Лабораторная работа № 7 (легкий вариант).

Создание простейшей СУБД с графическим интерфейсом

Цель работы: изучить средства стыковки приложения с графическим интерфейсом и базы данных. В работе используется приложение на базе библиотеки Qt, база данных на основе SQLite, команды языка SQL.

Задание

Разработать оконное приложение, управляющее простейшей базой данных. Структура базы данных задается в соответствии с заданием лабораторной работы № 1.

Приложение должно выполнять следующие функции:

1. создание новой базы данных или открытие созданной ранее,
2. добавление записи произвольного содержания в базу данных,
3. удаление записи с заданным номером из базы данных,
4. просмотр содержимого всей базы данных без сортировки,
5. вывод результатов двух запросов на выборку (фильтрация),
6. просмотр содержимого всей базы данных с сортировкой.

Приложение должно использовать библиотеку Qt. База данных создается при помощи СУБД SQLite. Стыковка приложения и базы данных осуществляется при помощи модуля QtSql.

Этапы выполнения работы

Создание каркаса приложения на основе GUI.

Редактирование формы.

Редактирование конструктора и деструктора класса QMainWindow.

Создание и реализация слотов.

Справочный материал

Средства Qt для взаимодействия приложения и базы данных.

Для соединения приложения, созданного на основе библиотеки Qt, с базой данных необходимы:

- драйвера базы данных для получения приложением данных на физическом уровне,
- классы для взаимодействия приложения и базы данных,
- классы для отображения результатов запросов.

Использование этих средств позволяет избежать циклов при просмотре таблиц базы данных. Чтобы приложение имело возможность использовать эти средства, в библиотеке Qt имеется модуль QtSql, подключаемый директивой #include <QtSql>. Также надо указать в файле проекта необходимость компоновки этого модуля, для этого в файле *.pro следует добавить строчку:

```
QT += sql
```

Выбор драйвера определяется используемой в приложении СУБД. Драйвер QMYSQL предназначен для взаимодействия приложения с популярной СУБД MySQL, QODBC — для взаимодействия с Microsoft SQL Server и другими базами данных, использующими открытый интерфейс доступа к базе данных ODBC (Open Database Connectivity), QSQLITE — для взаимодействия с SQLite версии 3. По умолчанию Qt работает с СУБД SQLite, поставляемой вместе с Qt в виде библиотеки.

Для взаимодействия приложения и базы данных используются следующие классы (не зависимо от СУБД):

QSqlDatabase — класс базы данных.

Методы:

addDatabase() - создание объекта для соединения с базой данных, принимает параметр — имя драйвера, соответствующее используемой СУБД.

setDatabaseName() - устанавливает имя базы данных, с которой будет работать приложение.

open() - открытие существующей базы данных или создание новой, если база данных с таким именем не обнаружена.

tables() - возвращает список таблиц базы данных.

QSqlTableModel — класс, предоставляющий в приложении редактируемую модель данных для одной таблицы.

Методы класса:

setTable() - выбор таблицы для запроса.

setFilter() - установление критерия отбора записей: setFilter(“year >= 2000”).

setEditStrategy() - определение стратегии редактирования и записи данных. При значении параметра метода QSqlTableModel::OnRowChange запись занных выполняется при переходе к другой строке таблицы, QSqlTableModel::OnFieldChange — запись данных выполняется уже при переходе в к другой ячейке таблицы.

select() - выполнение запроса.

Таким образом, код

```
QSqlTableModel model;
model.setTable(“tab1”);
model.setFilter(“year>=2000”);
model.select();
```

эквивалентен запросу на SQL:

```
SELECT * FROM tab1 WHERE year >= 2000;
```

QSqlQuery — класс, предназначенный для создания запроса к базе данных. Объект-запрос создается, когда открывается база данных, и при создании запроса надо указать, к какой базе данных запрос относится.

Методы:

clear() - очистка объекта-запроса, выполняется перед исполнением каждого нового запроса.

exec() - исполнение запроса; запрос, сформулированный на SQL, передается как строковый параметр.

За отображение данных отвечает класс QTableView — это визуальный компонент в Qt Creator. Классы QSqlTableModel и QTableView образуют структуру программы, называемую «модель-представление». Эта конструкция программного обеспечения позволяет отделить структуру данных и их хранилище от их визуального представления. Метод setModel() класса QTableView позволяет подключить данные из модели таблицы к представлению.

Программирование взаимодействия приложения и базы данных требует выполнения

следующих действий.

