

Методические указания к выполнению лабораторной работы  
"Создание оконного приложения для работы с базой данных"

по курсу Базы данных

Виноградова М.В.

Ковалева Н.А.

Маслеников К.Ю.

Силантьева Е.Ю.

МГТУ им.Н.Э.Баумана

кафедра СОИУ (ИУ5)

Москва, 2023 г.

## Содержание

Задание.....	3
Среда выполнения .....	3
Начало работы с Qt/Qt Creator .....	4
Создание нового проекта .....	4
Окна и панели в среде Qt Creator .....	6
Меню среды разработки.....	9
Структура оконного десктопного приложения .....	9
Взаимодействие компонентов оконного приложения .....	13
Создание приложения Hello world .....	14
Режим редактирования формы.....	14
Первое приложение .....	16
Ошибки и отладка программы .....	18
Добавление виджетов из программы.....	19
Создание приложения для работы с базой данных .....	22
Создание базы данных .....	22
Создание оконного приложения .....	22
Соединение с базой данных.....	24
Чтение данных из базы данных.....	27
Добавление записей базы данных .....	31
Удаление записей базы данных.....	32
Литература.....	34
Приложение 1. Исходный код проекта test1. ....	35

## Задание

Создать в среде Qt Creator на языке Qt C++ оконное приложение для работы с базой данных.

База данных создается в СУБД PostgreSQL и содержит две связанные таблицы (по индивидуальной теме студента).

Приложение должно выполнять действия:

1. Проверка установленных драйверов для работы сБД.
2. Установка соединения с БД.
3. Отображение записей из таблицы.
4. Добавление, удаление и редактирование записей таблицы.
5. Отображение результата выполнения запроса к двум таблицам. Запрос формируется динамически на основе параметров, заданных пользователем.
6. Выбор записи в основной таблице БД и отображение связанных с ней записей из дочерней таблицы с возможностью их добавления, удаления и редактирования.

Дополнительное задание:

1. Работа с базой данных через механизм Модель-Представление.
2. Добавление прокси-модели для сортировки и фильтрации записей.

## Среда выполнения

Лабораторная работы выполняется в среде Qt Creator (написание приложения на языке C++ с применением библиотек Qt) и в среде PgAdmin для создания базы данных PostgreSQL.

Виртуальная машина с установленным ПО (QT5, QT6, QTCreator и драйвера для работы с БД для QT5 и QT6): <https://disk.yandex.ru/d/UkO1nzrlFca1Ug> (пароль для ВМ и PostgreSQL: student; логин: 1).

Для запуска ВМ под Windows нужен VirtualBox:

<https://download.virtualbox.org/virtualbox/7.0.6/VirtualBox-7.0.6-155176-Win.exe>

Для установки ПО под Linux: PgAdmin можно взять с официального сайта.

QT5 ставится командами:

```
sudo apt install qt5-default  
sudo apt install qtbase5-examples qtdeclarative5-examples
```

Драйвер ставится командой:

```
sudo apt install libqt5sql5-psql (Для QT5)
```

QTCreator ставится командой:

```
sudo apt-get install qtcreator
```

## Начало работы с Qt/Qt Creator

**Qt** - кросс-платформенный инструментарий разработки ПО (фреймворк) на языке программирования C++. Есть также привязки к другим языкам программирования: Python - PyQt, Ruby - QtRuby, Java - Qt Jambi, PHP - PHP-Qt и другие.

Позволяет запускать написанное с его помощью ПО в большинстве современных операционных систем путём простой компиляции программы для каждой ОС без изменения исходного кода. Включает в себя все основные классы, которые могут потребоваться при разработке прикладного программного обеспечения, начиная от элементов графического интерфейса и заканчивая классами для работы с сетью, базами данных и XML. Qt является полностью объектно-ориентированным, легко расширяемым и поддерживающим технику компонентного программирования.

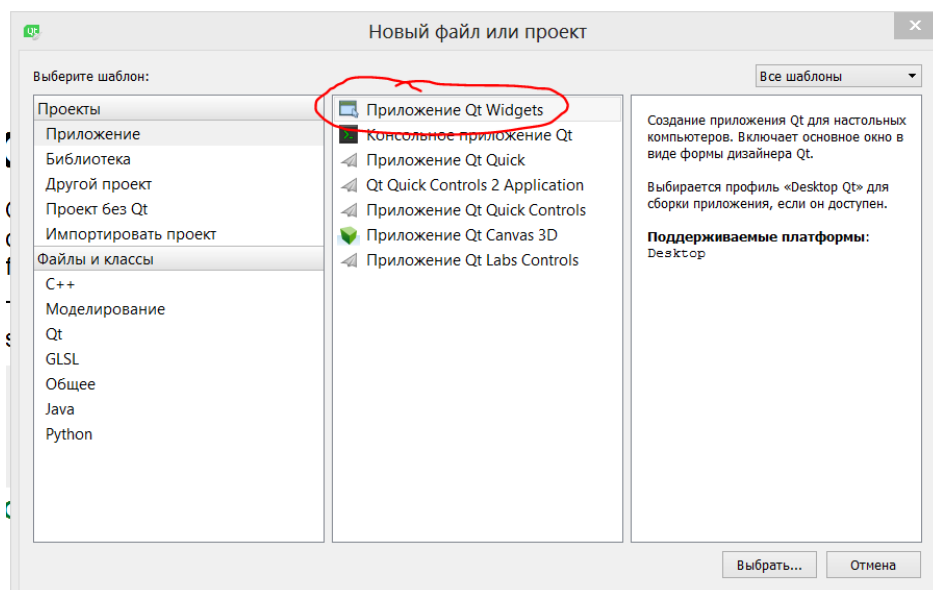
Существуют версии библиотеки для Microsoft Windows, систем класса UNIX с графической подсистемой X11, Mac OS X, Microsoft Windows CE, встраиваемых Linux-систем и платформы S60.

**Qt Creator** — свободная IDE для разработки для работы с фреймворком Qt. Включает в себя графический интерфейс отладчика и визуальные средства разработки интерфейса как с использованием QtWidgets, так и QML.

Документация по работе с Qt на русском языке - <http://doc.crossplatform.ru/qt/>

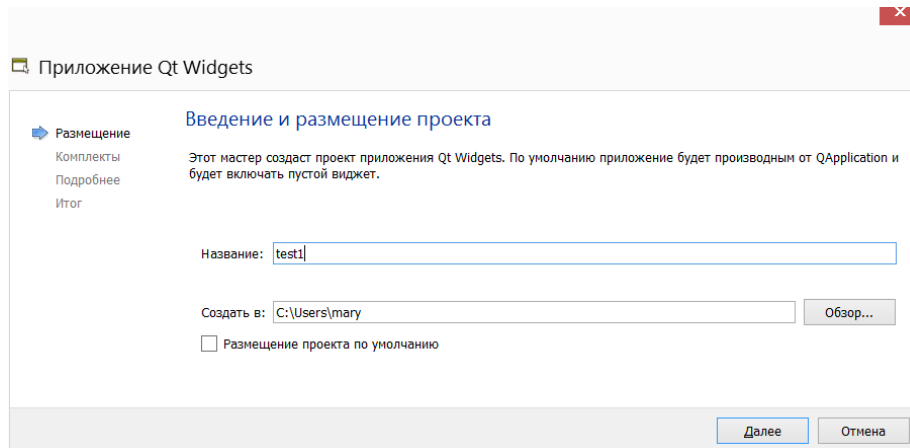
### Создание нового проекта

После запуска **Qt Creator** открывается окно с предложением создать или открыть проект. Выбираем в меню Файл - Создать файл или проект:



Выберите шаблон "Приложение Qt Widgets" (создание оконного десктопного приложения). Нажмите кнопку Выбрать.

Далее в окне укажите название проекта и местоположение:



Приложение Qt Widgets

**Введение и размещение проекта**

Этот мастер создаст проект приложения Qt Widgets. По умолчанию приложение будет производным от QApplication и будет включать пустой виджет.

Название:

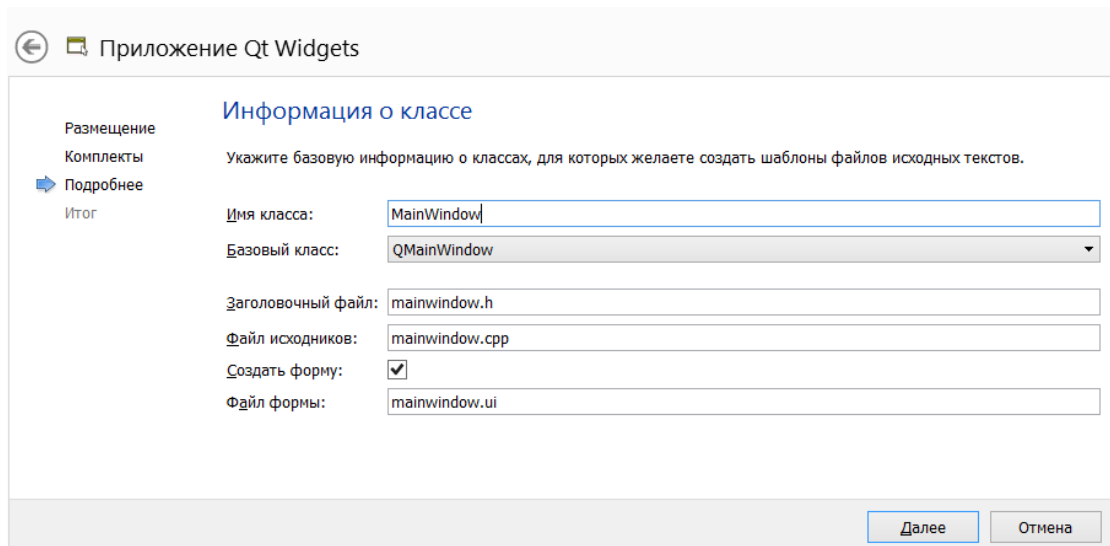
Создать в:  Обзор...

