执行SQL语句查询整张student表，并使用setHeaderData()来设置显示的标头。后面创建了视图，并将QSqlQueryModel对象作为其要显示的模型。这里要注意，其实QSqlQueryModel中存储的是执行完setQuery()函数后的结果集，所以视图中显示的是结果集的内容。QSqlQueryModel中还提供了columnCount()返回一条记录中字段的个数；rowCount()返回结果集中记录的条数；record()返回第n条记录；index()返回指定记录的指定字段的索引；clear()可以清空模型中的结果集。

- QSqlTableModel提供了一个一次只能操作一个SQL表的读写模型，它是QSqlQuery的更高层次的替代品，可以浏览和修改独立的SQL表，并且只需编写很少的代码，而且不需要了解SQL语法。



```
QSqlQuery query;  
// 创建student表  
query.exec("create table student (id int primary key, "  
           "name varchar, course int)");  
query.exec("insert into student values(1, '李强', 11)");  
query.exec("insert into student values(2, '马亮', 11)");  
query.exec("insert into student values(3, '孙红', 12)");  
  
// 创建course表  
query.exec("create table course (id int primary key, "  
           "name varchar, teacher varchar)");  
query.exec("insert into course values(10, '数学', '王老师')");  
query.exec("insert into course values(11, '英语', '张老师')");  
query.exec("insert into course values(12, '计算机', '白老师')");
```



## ■ 显示表：

```
model = new QSqlTableModel(this);
```

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

```
model->select();
```

```
// 设置编辑策略
```

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

```
ui->tableView->setModel(model);
```

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

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枚举类型定义

```
// 提交修改按钮
void MainWindow::on_pushButton_clicked()
{
    // 开始事务操作
    model->database().transaction();
    if (model->submitAll()) {
        if(model->database().commit()) // 提交
            QMessageBox::information(this, tr("tableModel"),
                                     tr("数据修改成功！"));
    } else {
        model->database().rollback(); // 回滚
        QMessageBox::warning(this, tr("tableModel"),
                             tr("数据库错误: %1").arg(model->lastError().text()),
                             QMessageBox::Ok);
    }
}

// 撤销修改按钮
void MainWindow::on_pushButton_2_clicked()
{
    model->revertAll();
}
```

// 查询按钮，进行筛选

```
void MainWindow::on_pushButton_5_clicked()
{
    QString name = ui->lineEdit->text();
    // 根据姓名进行筛选，一定要使用单引号
    model->setFilter(QString("name = '%1']").arg(name));
    model->select();
}
```

// 显示全表按钮

```
void MainWindow::on_pushButton_6_clicked()
{
    model->setTable("student");
    model->select();
}
```

// 按id升序排列按钮

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

// 按id降序排列按钮

```
void MainWindow::on_pushButton_8_clicked()
{
    model->setSort(0, Qt::DescendingOrder);
    model->select();
}
```

```
// 删除选中行按钮
void MainWindow::on_pushButton_4_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)
    { // 如果不删除，则撤销
        model->revertAll();
    } else { // 否则提交，在数据库中删除该行
        model->submitAll();
    }
}
```

```
// 添加记录按钮
void MainWindow::on_pushButton_3_clicked()
{
    // 获得表的行数
    int rowNum = model->rowCount();
    int id = 10;

    // 添加一行
    model->insertRow(rowNum);
    model->setData(model->index(rowNum,0), id);

    // 可以直接提交
    //model->submitAll();
}
```

# SQL关系表格模型 QSqlRelationalTableModel

- QSqlRelationalTableModel继承自QSqlTableModel，并且对其进行了扩展，提供了对外键的支持。一个外键就是一个表中的一个字段和其他表中的主键字段之间的一对一的映射。例如，student表中的course字段对应的是course表中的id字段，那么就称字段course是一个外键。因为这里的course字段的值是一些数字，这样的显示很不友好，使用关系表格模型，就可以将它显示为course表中的name字段的值。

```
QSqlRelationalTableModel *model = new QSqlRelationalTableModel(this);
```

