Junior C++ developer

Лекция 10

- Qt SQLite

SQLite

Встраиваемая кроссплатформенная БД, которая поддерживает достаточно полный набор команд SQL и доступна в исходных кодах (на языке C).

Исходные коды SQLite находятся в public domain, то есть вообще никаких ограничений на использование.

SQLite

- Приложение является сервером
- Доступ через подключение к БД
- Множество подключений к одной ДБ в одном или нескольких приложений

Qt + SQLite

В .pro файл необходимо подключить модуль

```
QT += sql
```

Для работы с классами этого модуля необходимо подключить

```
#include <QtSql>
```

Уровни модуля

- Уровень драйверов
- Программный уровень
- Уровень пользовательского интерфейса

Программный уровень

Для соединения с базой данных прежде всего нужно активизировать драйвер используя статический метод *QSqlDatabase::addDatabase()*. Метод получает строку как аргумент, обозначающий идентификатор драйвер СУБД. Нам понадобится «QSQLITE».

```
QSqlDatabase sdb = QSqlDatabase::addDatabase( "QSQLITE");
sdb.setDatabaseName("db_name.sqlite");

if (!sdb.open()) {
        //....
}
```

Соединение осуществляется методом ***open()***. Класс **QSqlDatabase** представляет соединение с БД. Соединение предоставляет доступ к БД через поддерживаемый драйвер БД. Важно, что можно иметь несколько соединений к одной БД.

Если при соединении (метод **open()**) возникла ошибка, то получить информацию об ошибке можно через метод Q**SqlDatabase::lastError()** (возвращает **QSqlError**).

```
if (!sdb.open()) {
      qDebug() << sdb.lastError().text();
}
```

QSqlQuery

Позволяет выполнять:

- SELECT, INSERT, UPDATE, DELETE
- CREATE TABLE, ALERT, DROP

Query

```
QSqlQuery query("SELECT country FROM artist");
while (query.next()) {
        QString country = query.value(0).toString();
        do_something(country);
}
```

После запроса **SELECT** можно перемещаться по собранным данным при помощи методов ***next()***, ***previous()***, ***first()***, ***last()*** и ***seek()***.

Метод ***next()*** позволяет перемещаться на следующую строку данных, а вызов ***previous()*** на предыдущую строку, соответственно. ***first()***, ***last()*** извлекают, соответственно, первую запись из результата.

Время выполнения запросов

```
QSqlQuery query("SELECT country FROM artist");
```

Или

```
QSqlQuery query;
query.exec("SELECT country FROM artist");
```

exec() получает запрос в виде ***QString***

prepare

```
QSqlQuery my_query;
my_query.prepare("INSERT INTO my_table (number, address, age)"
                                 "VALUES (:number, :address, :age);" );
my_query.bindValue(":number", "14");
my_query.bindValue(":address", "hello world str.");
my_query.bindValue(":age", "37");
```

С помощью prerare можно подставлять в запрос значения полученные на этапе выполнения.

Безымянные параметры

```
QSqlQuery my_query;
my_query.prepare("INSERT INTO my_table (number, address, age)"
                                 "VALUES (?, ?, ?);");
my_query.bindValue("14");
my_query.bindValue("hello world str.");
my_query.bindValue("37");
```

Параметры из QString

```
QSqlQuery my_query;
my_query.prepare(
          QString("INSERT INTO my_table (number, address, age) VALUES (%1,
'%2', %3);")
                              .arg( "14").arg("hello world str.").arg("37")
          );
```