☐ Размещение проекта по умолчанию

Далее Отмена

Нажмите кнопку Далее.

Укажите название главной формы приложения (или оставьте по умолчанию):



Приложение Qt Widgets

**Информация о классе**

Укажите базовую информацию о классах, для которых желаете создать шаблоны файлов исходных текстов.

Имя класса:

Базовый класс:

Заголовочный файл:

Файл исходников:

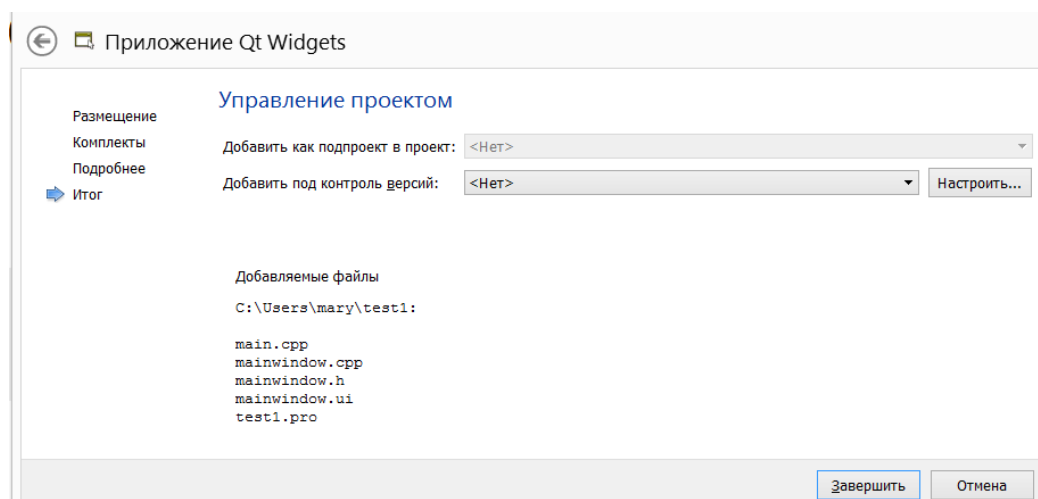
Создать форму: ☒

Файл формы:

Далее Отмена

Нажмите кнопку Далее.

Нажмите кнопку Завершить:



Приложение Qt Widgets

**Управление проектом**

Добавить как подпроект в проект:

Добавить под контроль версий:  Настроить...

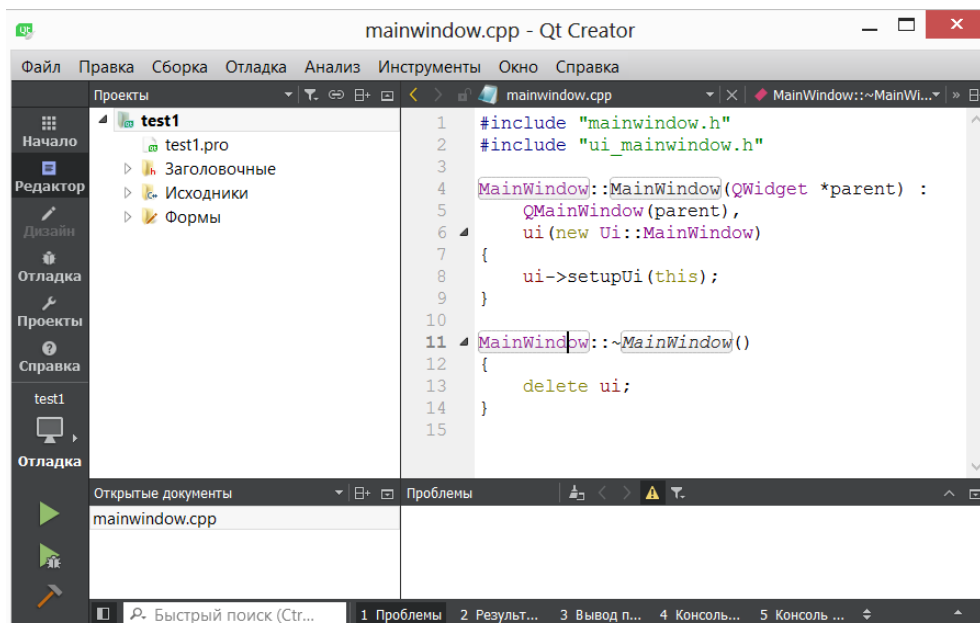
Добавляемые файлы

C:\Users\mary\test1:

main.cpp  
mainwindow.cpp  
mainwindow.h  
mainwindow.ui  
test1.pro

Завершить Отмена

Будет создан новый проект:



## Окна и панели в среде Qt Creator

Окно среды Qt Creator имеет следующий вид и содержит панели:

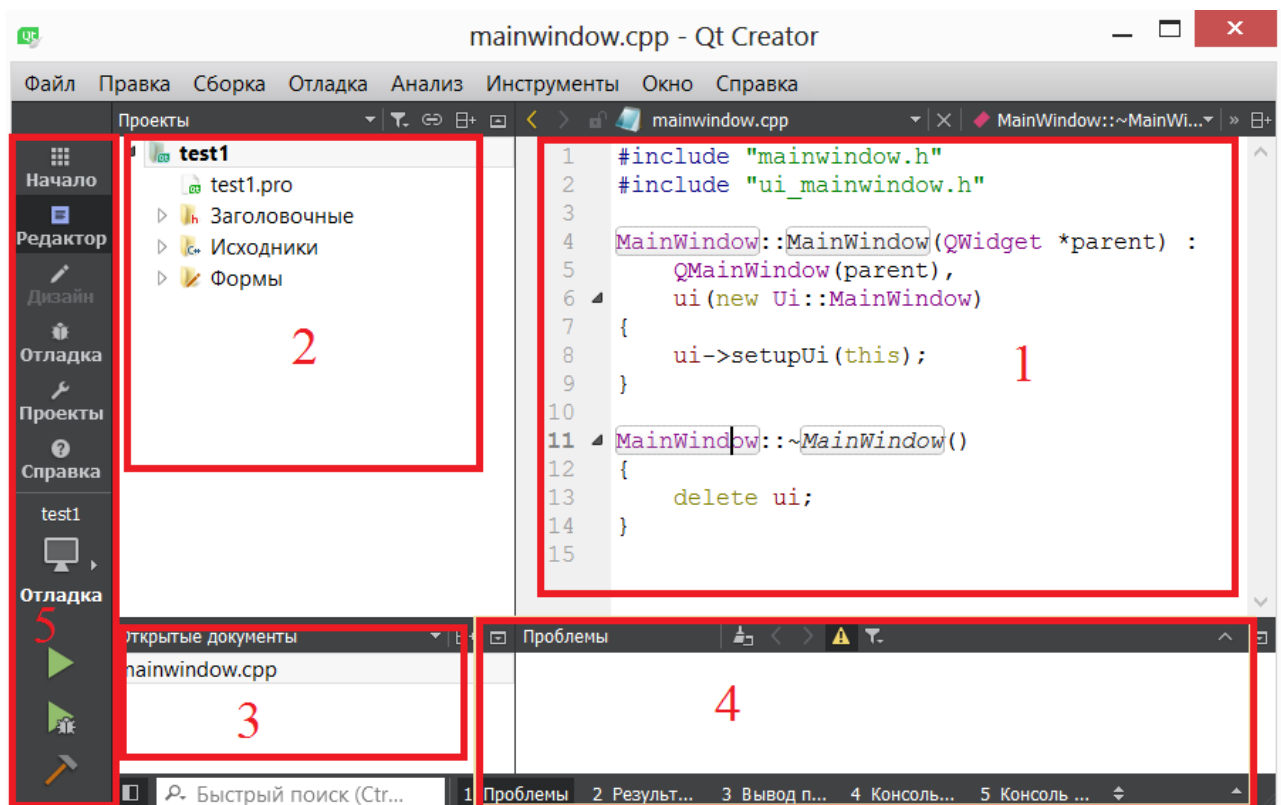


Рис.1. Панели среды разработки

(.cpp) и формы (.ui). При подключении дополнительных ресурсов они отображаются здесь же. При выборе любого файла проекта он будет открыт в текстовом редакторе (окно 1). Контекстное меню позволяет закрыть проект или сделать его активным (выделяется жирным):

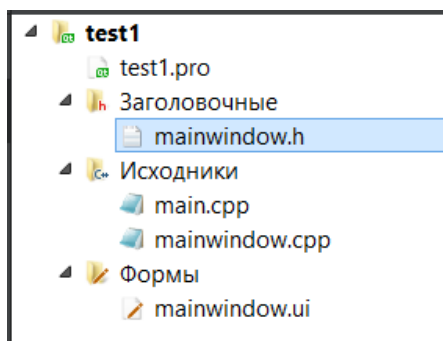


Рис.2. Обзорщик проектов.