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

```
model->setRelation(2, QSqlRelation("course", "id", "name"));
```

```
model->select();
```

```
QTableView *view = new QTableView(this);
```

```
view->setModel(model);
```

```
setCentralWidget(view);
```

- Qt中还提供了一个QSqlRelationalDelegate委托类，它可以为QSqlRelationalTableModel显示和编辑数据。这个委托为一个外键提供了一个QComboBox部件来显示所有可选的数据，这样就显得更加人性化了。

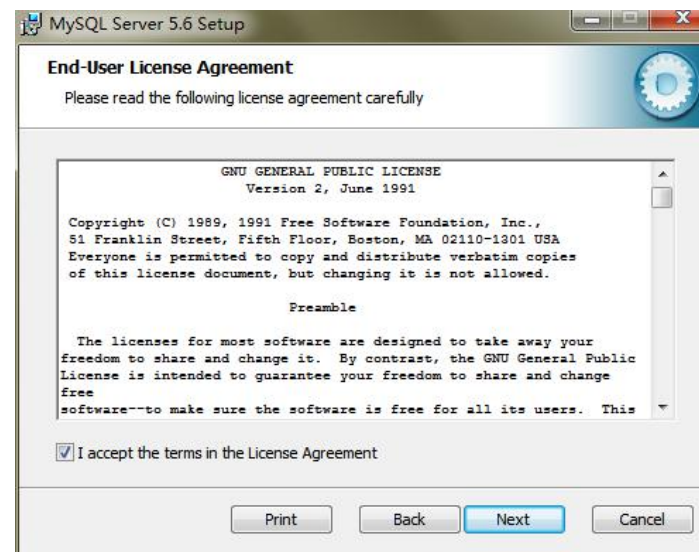
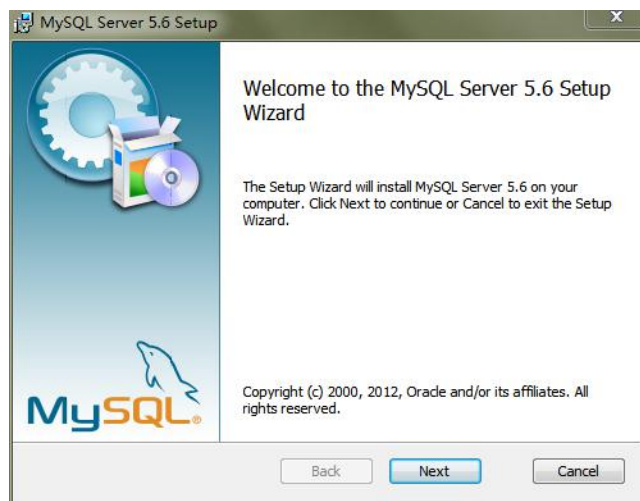
```
view->setItemDelegate(new QSqlRelationalDelegate(view));
```





