

除了 QSqlQuery，Qt 提供了访问数据库的 3 个高级类。这些类是 QSqlQueryModel，QSqlTableModel 和 QSqlRelationalTableModel。

- QSqlQueryModel 基于任意 SQL 查询 QSqlQueryModel 只读模式。
- QSqlTableModel 单个数据表的读写模式。
- QSqlRelationalTableModel QSqlTableModel 的一个子类，与外键的支持。

这些类从 QAbstractTableModel（这反过来又继承了 QAbstractItemModel）派生并可以很容易地从项目视图类中的数据库显示数据，例如 QListView 和 QTableView。详细信息在 Presenting Data in a Table View 部分有详细说明。

使用这些类的另一个优点是，它可以使你的代码更容易适应其他数据源。例如，如果使用 QSqlTableModel，后来决定使用 XML 文件来存储，而不是数据库，它本质上只是与另一种替代一个数据模型。

SQL 查询模式

QSqlQueryModel 提供基于 SQL 查询的只读模式。

```
QSqlQueryModel model;
model.setQuery("SELECT * FROM employee");

for (int i = 0; i < model.rowCount(); ++i) {
    int id = model.record(i).value("id").toInt();
    QString name = model.record(i).value("name").toString();
    qDebug() << id << name;
}
```

使用 QSqlQueryModel::setQuery() 设置查询后，就可以使用 QSqlQueryModel::record (int) 来访问个人记录。您还可以使用 QSqlQueryModel::data() 和任何其它继承自 QAbstractItemModel 的函数。

还有一个需要一个 QSqlQuery 对象和结果集的 setQuery() 重载。这使您可以使用 QSqlQuery 的任何功能，以设置查询（例如，准备查询）。

SQL 表模型

QSqlTableModel 提供了工作在单个表的读写模式。

例：

```
QSqlTableModel model;
model.setTable("employee");
model.setFilter("salary > 50000");
model.setSort(2, Qt::DescendingOrder);
model.select();
```

```
for (int i = 0; i < model.rowCount(); ++i) {
    QString name = model.record(i).value("name").toString();
    int salary = model.record(i).value("salary").toInt();
    qDebug() << name << salary;
}
```

QSqlTableModel 是一个高层次的替代 QSqlQuery 浏览和修改单个 SQL 表。它通常代码量少，不需要任何 SQL 语法的知识。

使用 QSqlTableModel::record() 来检索表中的行，并与 QSqlTableModel::setRecord() 来修改行。例如，下面的代码会增加每个员工的工资按 10%：

```
for (int i = 0; i < model.rowCount(); ++i) {
    QSqlRecord record = model.record(i);
    double salary = record.value("salary").toInt();
    salary *= 1.1;
    record.setValue("salary", salary);
    model.setRecord(i, record);
}
model.submitAll();
```

您还可以使用 QSqlTableModel::data() 和 QSqlTableModel::setData() 访问数据。例如，这里是如何使用更新使用 setData() 的记录：

```
model.setData(model.index(row, column), 75000);
model.submitAll();
```

以下是如何插入一行，并填充它：

```
model.insertRows(row, 1);
model.setData(model.index(row, 0), 1013);
model.setData(model.index(row, 1), "Peter Gordon");
model.setData(model.index(row, 2), 68500);
model.submitAll();
```

以下是如何删除连续五行：

```
model.removeRows(row, 5);
model.submitAll();
```

QSqlTableModel::removeRows() 的第一个参数是删除第一行的索引。

当你更改记录完成，你应该总是调用与 `QSqlTableModel :: submitAll()`，以确保更改写入到数据库中。

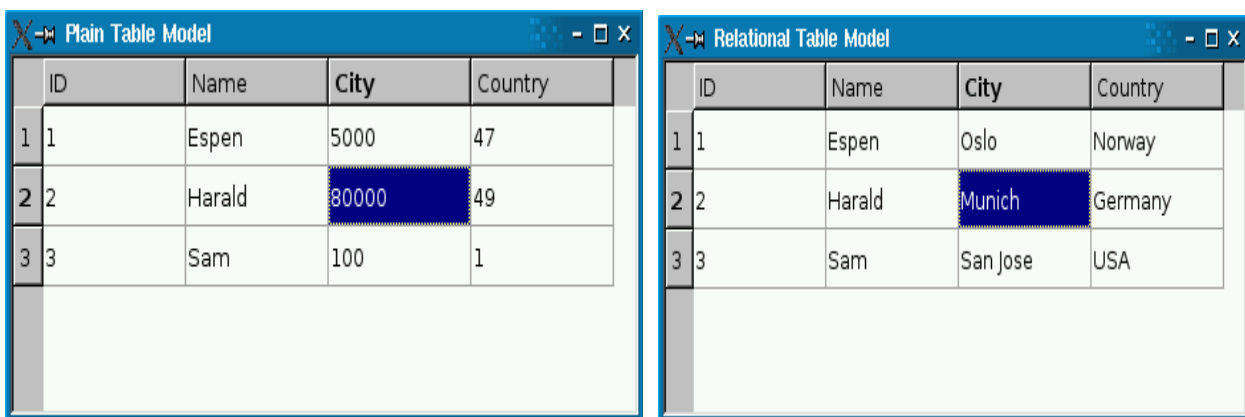
什么时候需要调用 `submitAll()` 取决于表的编辑策略（edit strategy）。默认的策略是与 `QSqlTableModel :: OnRowChange`，当用户选择不同的行时，它指定挂起的更改应用到数据库中。其他的策略是与 `QSqlTableModel :: OnManualSubmit`（其中所有的变化都缓存在模型中，直到调用 `submitAll()`），以及 `QSqlTableModel :: OnFieldChange`（没有改变缓存）。这些大多是使用 `QSqlTableModel` 时有用。

`QSqlTableModel :: OnFieldChange` 似乎传递你永远不需要调用 `submitAll()` 明确的承诺。但有两个缺陷：

- 无需任何缓存时，性能可能显著下降。
- 如果修改主键，记录可能会通过你的手指打滑，而你正试图填充它。

SQL 关系表模型

`QSqlRelationalTableModel` 是 `QSqlTableModel` 的延伸，提供外键的支持。外键是在一个表字段和另一个表主键字段之间的 1 对 1 的映射。例如，如果一本书表有一个字段名为 `AUTHORID`，它是指作者表的 `id` 字段，我们说 `AUTHORID` 是一个外键。



	ID	Name	City	Country
1	1	Espen	5000	47
2	2	Harald	80000	49
3	3	Sam	100	1

	ID	Name	City	Country
1	1	Espen	Oslo	Norway
2	2	Harald	Munich	Germany
3	3	Sam	San Jose	USA

左边的屏幕截图在一个 `QTableView` 中显示一个无格式的 `QSqlTableModel`。外键（城市和国家）都不是人类可读的值。右边的屏幕截图显示了 `QSqlRelationalTableModel`，与外键分解成可读的文本字符串。

下面的代码片段显示了 `QSqlRelationalTableModel` 是如何设置：

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

model->setRelation(2, QSqlRelation("city", "id", "name"));
model->setRelation(3, QSqlRelation("country", "id", "name"));
```

请参阅 `QSqlRelationalTableModel` 文档的详细信息。