**3** - Список открытых файлов. Позволяет переключаться между ними в текстовом редакторе, а также закрывать ненужные. Доступно контекстное меню.

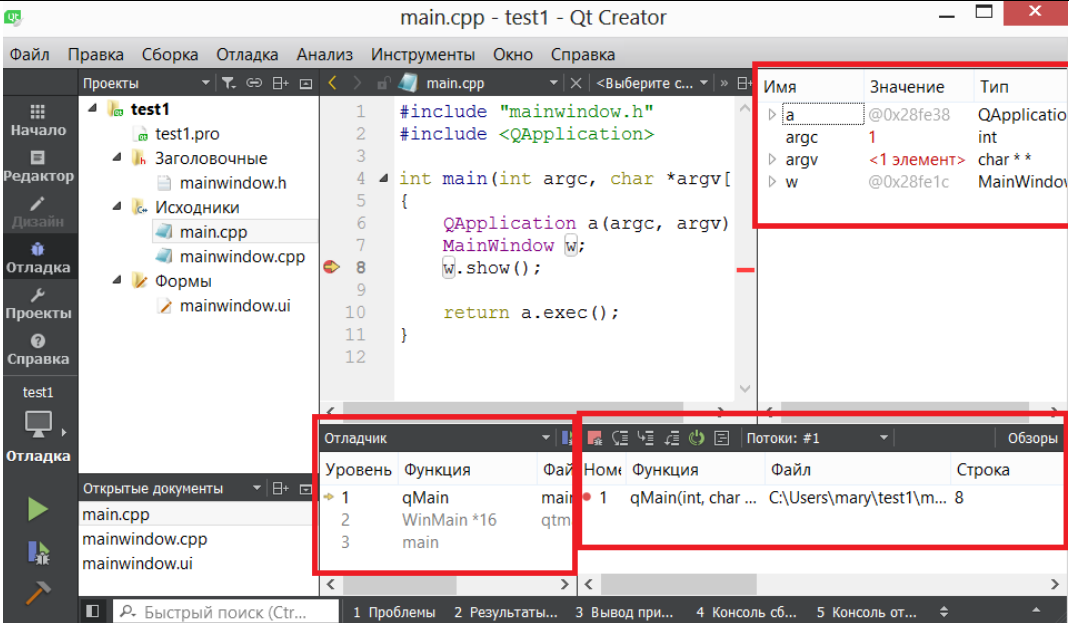

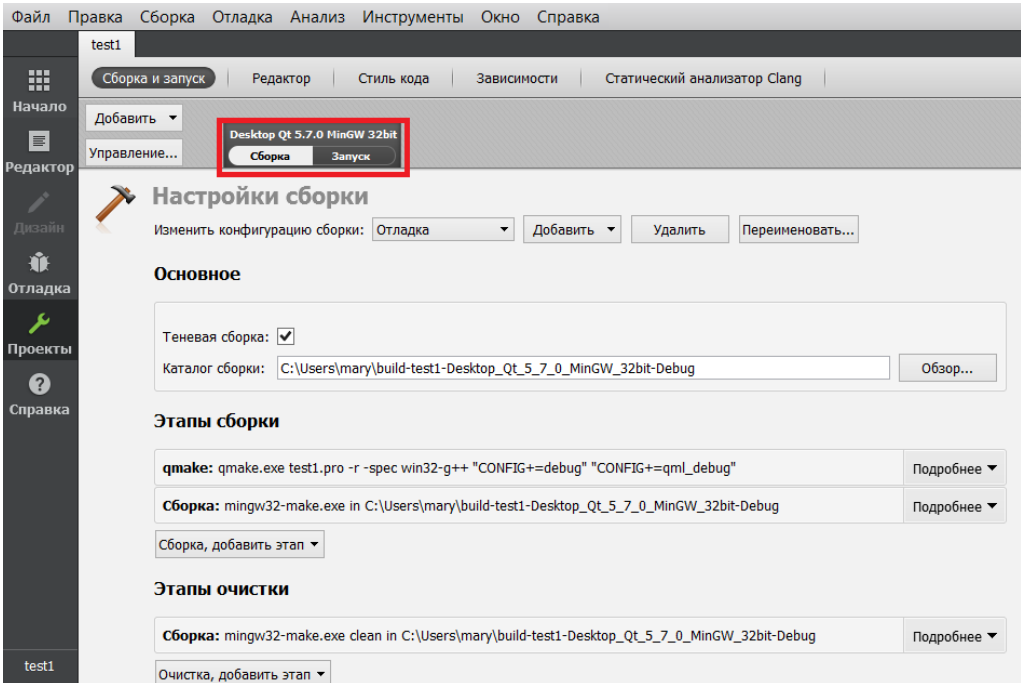





**4** - окно для отображения вспомогательных сервисов:

- Проблемы - отображает список ошибок, возникших при компиляции и запуске проекта.
- Результаты поиска - окно для поиска по файлам/проекту.
- Вывод приложения - вывод программы во время ее работы.
- Консоль сборки - журнал выполнения действий при компиляции и сборке проекта.
- Консоль отладчика - окно отладчика во время его работы.

**5** - Переключение между режимами среды (см. таблицу 1).

Таблица 1. Режимы работы в среде.

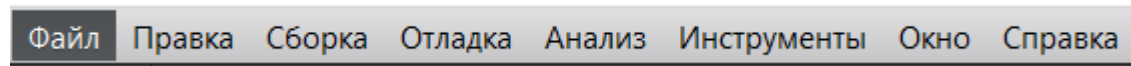
Иконка режима	Режим
 Начало	Стартовая страница среды разработки.
 Редактор	Режим редактирования исходных файлов. Его внешний вид рассмотрен выше.
 Дизайн	Режим графического редактора экранных форм проекта. Открывается, если выбрать формы (.ui) в обзорщике проекта.
 Отладка	Открывает окна для просмотра отладочной информации при отладке программы

	
 <b>Проекты</b>	<p>Окно для настройки параметров сборки и параметров запуска проекта:</p> 
 <b>Справка</b>	Вызов окна справки.
 <b>test1</b> <b>Отладка</b>	Выбор режима запуска программы (релиз или дебаг).
	Запуск программного проекта на выполнение в режиме релиза или дебага. Режим выбирается в пункте выше.
	Запуск программного проекта на выполнение в режиме отладки.
	Сборка проекта.



## Меню среды разработки

Меню среды Qt Creator содержит пункты:



**Файл** - набор пунктов для открытия, закрытия и сохранения файлов и проектов и выхода из среды.

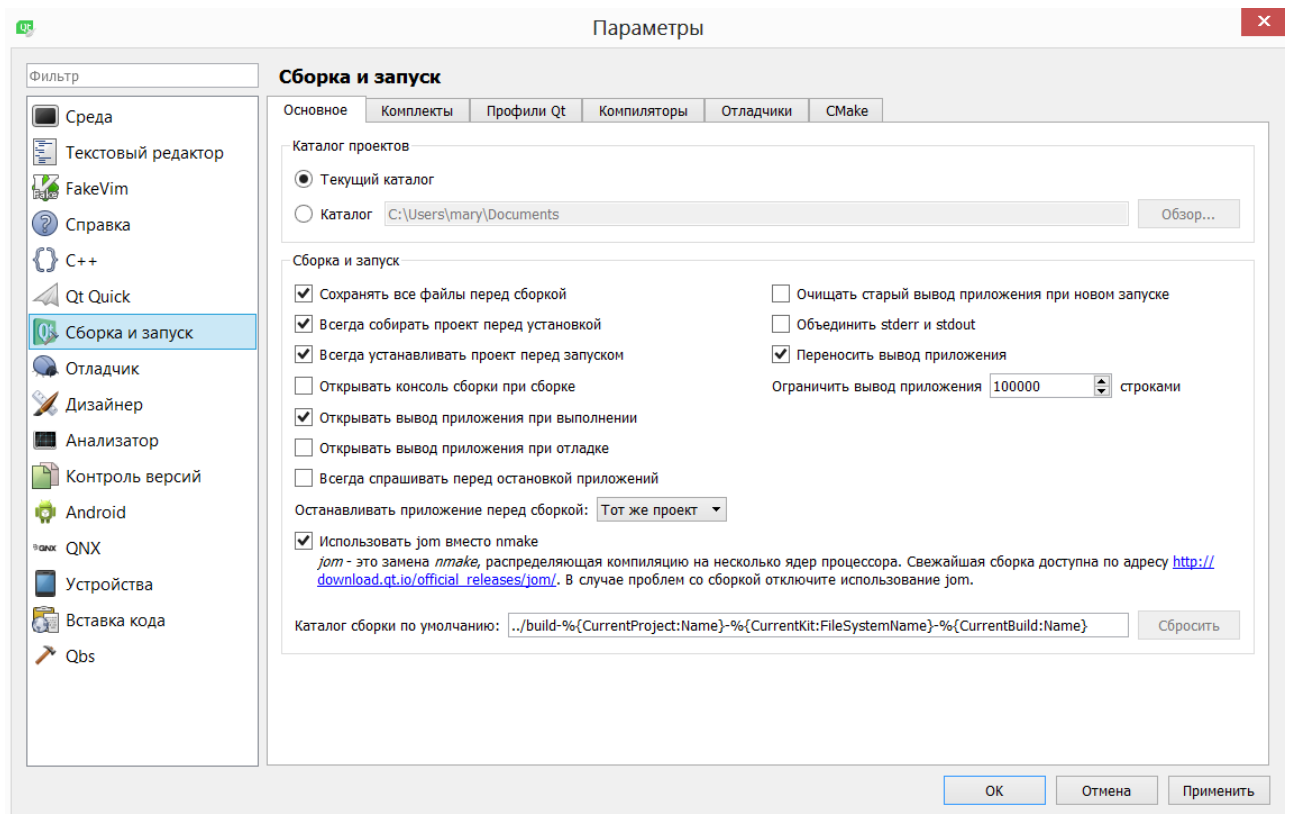
**Правка** - набор пунктов для копирования, вставки и поиска текста; отмены действий.