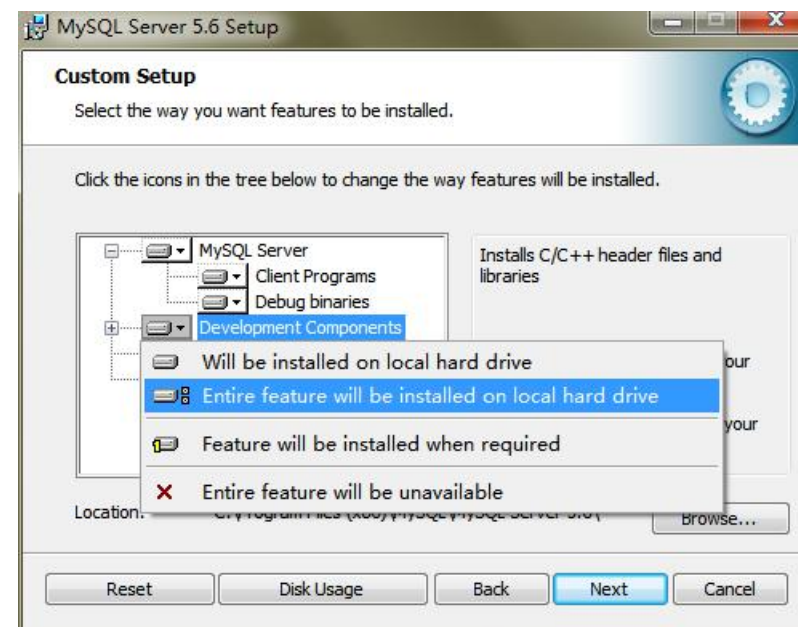
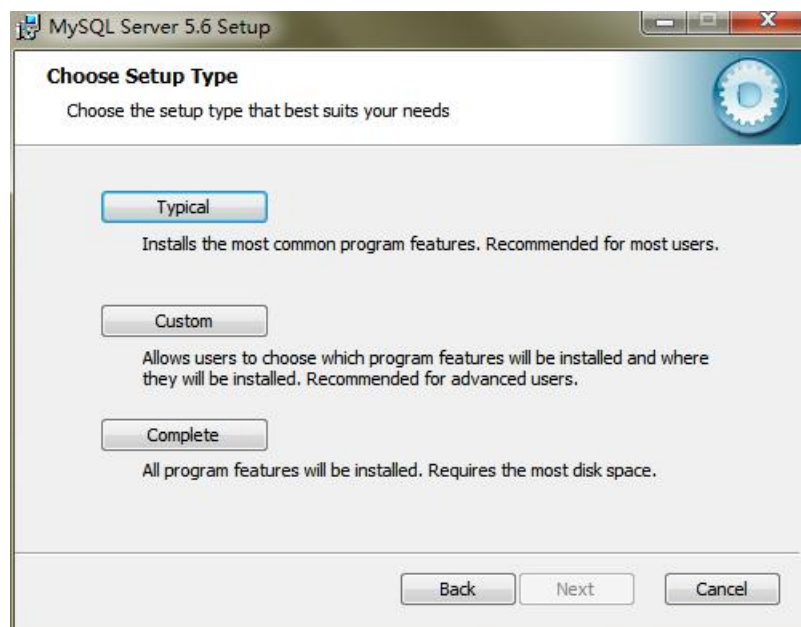
## 11.3 安装MySQL数据库

- 下载：可以从<http://www.qter.org/portal.php?mod=view&aid=10>下载MySQL安装包，具体文件为mysql-5.6.10-win32。
- 运行下载的安装包，首先出现的是向导欢迎界面。如下图所示。单击“Next”按钮。
- 该界面选择同意条款。单击“Next”按钮。