- Создание объекта базы данных и его подключение к базе данных (для примера, использована база данных SQLite):

```
QSqlDatabase db = QSqlDatabase::addDatabase("QSQLITE");
```

- Создание объекта для выдачи запросов и соединение его с базой данных:

```
QSqlQuery* query = new QSqlQuery(m_db);
```

- Отсылка запроса:

```
query.exec("INSERT INTO Person (number, name, salary) VALUES (1,'Ann', 25)")
```

- При повторном использовании объекта query его следует очистить:

```
query->clear();
```

- Создание объекта – модели таблицы для хранения результата запроса, полученного из базы m_db:

```
QSqlTableModel* model;
model = new QSqlTableModel(this,m_db);
```

- Связывание модели и конкретной таблицы в базе данных:

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

- Получение результата запроса:

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

- Фильтрация в запросе:

```
model->setFilter("Year>2000");
```

- Вывод результатов запроса из модели на экран в таблицу:

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

где ui – идентификатор формы, tableView — таблица в форме.

Создание простейшего приложения с базой данных

Данный пример разработан в инструментальной среде разработки (IDE) Qt Creator. В примере создается приложение, взаимодействующее с базой данных, в которой находится таблица с именем Person, хранящая данные об идентификаторе записи ID, имени человека Name и годе приема на работу Year. Поля ID и Year сделаем целочисленными, Name — строкой.

Приложение подключается в базе данных в начале работы программы, а если файл базы данных отсутствует, то приложение создаст новую базу данных. Также приложение будет выполнять добавление записей, удаление записей, просмотр базы и фильтрацию данных.

Для решения задачи воспользуемся СУБД SQLite, т.к. она устанавливается вместе с Qt.

Рассмотрим этапы создания приложения.

1. Создадим новое приложение на основе Qt Widget - GUI приложение. Такое приложение состоит из главного модуля, являющегося точкой входа в программу, модуля класса окна MainWindow (файлы mainwindow.h и mainwindow.cpp) и файла с описанием формы mainwindow.ui. Файл с расширением pro содержит описание компонентов проекта. Для того, чтобы приложение могло использовать модуль QtSql, добавим строчку QT += sql. Текст отредактированного pro-файла для проекта с

именем mytaml (mytaml.pro) приведен ниже.

```
#-------------------------------------------------
#
# Project created by QtCreator 2014-08-28T22:23:39
#
#-------------------------------------------------

QT       += core gui sql

TARGET = mytamle
TEMPLATE = app

SOURCES += main.cpp\
        mainwindow.cpp

HEADERS  += mainwindow.h

FORMS    += mainwindow.ui
#-------------------------------------------------
```

2. Спроектируем интерфейс пользователя. Примерный вариант главного окна приложения в соответствии с задачей представлен на Рис. 1. Его образуют элементы:

- таблица,
- пять кнопок для выполнения добавления, удаления записей, двух вариантов фильтрации и вывода всего содержимого базы данных,
- три текстовых поля для ввода новых данных.

Для редактирования окна откроем форму (щелчок мыши на файле mainwindow.ui) в редакторе форм, найдем на панели визуальных компонентов кнопки QPushButton, однострочные редакторы текста QLineEdit, метки (статические тексты) QLabel, таблицу QTableView и разместим необходимое количество этих компонентов на форме. При этом в программе объекты получат имена:
таблица — tableView (оставим имя по умолчанию),
метки - оставим имена по умолчанию,
окно для ввода в поле ID - lineEditID,
окно для ввода в поле Name - lineEditName,
окно для ввода в поле Year - lineEditYear,
кнопка для ввода данных в таблицу — pushButtonAdd (надпись на кнопке «Добавить запись»),
кнопка для удаления записи - pushButtonRem (надпись на кнопке «Удалить запись»),
кнопка для вывода данных из таблицы — allDataViewButton (надпись на кнопке «Вывести все»),
кнопка для вывода записей, удовлетворяющих условию Year>2000, — filterDataButton (надпись на кнопке «Год>2000»),

кнопка для вывода записей, удовлетворяющих условию Year<2001, — filterData2Button (надпись на кнопке «Год <2001»).