**Сборка** - набор пунктов для сборки и запуска программы.

**Отладка** - набор пунктов для управления отладчиком.

**Анализ** - набор пунктов для анализа использования ресурсов при работе программы.

**Инструменты** - вызов сервисов (в т.ч. внешних) для просмотра и редактирования проекта. Здесь же вызов окна параметров (пункт **Параметры**) для настройки среды, редактора, компилятора и т.д.

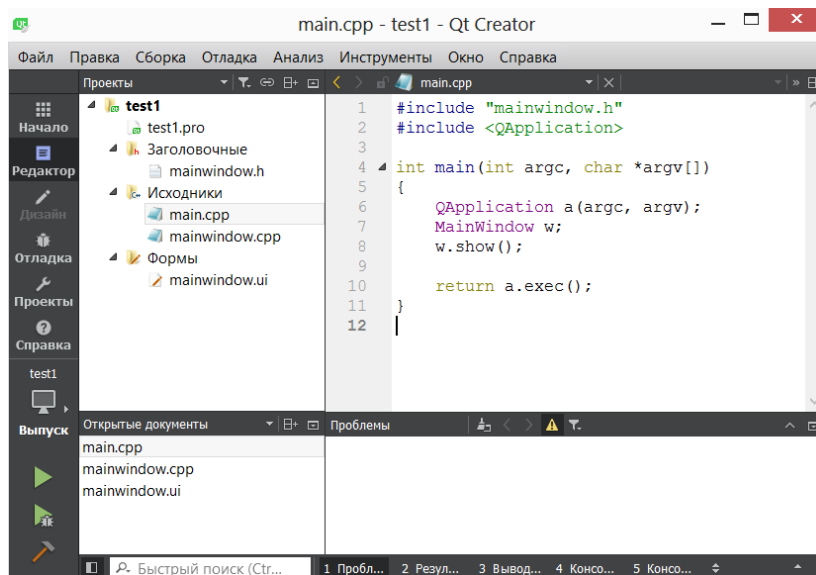


**Окно** - набор пунктов для управления отображением окон.

**Справка** - набор пунктов для вызова справки.

## Структура оконного десктопного приложения

После создания оконного десктопного приложения (под названием test1) будут созданы следующие файлы:



**test1.pro** - файл конфигурации проекта. Содержит директивы для подключения библиотек, исходных файлов и файлов ресурсов; указания компилятору и сборщику:

```
// подключили модули:
QT      += core gui

// версия и включение виджетов:
greaterThan(QT_MAJOR_VERSION, 4): QT += widgets

TARGET = test1
TEMPLATE = app

// подключение исходных файлов:

SOURCES += main.cpp\
           mainwindow.cpp

HEADERS  += mainwindow.h

FORMS    += mainwindow.ui
```

Изначально проект содержит одну главную форму. Прочие формы добавляются при разработке.

**mainwindow.h** - заголовочный файл главной формы. Содержит объявление ее класса:

```
#ifndef MAINWINDOW_H
#define MAINWINDOW_H

#include <QMainWindow>

namespace Ui {
class MainWindow;
}

class MainWindow : public QMainWindow
{
    Q_OBJECT
```

```

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

private:
    Ui::MainWindow *ui;
};

#endif // MAINWINDOW_H

```

Главная форма наследуется от класса **QMainWindow**, это класс для создания главного окна приложения с кнопками его закрытия и минимизации. Содержит ссылку на **Ui::MainWindow \*ui** для обращения к элементам формы.

**mainwindow.cpp** - исходный файл главной формы. Содержит описание ее методов, изначально это конструктор и деструктор:

```

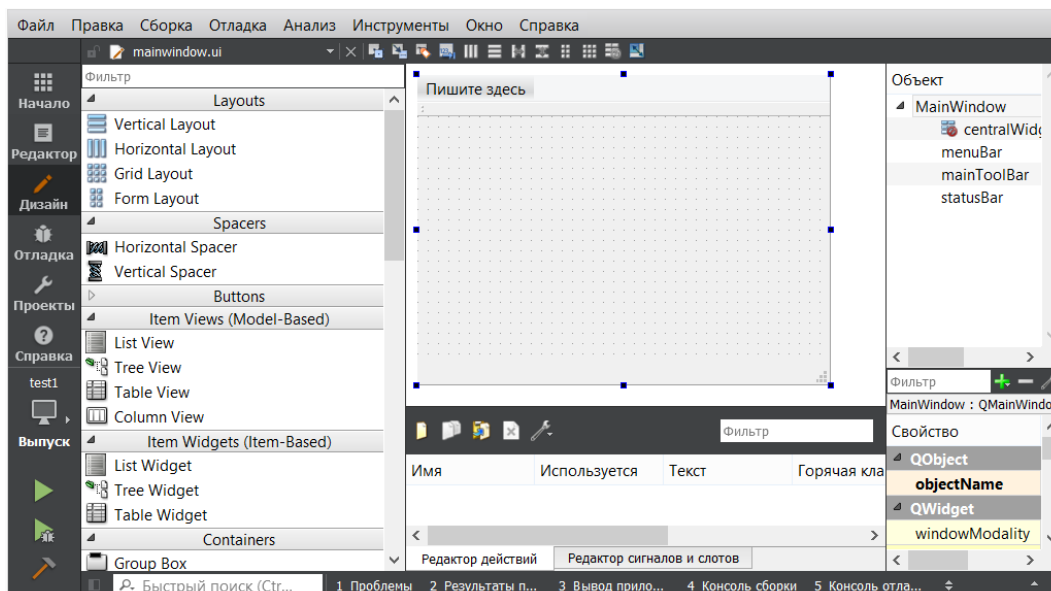
#include "mainwindow.h"
#include "ui_mainwindow.h"

MainWindow::MainWindow(QWidget *parent) :
    QMainWindow(parent),
    ui(new Ui::MainWindow)
{
    ui->setupUi(this);
}

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

```

**mainwindow.ui** - описание элементов главной формы. Содержит настройки форматирования и описания дочерних виджетов. В режиме Дизайн отображается графический редактор:



В режиме Редактор отображается описание формы в формате XML:

```
<?xml version="1.0" encoding="UTF-8"?>
<ui version="4.0">
  <class>MainWindow</class>
  <widget class="QMainWindow" name="MainWindow">
    <property name="geometry">
      <rect>
        <x>0</x>
        <y>0</y>
        <width>400</width>
        <height>300</height>
      </rect>
    </property>
    <property name="windowTitle">
      <string>MainWindow</string>
    </property>
    <widget class="QWidget" name="centralWidget"/>
    <widget class="QMenuBar" name="menuBar">
      <property name="geometry">
        <rect>
          <x>0</x>
          <y>0</y>
          <width>400</width>
          <height>26</height>
        </rect>
      </property>
    </widget>
    <widget class="QToolBar" name="mainToolBar">
      <attribute name="toolBarArea">
        <enum>TopToolBarArea</enum>
      </attribute>
      <attribute name="toolBarBreak">
        <bool>false</bool>
      </attribute>
    </widget>
    <widget class="QStatusBar" name="statusBar"/>
  </widget>
  <layoutdefault spacing="6" margin="11"/>
  <resources/>
  <connections/>
</ui>
```

**main.cpp** - исходный файл проекта с функцией **main**. Содержит программный код, запускающий программу:

```
#include "mainwindow.h"
#include <QApplication>

int main(int argc, char *argv[])
{
    QApplication a(argc, argv);
    MainWindow w;
    w.show();

    return a.exec();
}
```

Класс **QApplication** - это класс оконного приложения. Его функция **exec()** запускает цикл обработки событий. Внутри цикла происходит обмен сообщениями между активными объектами программы (они должны быть унаследованы от класса **QObject**). Приложение будет работать, пока этот цикл не завершится (при команде закрытия главного окна). Метод **exec()** возвращает код завершения программы.