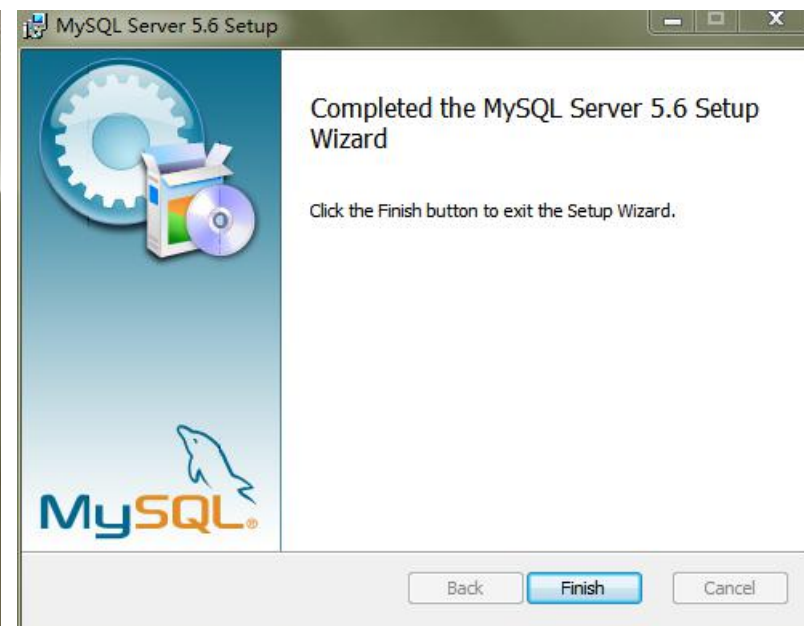
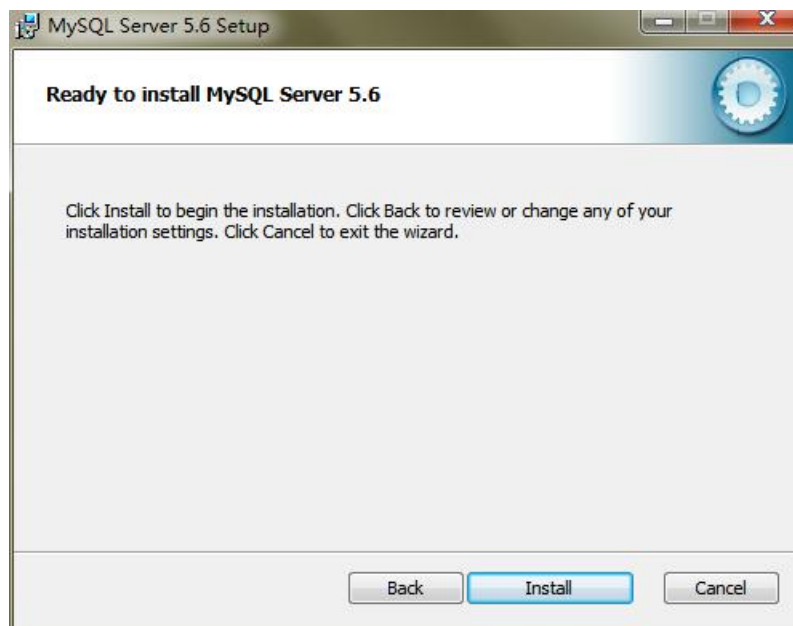
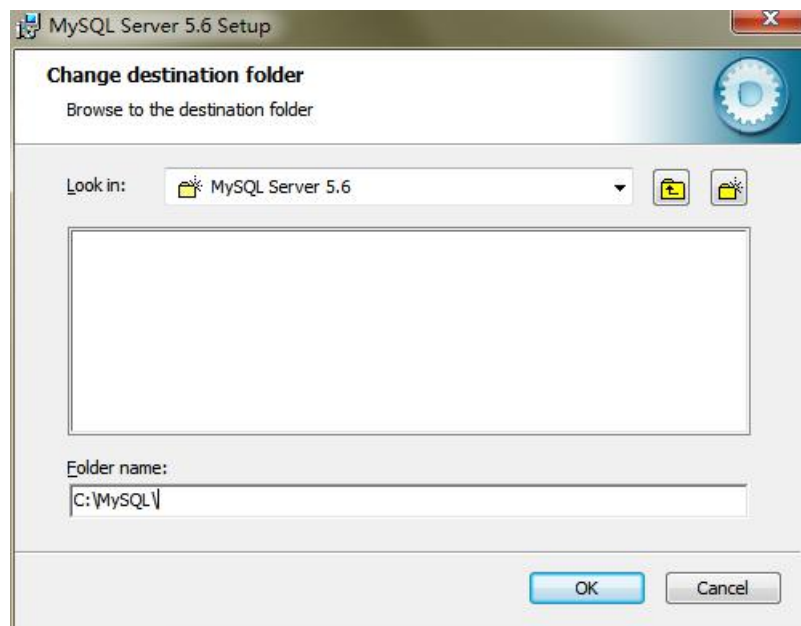