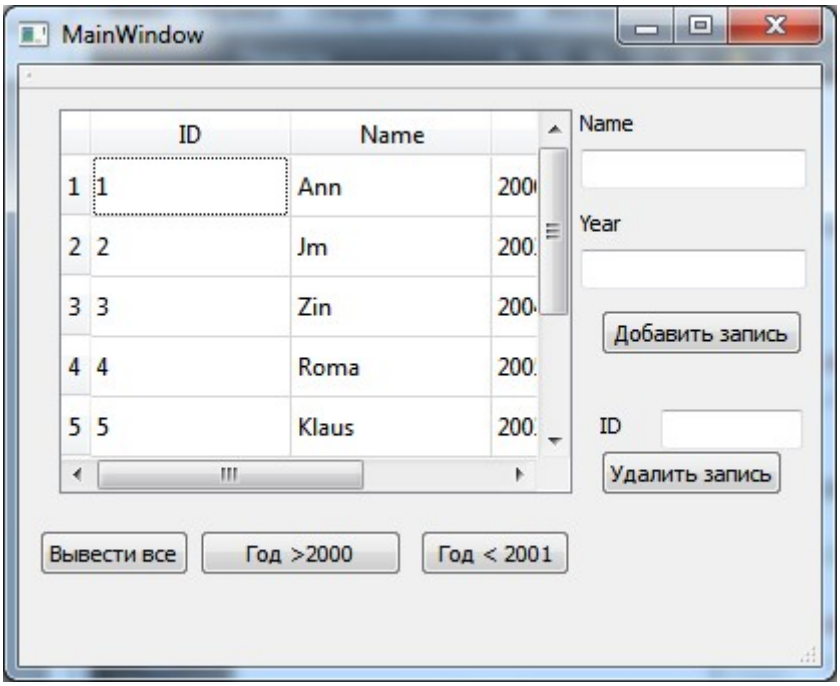

Рис. 1. Интерфейс пользователя приложения, работающего с базой данных

Слоты для кнопок создадим и отредактируем позже.

3. Отредактируем файл mainwindow.h для работы с базой данных:

```
#ifndef MAINWINDOW_H
#define MAINWINDOW_H

#include <QMainWindow>
#include <QtSql>

namespace Ui {
    class MainWindow;
}

class MainWindow : public QMainWindow
{
    Q_OBJECT

public:
    explicit MainWindow(QWidget *parent = 0);
    ~MainWindow();
```

```
private:
    Ui::MainWindow *ui;
    QSqlDatabase m_db; // объект базы данных
    QSqlQuery* query; // указатель на запрос
    QSqlTableModel* model; // указатель на таблицу данных в приложении
};

#endif // MAINWINDOW_H
```

4. Отредактируем файл mainwindow.cpp:

```
#include "mainwindow.h"
#include "ui_mainwindow.h"
#include <QMessageBox>

MainWindow::MainWindow(QWidget *parent) :
    QMainWindow(parent),
    ui(new Ui::MainWindow)
{
    ui->setupUi(this);
    m_db = QSqlDatabase::addDatabase("QSQLITE"); //соединение объекта базы данных
                                                 // с СУБД
    m_db.setDatabaseName("myDB");  //определение имени базы данных
    query = new QSqlQuery(m_db); // создание объекта для запроса
    if(!m_db.open()) // проверка на ошибку при открытии или создании базы данных
        throw "can't open database";
    if(!m_db.tables().contains("Person")) // если в базе не существует таблица  Person,
    { //то создание таблицы  Person и заполнение данными
        query->clear(); // очистка запроса
       query->exec("CREATE TABLE Person(ID INTEGER PRIMARY KEY, Name
VARCHAR, Year INTEGER);");  // исполнение запроса на добавление записи
        query->clear();
        query->exec("INSERT INTO Person (ID,Name,Year) VALUES (1,'Ann', 2000);");
        query->clear();
        query->exec("INSERT INTO Person (ID,Name,Year) VALUES (2,'Jim', 2003);");
    }
    model = new QSqlTableModel(this,m_db); // создание
                                           // редактируемой модели базы данных
    model->setTable("Person"); // создание модели таблицы  Person

    model->select(); // заполнение модели данными
```

```
    model->setEditStrategy(QSqlTableModel::OnFieldChange); // выбор стратегии
                                        //  сохранения изменений в базе данных
                                        //– сохранение происходит при переходе к другому полю
    ui->tableView->setModel(model); // соединение модели
                                        // и ее табличного представления в форме
}

MainWindow::~MainWindow()
{
    delete ui;
    delete query;
    delete model;
}
```

5. Свяжем кнопки на форме со слотами при помощи специальной команды Qt Creator **Перейти к слоту...**: В результате получим следующие слоты:
    - слот on_allDataViewButton_clicked() - с кнопкой «Вывести все»,
    - слот on_filterDataButton_clicked() - с кнопкой, фильтрующей записи со значением года больше 2000,
    - слот on_filterData2Button_clicked() - с кнопкой, фильтрующей записи со значением года, меньшим 2001,
    - слот on_pushButtonAdd_clicked() - с кнопкой добавления записи,
    - слот on_pushButtonRem_clicked() - с кнопкой удаления записи.