Класс **MainWindow** - это класс главной формы, созданной в проекте. Метод **show()** отображает ее.

## ***Взаимодействие компонентов оконного приложения***

Оконное приложение состоит из главной формы, набора дочерних форм и виджетов, размещенных на них.

Виджет - это графический компонент, являющийся активным объектом и взаимодействующий с пользователем программы. Классы виджетов являются производными от класса **QWidget**, в котором определены основные методы для взаимодействия.

К виджетам относят формы (главная, диалоговая и модальная), кнопки, поля ввода, флажки, списки, таблицы, полосы прокрутки и другие элементы ввода и отображения данных.

Все виджеты (в том числе формы) обмениваются сообщениями по схеме сигнально-слотовых соединений. Класс, производный от класса **QObject**, может отправлять сигналы и принимать их. Перечень сигналов, которые может отправить объект класса, указан в определении класса с ключевым словом **signals**.

Испускание сигнала выполняется командой **emit <имя\_сигнала>**.

Для получения сигналов в классе необходимо определить соответствующие слоты. Они указываются в объявлении класса ключевым словом **slots**.

Для установки соответствия сигналов и слотов используют команду **connect()**. Она выполняет привязку сигналов одного объекта к слотам другого (или этого же) объекта. При

испускании указанного сигнала объектом-источником будет вызван метод-слот в объекте-приемнике.

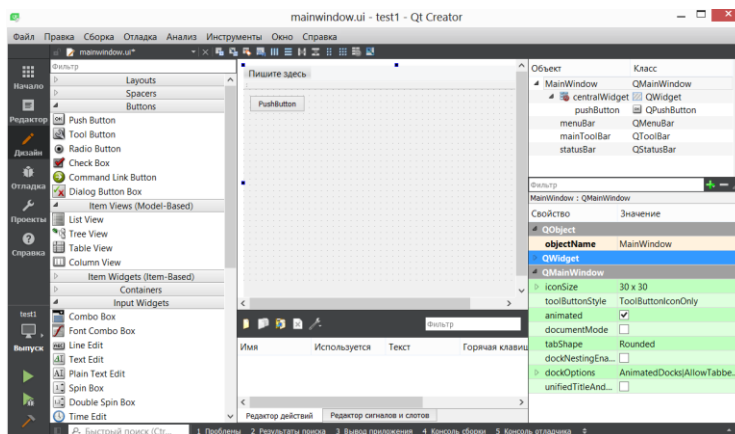
При испускании сигнала можно передавать значения, их указывают как параметры сигнальных и слотовых функций. Сигнатуры сигнальных и слотовых функций, соединяемых между собой, должны быть полностью идентичны.

Примеры использования сигнально-слотовых соединений можно посмотреть здесь:

<http://doc.crossplatform.ru/qt/4.3.2/tutorial.html>

## Создание приложения Hello world

После создания пустого оконного приложения (см. выше) перейдите в режим редактирования формы (двойной щелчок мышью по файлуmainwindow.ui в обозревателе объектов):



## Режим редактирования формы

В режиме редактирования формы доступны панели:

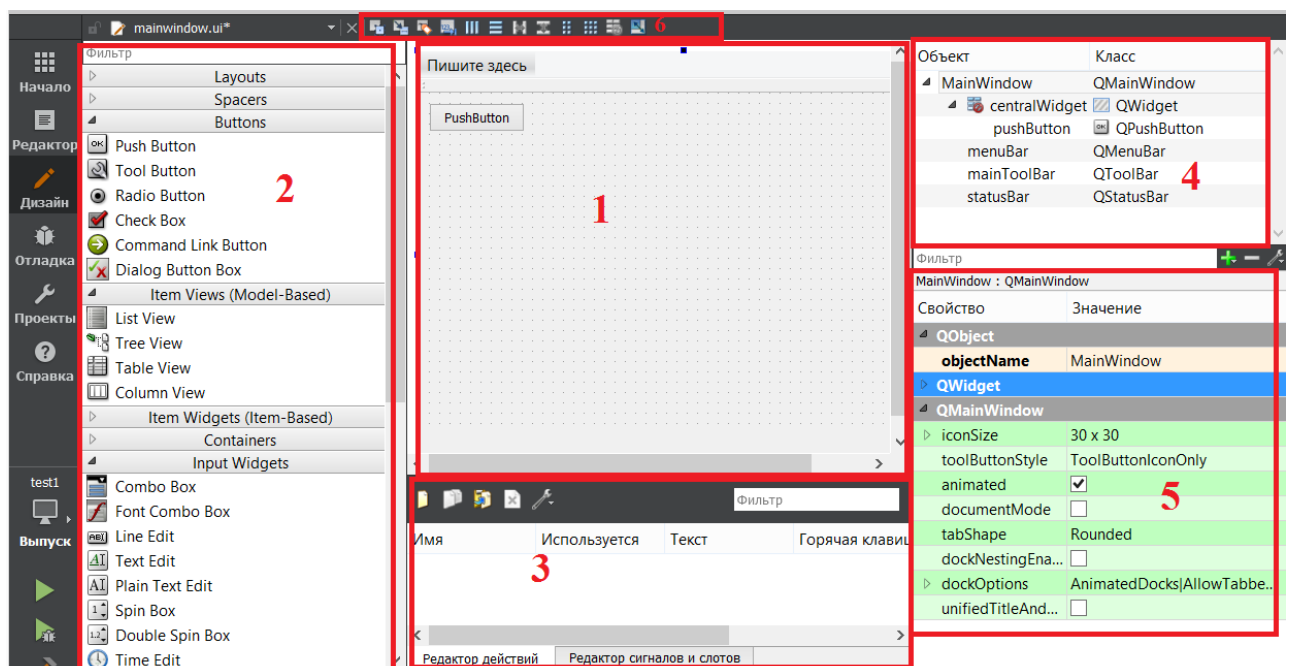


Рис.3. Панели в режиме редактирования формы.

**1** - область редактирования формы. Сюда переносят виджеты и элементы форматирования, формируют внешний вид экранной формы. Можно перемещать виджеты и изменять их размеры. Для лучшей компоновки применяют специальные элементы.

**2** - панель инструментов, содержащая доступные виджеты. Виджет захватывают левой кнопкой мыши и переносят на область редактирования формы (1). Виджеты разбиты на группы:

**Layouts** - компоновщики (выравнивают размещение элементов по вертикали, горизонтали или таблицей).

**Spacers** - "пружинки" для лучшего отображения.

**Buttons** - набор различных кнопок.

**Item Views** - отображение списков и таблиц через механизм модель-представление.

**Item Widgets** - виджеты для отображения списков и таблиц.

**Containers** - контейнеры виджетов. Реализуют области группировки, прокрутки, дочерние виджеты и вкладки.

**Input Widgets** - поля для ввода данных: текстовые, календарь, раскрывающиеся и обычные списки и т.д.

**Display Widgets** - виджеты для отображения данных: текст, линии, рисунки и т.д.

**3** - редактор действий и редактор сигнально-слотовых соединений для выбранного виджета (выделенного на области редактирования формы).

**4** - обозреватель виджетов формы. Содержит список дочерних виджетов формы с указанием их классов и иерархии вложенности. Доступно контекстное меню. Виджеты можно перемещать по иерархии вложенности.

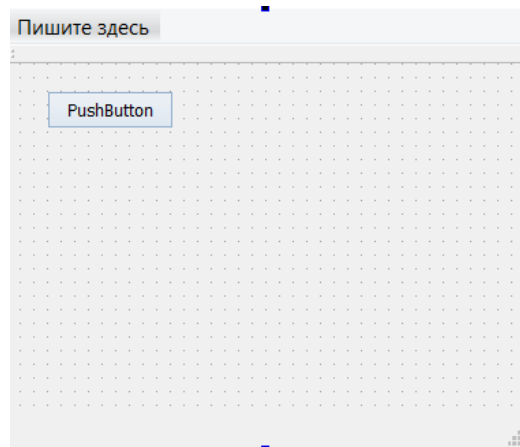
**5** - редактор свойств выбранного виджета (выделенного на области редактирования формы). Содержит атрибуты выбранного виджета и их значения. Значения можно изменять. Основной атрибут - **objectName** - содержит название объекта виджета, которое используется как имя переменной при обращении к виджету из кода программы. Атрибуты группируются в зависимости от того, к какому классу по иерархии наследования они относятся.

**6** - панель компоновки. Содержит элементы для выравнивания группы виджетов формы. Для применения надо выбрать на форме контейнер или компоновщик (или всю форму) и нажать на требуемый элемент выравнивания. Его эффект будет применен ко всем элементам выбранного контейнера.

Для выхода из режима редактирования формы нажмите клавишу Esc или кнопку Редактор на панели переключения между режимами.