# 安装MySQL数据库

- 下面选择定制安装“Custom”。这里需要安装所有的头文件和库，点击Development Components前面的下拉箭头，然后选择第二项。

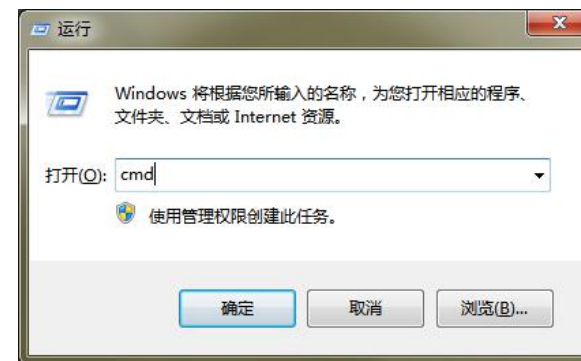
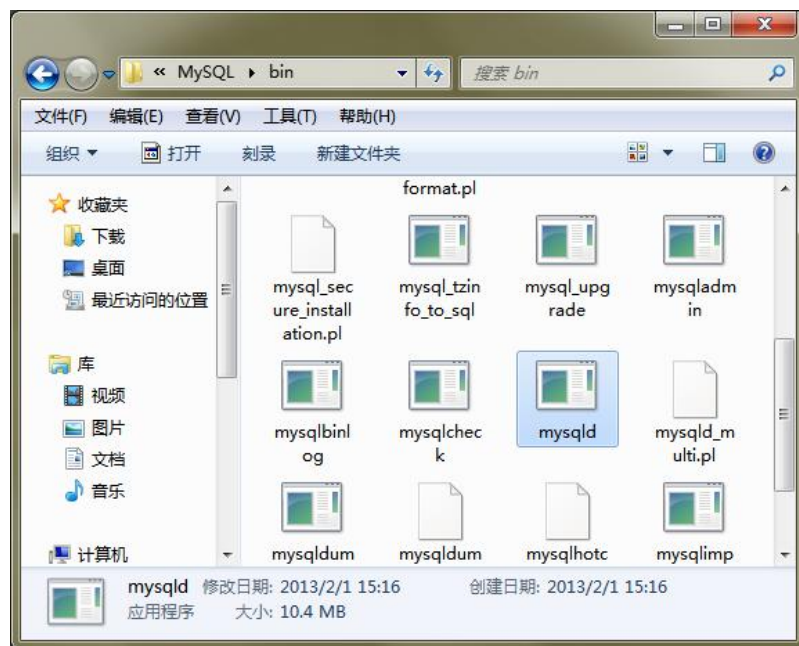


- 然后选择下面的 “Browse...” 按钮来更改安装路径，这里设置为C:\MySQL\。
- 填写完路径后单击 “OK” 按钮回到主页面，单击 “Next” 按钮来到新的页面，这里单击 “Install” 按钮开始安装。
- 等安装完毕后，点击 “Finish” 按钮完成安装。



# 在MySQL中创建数据库

- 下面先在安装的MySQL中创建一个数据库，用于后面的测试。首先到MySQL的安装目录C:\MySQL\bin目录下运行mysqld.exe程序，该程序运行完成后会自动关闭。
- 同时按下键盘上的Win图标和R键，在弹出的对话框中输入cmd。