Файл mainwindow.h примет следующий вид:

```
#ifndef MAINWINDOW_H
#define MAINWINDOW_H

#include <QMainWindow>
#include <QtSql>

namespace Ui {
    class MainWindow;
}

class MainWindow : public QMainWindow
{
    Q_OBJECT

public:
```

```
    explicit MainWindow(QWidget *parent = 0);
    ~MainWindow();

private slots:
    void on_allDataViewButton_clicked();
    void on_filterDataButton_clicked();
    void on_filterData2Button_clicked();
    void on_pushButtonAdd_clicked();
    void on_pushButtonRem_clicked();
private:
    Ui::MainWindow *ui;
    QSqlDatabase m_db;
    QSqlQuery* query;
    QSqlTableModel* model;
};

#endif // MAINWINDOW_H
```

6. Отредактируем функции-слоты в файле mainwindow.cpp - добавим действия, выполняемые при нажатии на соответствующую кнопку:

```
#include "mainwindow.h"
#include "ui_mainwindow.h"
#include <QMessageBox>

MainWindow::MainWindow(QWidget *parent) :
    QMainWindow(parent),
    ui(new Ui::MainWindow)
{
    ui->setupUi(this);
    m_db = QSqlDatabase::addDatabase("QSQLITE");
    m_db.setDatabaseName("myDB");
    query = new QSqlQuery(m_db);
    if(!m_db.open())
        throw "can't open database";
    if(!m_db.tables().contains("Person"))
    {
        query->clear();
        query->exec("CREATE TABLE Person(ID INTEGER PRIMARY KEY, Name VARCHAR,
Year INTEGER);");
        query->clear();
        query->exec("INSERT INTO Person (ID,Name,Year) VALUES (1,'Ann', 2000);");
```

```
        query->clear();
        query->exec("INSERT INTO Person (ID,Name,Year) VALUES (2,'Jim', 2003);");
    }
    model = new QSqlTableModel(this,m_db); создание редактируемой модели базы данных
    model->setTable("Person"); создание модели таблицы  Person

    model->select(); заполнение модели данными
      model->setEditStrategy(QSqlTableModel::OnFieldChange); выбор стратегии сохранения изменений в базе данных
      ui->tableView->setModel(model); соединение модели и ее табличного представления в форме
  }

  MainWindow::~MainWindow()
  {
    delete ui;
    delete query;
    delete model;
  }

  void MainWindow::on_allDataViewButton_clicked()
  {
    model->setFilter("");
    model->select();
    ui->tableView->setModel(model);
    QMessageBox::information(0,tr("Action"),tr("All data"));// Сообщение, не обязательно
  }

  void MainWindow::on_filterDataButton_clicked()
  {
    model->setFilter("Year>2000");
    model->select();
    ui->tableView->setModel(model);
    QMessageBox::information(0,"Action","Filter"); //Сообщение, не обязательно
  }

  void MainWindow::on_filterData2Button_clicked()
  {
    model->setFilter("Year<2001");
    model->select();
```

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

void MainWindow::on_pushButtonAdd_clicked()
{
                if(ui->lineEditName->text().isEmpty()||ui->lineEditYear->text().isEmpty()||ui->lineEditID->text().isEmpty())
        return;
    QString id = ui->lineEditID->text();
    QString name = ui->lineEditName->text();
    QString year = ui->lineEditYear->text();
      QString buf = tr("INSERT INTO Person (ID,Name,Year) VALUES (")+id+tr(",'")+name+tr("',")+year+tr(");");
    query->clear();
    query->exec(buf);
    model->select();
}

void MainWindow::on_pushButtonRem_clicked()
{
    if(ui->lineEditID->text().isEmpty())
      return;
    QString id = ui->lineEditID->text();
    query->clear();
    query->exec(tr("DELETE FROM Person WHERE ID=")+id);
    model->select();
}
```

7. Наше приложение готово! Выполните сборку и тестирование приложения. При исполнении приложения обратите внимание:
    - поле ID является ключевым, поэтому попытки ввести значение повторно база данных не примет.