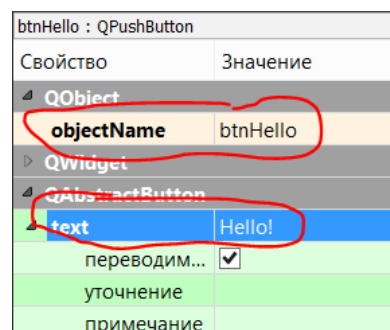
## Первое приложение

Перенесите кнопку (PushButton) из панели инструментов на область формы. Для этого нажмите левой кнопкой мыши на компонент PushButton на панели инструментов и, удерживая кнопку мыши нажатой, перенесите компонент PushButton на область редактирования формы, отпустите кнопку мыши. В результате компонент кнопки (PushButton) будет отображен на форме:



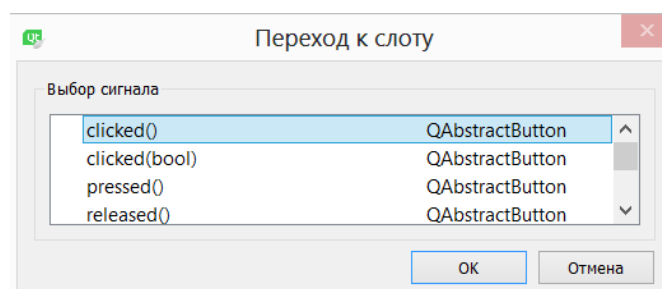
Выделите созданную кнопку (щелкните по ней левой кнопкой мыши) - она станет активным элементом. После этого в окне редактора свойств измените ее свойства:

```
objectName = btnHello    // название объекта кнопки в программе
text = Hello!             // отображаемая надпись на кнопке
```



Сохраните внесенные изменения: меню Файл - Сохранить все.

Теперь надо настроить обработчик нажатия на кнопку. Щелкните правой кнопкой мыши по кнопке (на области редактирования формы), откроется контекстное меню. Выберите пункт Перейти к слоту.... Откроется список обработчиков событий, выберите пункт clicked() и нажмите ОК:





Будет открыто окно редактирования для исходного класса формы mainwindow.cpp, где добавится функция обработки нажатия на кнопку:

```
void MainWindow::on_btnHello_clicked()
{
}
```

В теле метода напишите код для вывода сообщения:

```
void MainWindow::on_btnHello_clicked()
{
    QMessageBox::about(this, "info", "Hello!");
}
```

QMessageBox - класс для отображения сообщений. Основные методы: about() - вывод информации, warning() - вывод предупреждения с запросом, question() - запрос к пользователю.

Для просмотра всех методов и их описаний поставьте курсор на название класса и нажмите F1. Будет открыто окно справки с описанием класса и всех его функций.

Размер шрифта в окне редактора или справки можно изменить, нажав и удерживая Ctrl и прокручивая колесико мыши.

Описание также можно посмотреть здесь:


<http://doc.crossplatform.ru/qt/4.6.x/qmessagebox.html>

Перед использованием стандартных классов надо подключить заголовочные файлы с их объявлением. Для этого добавьте в начало файла mainwindow.cpp директиву:

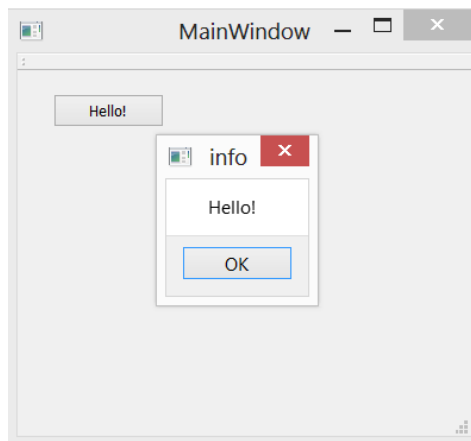
```
#include "QMessageBox"
```

Для подключения своих файлов указывайте расширения h. Для стандартных файлов расширение можно не указывать.

Сохраните внесенные изменения.

Запустите программу: меню Сборка - Запустить. Или кнопку  на панели переключения между режимами.

После компиляции и запуска программы будет открыто окно созданной кнопкой. Проверьте ее работу (нажмите на нее):



Для закрытия программы нажмите OK и потом кнопку закрытия (x) на главной форме.

## Ошибки и отладка программы

Если при компиляции или сборке программы возникли ошибки, то сообщения о них будут выведены в окне Проблемы (область 4 в нижней части на рис.1).

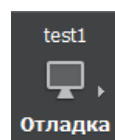
Двойной щелчок левой кнопкой мыши по тексту ошибки в списке выполнит открытие на редактирование исходного кода, содержащего ошибку. Он будет помечен:



Исправьте ошибку, сохраните изменения и повторно запустите программу.

Для отладки можно выполнить программу по шагам.

Сначала переведите программу в режим Debug (выбирается на панели переключения режимов):



Определите точки останова: меню Отладка - Поставить/Снять точку останова или щелчок левой кнопкой мыши по области слева от номера строки в режиме редактора исходного кода:





Далее запустите режим отладки: кнопка на панели переключения режимов (панель 5 на рис.1).

Программа выполнится до указанной строки.

Далее можно проходить программу по шагам: меню Отладка - Перейти через (выполнить следующую команду целиком) или Войти в (перейти к строкам вложенной функции). Или горячие клавиши F10 и F11.

Текущие значения переменных во время отладки можно посмотреть в окне Переменные и выражения (рис. в табл.1).

Для завершения отладки: меню Отладка - Прервать отладку.

Подробнее об отладке: <http://doc.crossplatform.ru/qt/4.6.x/debug.html>

Для вывода отладочной информации используйте класс **QDebug**.

Подключение:

```
#include <QDebug>
```

Вывод из программы (текст или значения переменных):

```
QDebug()<<"Error in ...";  
QDebug()<<"win title" << ui->btnHello->text();
```

Если отладочная информация не выводится, то проверьте, что нет других запущенных копий QtCreator или приложения. И пересоберите программу: меню Сборка - Пересобрать все.

## Добавление виджетов из программы

Виджеты можно добавлять в редакторе формы и в программе.

Откройте на редактирование заголовочный файл формы mainwindow.h.

Добавьте в начало подключение заголовочных файлов для классов кнопки и поля редактирования (в угловых скобках указывают стандартные классы):

```
#include <QPushButton>  
#include <QLineEdit>
```

В объявление класса главной формы добавьте указатели на кнопку и поле:

```
private:  
    Ui::MainWindow *ui;  
  
    QPushButton *btnHi;  
    QLineEdit *lnFio;
```

В конструктор формы добавьте команды их создания и размещения на форме, а в деструктор - удаления (для перехода к телу функции надо нажать и удерживать Ctrl и нажать левой кнопкой мыши на название функции):

```
MainWindow::MainWindow(QWidget *parent) :  
    QMainWindow(parent),
```

```

    ui(new Ui::MainWindow)
{
    ui->setupUi(this);

    // Создали объекты
    btnHi = new QPushButton("Hi");
    leFio = new QLineEdit();

    // Получить компоновщик главной формы (this - указатель на текущую форму)
    QLayout *layout = this->layout();

    // Если он пустой, то создать его и привязать к форме
    if( layout==0 )
    {
        layout = new QFormLayout();
        this->setLayout(layout);
    }

    // Создать новый виджет и компоновщик в виде таблицы
    QWidget *w = new QWidget(this);
    QGridLayout *formLayout = new QGridLayout();

    // Добавить на компоновщик метку (текст) и созданную ранее кнопку
    formLayout->addWidget(new QLabel(tr("Hello!")), 0, 0 );
    formLayout->addWidget(ui->btnHello, 0, 1 );

    // Добавить на компоновщик метку и поле ввода
    formLayout->addWidget(new QLabel(tr("Fio:")), 1, 0 );
    formLayout->addWidget(leFio, 1, 1 );

    // Добавить на компоновщик метку и новую кнопку
    formLayout->addWidget(new QLabel(tr("Hi!:")), 2, 0 );
    formLayout->addWidget(btnHi, 2, 1 );

    // Установить компоновщик для нового виджета
    w->setLayout(formLayout);
    w->setFixedSize(200, 300);
    formLayout->setSizeConstraint(QLayout::SetDefaultConstraint);

    // Поместить новый виджет в компоновщик главной формы
    layout->addWidget(w);
}

```

На форме или виджете могут размещаться дочерние виджеты. Для их компоновки используют компоновщики (вертикальные, горизонтальные и табличные - производные от класса QLayout). Компоновщики управляют размерами и отображением дочерних виджетов. При добавлении виджета в табличный компоновщик дополнительно указывабт номер строки и столбуа, начиная от 0.

Подробнее можно посмотреть:

<http://doc.crossplatform.ru/qt/4.6.x/layout.html>

Для обращения из программы к созданным компонентам используют имена переменных: ui->btnHello или leFio, btnHi.

Переменная ui - это указатель на объект виджета формы, она объявлена в объявлении класса формы:

```
Ui::MainWindow *ui;
```

Далее создайте функцию для обработки события нажатия на кнопку.