# 在MySQL中创建数据库

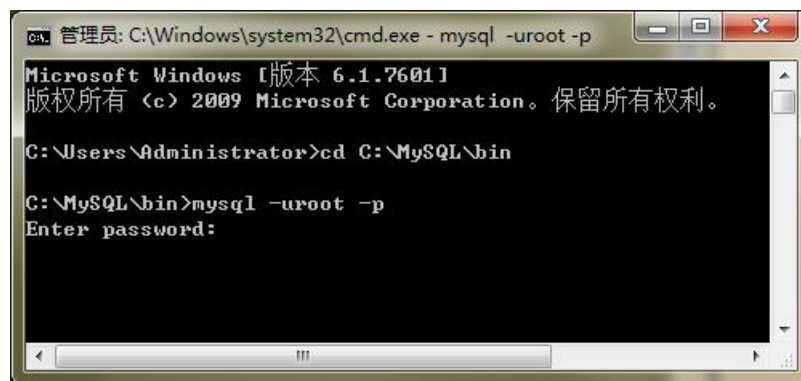
- 进入终端后输入下面的命令：`cd C:\MySQL\bin` 跳转到安装目录下。
- 然后输入下面的命令：`mysql -uroot -p` 使用root用户来登陆MySQL，因为默认密码是空的，所以这里不用设置密码。运行这行代码会提示Enter password，这时敲回车即可。
- 登录MySQL以后，使用下面的命令来查看现有的数据库：`show databases;` 注意后面有个分号。



```
C:\Windows\system32\cmd.exe
Microsoft Windows [版本 6.1.7601]
版权所有 (c) 2009 Microsoft Corporation。保留所有权利。

C:\Users\Administrator>cd C:\MySQL\bin

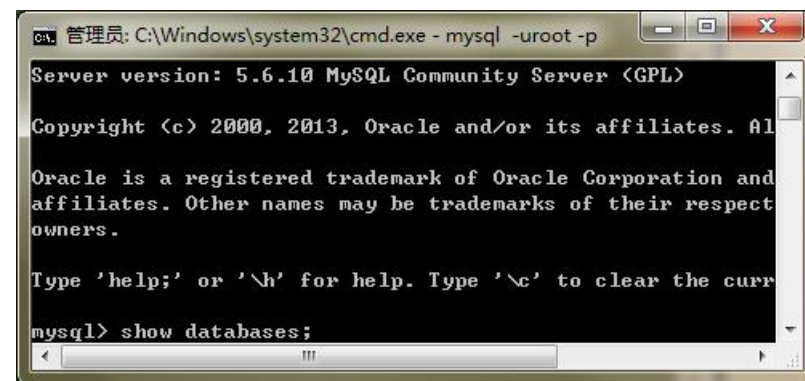
C:\MySQL\bin>
```



```
C:\Windows\system32\cmd.exe - mysql -uroot -p
Microsoft Windows [版本 6.1.7601]
版权所有 (c) 2009 Microsoft Corporation。保留所有权利。

C:\Users\Administrator>cd C:\MySQL\bin

C:\MySQL\bin>mysql -uroot -p
Enter password:
```



```
C:\Windows\system32\cmd.exe - mysql -uroot -p
Server version: 5.6.10 MySQL Community Server (GPL)

Copyright (c) 2000, 2013, Oracle and/or its affiliates. All rights reserved.

Oracle is a registered trademark of Oracle Corporation and/or its
affiliates. Other names may be trademarks of their respective
owners.

Type 'help;' or '\h' for help. Type '\c' to clear the current input statement.

mysql> show databases;
```



# 在MySQL中创建数据库

- 可以看到，这里现在已经有几个数据库了，它们是MySQL需要的。
- 这里不使用已经有的数据库，而是新建自己的数据库，下面新建名为mydata的数据库：create database mydata；
- 再次查看已经存在的数据库，发现显示出了刚才创建的数据库。
- 完成后，输入exit命令退出MySQL。

```
管理员: C:\Windows\system32\cmd.exe - mysql -uroot -p
mysql> show databases;
+-----+
| Database |
+-----+
| information_schema |
| mysql |
| performance_schema |
+-----+
3 rows in set (0.00 sec)

mysql>
```

```
管理员: C:\Windows\system32\cmd.exe - mysql -uroot -p
+-----+
| information_schema |
| mysql |
| performance_schema |
+-----+
3 rows in set (0.00 sec)

mysql> create database mydata;
Query OK, 1 row affected (0.00 sec)

mysql>
```

```
管理员: C:\Windows\system32\cmd.exe - mysql -uroot -p
Query OK, 1 row affected (0.00 sec)

mysql> show databases;
+-----+
| Database |
+-----+
| information_schema |
| mydata |
| mysql |
| performance_schema |
+-----+
4 rows in set (0.00 sec)

mysql>
```

```
#include <QCoreApplication>
#include <QSqlDatabase>
#include <QDebug>
#include <QSqlQuery>

int main(int argc, char *argv[])
{
    QCoreApplication a(argc, argv);

    // 打开MySQL
    QSqlDatabase db = QSqlDatabase::addDatabase("QMYSQL");
    db.setHostName("localhost");
    db.setDatabaseName("mydata");
    db.setUserName("root");
    db.setPassword("");
    if (!db.open())
        qDebug() << "Failed to connect to root mysql admin";
    else qDebug() << "open";
```

```
QSqlQuery query(db);
```