В объявление класса MainWindow добавьте описание слота:

```
private slots:  
  
void funHi(bool);
```

В конструкторе формы соедините событие от кнопки со слотом его обработки:

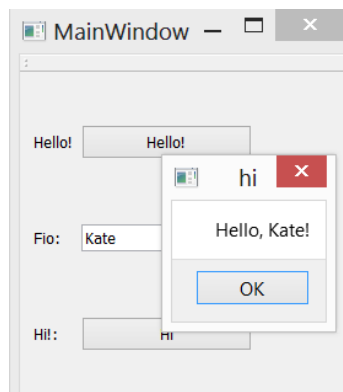
```
connect(btnHi, SIGNAL(clicked(bool)), this, SLOT(funHi(bool)));
```

Напишите реализацию метода (в файле mainwindow.cpp):

```
void MainWindow::funHi(bool flag)  
{  
    QString str = "Hello, " + leFio->text() + "!";  
    QMessageBox::about(this, "hi", str);  
}
```

Сохраните изменения и запустите программу.

Будет отображена форма с полем ввода и двумя кнопками. При нажатии на кнопку Hi будет выведено сообщением. В сообщении указано имя, введенное в поле:



Исходный код проекта в приложении 1.

# Создание приложения для работы с базой данных

## Создание базы данных

В среде PgAdmin установите соединение с сервером БД (login=student, password=1).

Создайте новую БД dbtest (с настройками по умолчанию).

Добавьте в нее две таблицы по теме своего варианта, например, Организацию и Персону:

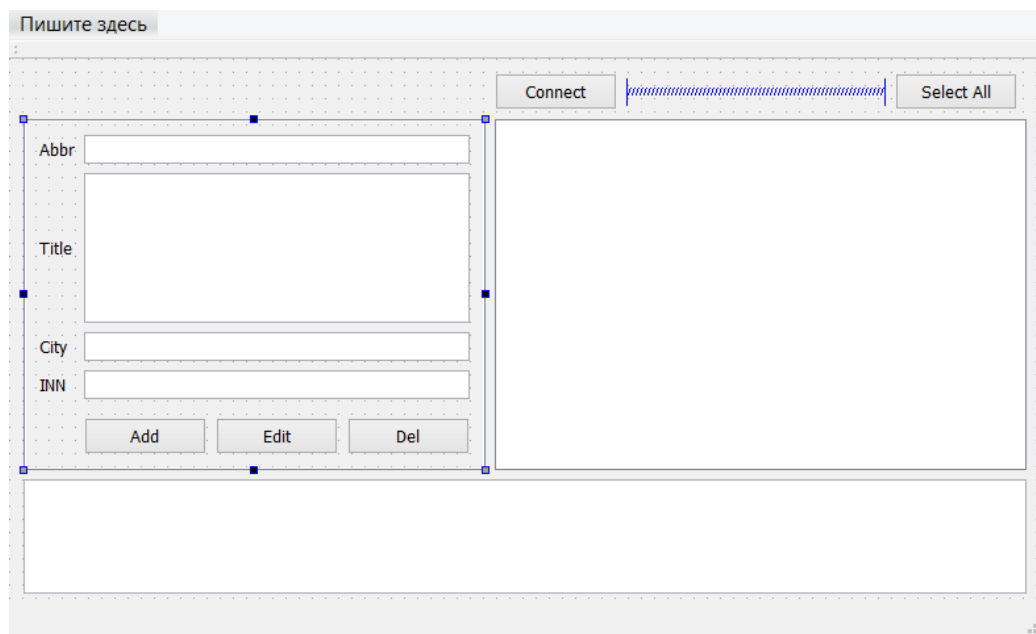
```
create table org
(
  abbr varchar(20) primary key,
  title varchar(200) unique,
  city varchar(50) not null default 'Moskva',
  inn numeric(20) unique
);
```

## Создание оконного приложения

Создайте новый проект оконного десктопного приложения test2.

Перейдите в редактор формы.

Добавьте на форму элементы и установите их свойства (в редакторе свойств):



- PushButton - (objectName=btnConnect, text=Connect)
- PushButton - (objectName=btnSelectAll, text=Select All)

- Frame - (frameShape=StyledPanel, frameShadow=Plain). На фрейм перенесите компоненты:

- Label - (objectName=lbAbbr, text=Abbr)
- Label - (objectName=lbTitle, text=Title)
- Label - (objectName=lbCity, text=City)
- Label - (objectName=lbInn, text=INN)
- LineEdit - (objectName=leAbbr)
- TextEdit - (objectName=teTitle)
- LineEdit - (objectName=leCity)
- LineEdit - (objectName=leInn)
- PushButton - (objectName=btnAdd, text=Add)
- PushButton - (objectName=btnEdit, text=Edit)
- PushButton - (objectName=btnDel, text=Del)
- между полями и кнопками поместите Vertical Spacer.

установите для фрейма компоновку по сетке (выделите фрейм и в панели компоновщиков выберите компоновку по сетке, в результате свойства будут изменены:

Layout	
layoutName	gridLayout

Также поменяйте политику при масштабировании формы - минимальный размер по горизонтали:

sizePolicy	[Minimum, Preferred,
Горизонтал...	Minimum
Вертикальн...	Preferred

- TableWidget - (objectName=twOrg)
- TextEdit - (objectName=teResult, readOnly=true/флаг ).

Для формы установите компоновку по сетке (выберите всю форму и на панели компоновки выберите "по сетке").

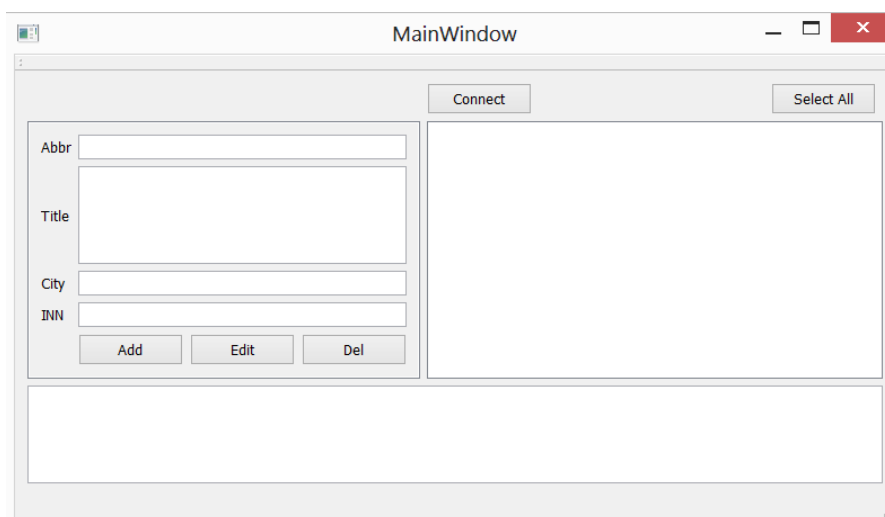
- между кнопками Connect и SelectAll поместите Horizontal Spacer.

Проверьте перечень и иерархию компонентов формы:

centralWidget	QWidget
btnConnect	QPushButton
btnSelectAll	QPushButton
frame	QFrame
btnAdd	QPushButton
btnDel	QPushButton
btnEdit	QPushButton
lbAbbr	QLabel
lbCity	QLabel
lbInn	QLabel
lbTitle	QLabel
leCity	QLineEdit
leInn	QLineEdit
leAbbr	QLineEdit
teTitle	QTextEdit
vert...acer	Spacer
horizo...Spacer	Spacer
teResult	QTextEdit
twOrg	QTableWidget
menuBar	QMenuBar

Сохраните изменения и запустите программу.

Проверьте ее отображаемый вид:



## Соединение с базой данных

Перейдите в режим редактора исходных файлов.

В файл **test2.pro** добавьте подключение библиотеки для работы с базой данных:

**QT** += core gui sql

Для работы с БД используют специальные классы Qt. Основные классы:

QSqlDatabase	Предоставляет соединение с базой данных.
QSqlError	Информация об ошибке базы данных SQL.
QSqlField	Управление полями в таблицах и представлениях базы данных SQL (доступ к полю в результате запроса).



QSqlQuery	Средства управления выражениями SQL и их выполнения (выполнение запроса к БД).
QSqlRecord	Заключает в себе запись базы данных (результат выполнения запроса).

Подробное описание классов и принципов работы с БД из Qt:

<http://doc.crossplatform.ru/qt/4.6.x/sql-programming.html>

Перейдите к файлу **mainwindow.h**.

Подключите нужные заголовочные файлы:

```
#include <QSqlDatabase>
```

В объявление класса формы MainWindow добавьте переменную для работы с БД и функцию-слот для установки соединения:

```
private:
    QSqlDatabase dbconn;

public slots:
    void dbconnect();
```

В классе **mainwindow.cpp** подключите нужные классы:

```
#include <QSqlError>
```

Добавьте в конструктор формы сигнально-слотовое соединение кнопки Connect со слотом ее обработки:

```
MainWindow::MainWindow(QWidget *parent) :
    QMainWindow(parent),
    ui(new Ui::MainWindow)
{
    ui->setupUi(this);

    connect(ui->btnConnect, SIGNAL(clicked(bool)), this, SLOT(dbconnect()));
}
```

Рассмотрим класс для установки соединения. Основные методы класса QSqlDatabase:

- Статический QSqlDatabase::**drivers()** - возвращает список строк с названиями доступных драйверов.
- Статический QSqlDatabase::**addDatabase**(название\_драйвера) - создает и возвращает соединение с БД, тип БД определяется названием драйвера.
- Функции устанавливают параметры соединения с БД:  
**setDatabaseName**(имя\_БД),      **setHostName**(IP-адрес),  
**setUserName**(имя\_пользователя), **setPassword**(пароль), **setPort**(порт).