//注意这里varchar一定要指定长度，不然会出错

```
query.exec("create table student(id int primary key,name varchar(20))");
```

```
query.exec("insert into student values(1,'xiaogang')");
```

```
query.exec("insert into student values(2,'xiaoming')");
```

```
query.exec("insert into student values(3,'xiaohong')");
```

```
query.exec("select id,name from student where id >= 2");
```

```
while(query.next())
```

```
{
```

```
    int value0 = query.value(0).toInt();
```

```
    QString value1 = query.value(1).toString();
```

```
    qDebug() << value0 << value1 ;
```

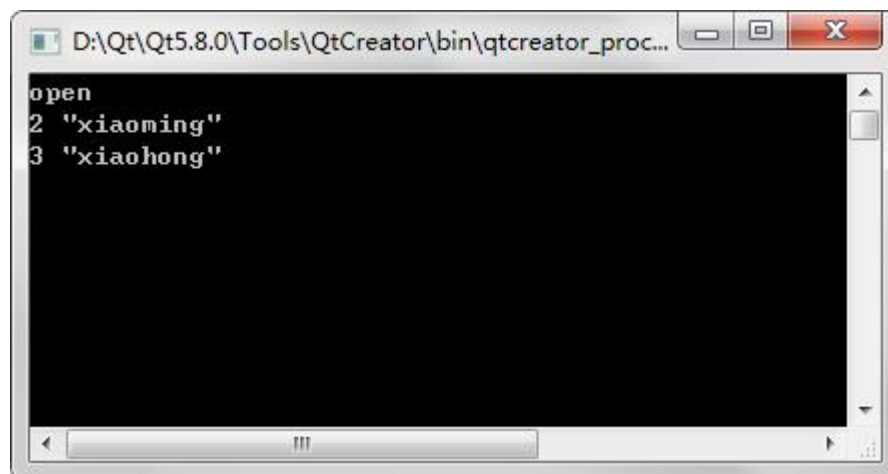
```
}
```

```
return a.exec();
```

```
}
```



- 要得到C:\MySQL\lib中将libmysql.dll文件复制到C:\Qt\Qt5.8.0\5.8\mingw53\_32\bin中，然后再运行程序。
- 发布程序时也要复制该文件。



```
D:\Qt\Qt5.8.0\Tools\QtCreator\bin\qtc_reactor_proc...  
open  
2 "xiaoming"  
3 "xiaohong"
```

## 11.4 小结

- 本章应该了解的内容
  - Qt支持哪些数据库
  - SQLite数据库的优点
- 本章应该掌握的内容
  - 在Qt中如何使用SQL语句操作数据库
  - 3个SQL模型类的使用

**【声明】** 本视频和幻灯片为炼数成金网络课程的教学资料，所有资料只能在课程内使用，不得在课程以外范围散播，违者将可能被追究法律和经济责任。

课程详情访问炼数成金培训网站

<http://edu.dataguru.cn>

- Dataguru ( 炼数成金 ) 是专业数据分析网站 , 提供教育 , 媒体 , 内容 , 社区 , 出版 , 数据分析业务等服务。我们的课程采用新兴的互联网教育形式 , 独创地发展了逆向收费式网络培训课程模式。既继承传统教育重学习氛围 , 重竞争压力的特点 , 同时又发挥互联网的威力打破时空限制 , 把天南地北志同道合的朋友组织在一起交流学习 , 使到原先孤立的学习个体组合成有组织的探索力量。并且把原先动辄成千上万的学习成本 , 直线下降至百元范围 , 造福大众。我们的目标是 : 低成本传播高价值知识 , 构架中国第一的网上知识流转阵地。
- 关于逆向收费式网络的详情 , 请看我们的培训网站 <http://edu.dataguru.cn>

# Thanks

**FAQ时间**