- **bool open()** - открывает соединение в БД с учетом заданных параметров, возвращает признак успешности.
- **lastError()** - возвращает информацию о последней ошибке, возникшей в рамках данного соединения. Возвращает объект класса QSqlError.
- **bool transaction ()** - начинает транзакцию.
- **bool commit ()** - фиксирует транзакцию.
- **close()** - закрывает соединение.

Полное описание методов класса QSqlDatabase здесь:

<http://doc.crossplatform.ru/qt/4.6.x/qsqldatabase.html>

Напишите код функции для установки соединения с БД:

```
void MainWindow::dbconnect()
{
    if(!dbconn.isOpen())
    {
        // Если соединение не открыто, то вывести список доступных драйверов БД
        // (вывод в поле teResult, метод append добавляет строки).

        ui->teResult->append("SQL drivers:");
        ui->teResult->append(QSqlDatabase::drivers().join(","));

        // Создать глобальную переменную для установки соединения с БД

        dbconn=QSqlDatabase::addDatabase("QPSQL");

        // Установить параметры соединения: имя БД, адрес хоста, логин и пароль
        // пользователя, порт (если отличается от стандартного)

        dbconn.setDatabaseName("dbtest");
        dbconn.setHostName("localhost");
        dbconn.setUserName("student");
        dbconn.setPassword("1");

        // Открыть соединение и результат вывести в окно вывода

        if( dbconn.open() )
            ui->teResult->append("Connect is open...");
        else
        {
            ui->teResult->append("Error of connect:");
            ui->teResult->append(dbconn.lastError().text());
        }
    }
    else
        // Если соединение уже открыто, то сообщить об этом

        ui->teResult->append("Connect is already open...");
}
```

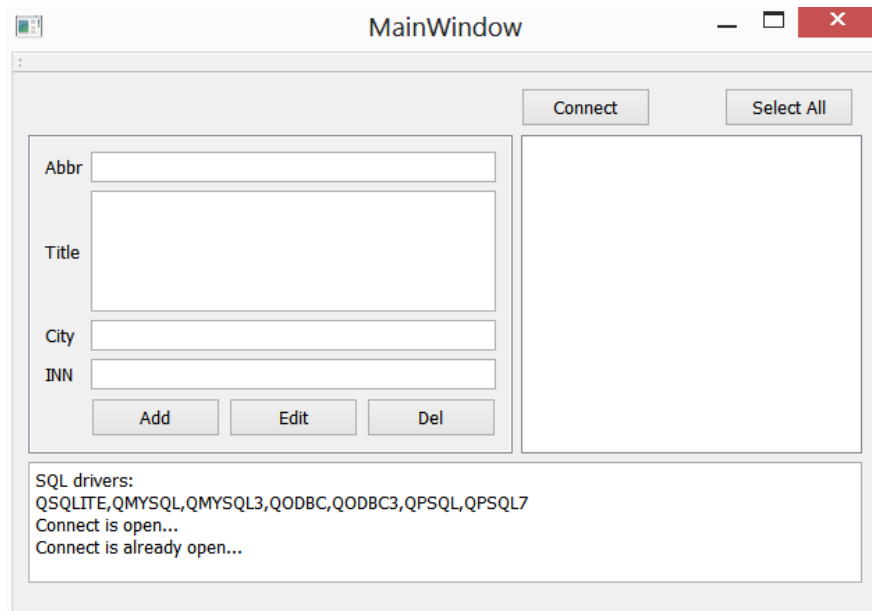
Функцию закрытия соединения добавьте в деструктор формы:

```
MainWindow::~MainWindow()
{
```

```
if( dbconn.isOpen())
    dbconn.close();

delete ui;
}
```

Сохраните все изменения и запустите программу. Проверьте обработку нажатия на кнопку Connect:



После нажатия на кнопку в нижнее окно должна быть выведена информация о подключении к БД.

### Чтение данных из базы данных

Добавьте в класс формы функцию-слот для обработки нажатия на кнопку SelectAll (объявление в классе и сигнально-слотовое соединение в конструкторе формы).

В коде функции напишите команды для чтения записей из таблицы БД и отображения их в таблице, используя класс QSqlQuery.

Класс QSqlQuery позволяет выполнять SQL запросы к БД и извлекать результат. В результате выполнения команды SELECT будет получено множество записей, которые сохраняются во временной табличной переменной (курсор). Чтение данных из курсора выполняется построчно. Указатель текущей записи перемещается по курсору и предоставляет доступ к одной записи. Возможно однонаправленное движение от начала к концу курсора или перемещения в разных направлениях, это определяется настройками курсора.

Основные функции класса QSqlQuery:

- bool **prepare**(строка\_запроса) - подготовка запроса. В подготовленный запрос передают параметры функцией **bindValue()**, затем запрос исполняют функцией **exec ()** без параметров.
- bool **exec**( строка\_запроса ) - подготовка и выполнение SQL-запроса из строки. Параметры должны содержаться в строке запроса.
- bool **next()** - извлекает из результата запроса очередную строку и помещает ее во внутреннюю переменную (текущая запись). Вызывается в цикле. Возвращает признак наличия очередной строки.
- QVariant **value** ( номер\_или\_название\_столбца ) - возвращает указанное поле (столбец) из текущей записи. Возвращает данные в формате QVariant, для преобразования в конкретный тип используются функции toString(), toInt(), toDate(), toDouble() и т.д.
- Кроме **next()** можно использовать функции **first ()**, **last ()**, **previous ()**, **seek ( )** для перемещения по курсору.
- int **size ()** - возвращает количество прочитанных строк или -1, если запрос не активен. Используется для select-запросов.
- int **numRowsAffected ()** - возвращает количество затронутых запросом строк. Используется для запросов на вставку, добавление и удаление записей.
- bool **isActive ()** - возвращает признак активности запроса. Также проверки типов запросов: **isSelect ()**, **isValid ()**.
- bool **isNull** ( номер\_поля ) - признак, что поле имеет значение NULL.
- QSqlError **lastError ()** - возвращает последнюю ошибку.
- QSqlRecord **record ()** - возвращает описание текущей строки курсора (мета-информация о наборе полей).

Полное описание методов класса QSqlDatabase и примеры работы с ним здесь:

<http://doc.crossplatform.ru/qt/4.6.x/qsqldata.html>

Код функции для чтения данных из таблицы и отображении их в компоненте QTableWidget (переменная **ui->twOrg**) :

```
void MainWindow::selectAll()
{
    // Очистить содержимое компонента

    ui->twOrg->clearContents();

    // Если соединение не открыто, то вызвать нашу функцию для открытия
    // если подключиться не удалось, то вывести сообщение об ошибке и
```

```

        // выйти из функции

if( !dbconn.isOpen() )
{
    dbconnect();
    if( !dbconn.isOpen() )
    {
        QMessageBox::critical(this,"Error",dbconn.lastError().text());
        return;
    }
}

// Создать объект запроса с привязкой к установленному соединению
QSqlQuery query(dbconn);

// Создать строку запроса на выборку данных
QString sqlstr = "select * from org";

// Выполнить запрос и поверить его успешность
if( !query.exec(sqlstr) )
{
    QMessageBox::critical(this,"Error", query.lastError().text());
    return;
}

// Если запрос активен (успешно завершен),
// то вывести сообщение о прочитанном количестве строк в окно вывода
// и установить количество строк для компонента таблицы

if( query.isActive() )
    ui->twOrg->setRowCount( query.size() );
else
    ui->twOrg->setRowCount( 0 );

ui->teResult->append( QString("Read %1 rows").arg(query.size()) );

// Прочитать в цикле все строки результата (курсора)
// и вывести их в компонент таблицы

int i=0;
while(query.next())
{
    ui->twOrg->setItem(i,0,new
QTableWidgetItem(query.value("abbr").toString()));
    ui->twOrg->setItem(i,1,new
QTableWidgetItem(query.value("title").toString()));
    ui->twOrg->setItem(i,2,new
QTableWidgetItem(query.value("city").toString()));
    ui->twOrg->setItem(i,3,new
QTableWidgetItem(query.value("inn").toString()));

    i++;
}
}

```

При выводе данных в компонент таблицы создаются и заполняются поля таблицы методом `setItem( номер_строки, номер_столбца, элемент )`. Элементы создаются как объекты класса `QTableWidgetItem( текст_поля )`.

Параметры для отображения компонента таблицы указывают один раз, например, в конструкторе формы:

```
// Количество столбцов
ui->twOrg->setColumnCount(4);

// Возможность прокрутки
ui->twOrg->setAutoScroll(true);

// Режим выделения ячеек - только одна строка
ui->twOrg->setSelectionMode(QAbstractItemView::SingleSelection);
ui->twOrg->setSelectionBehavior(QAbstractItemView::SelectRows);

// Заголовки таблицы
ui->twOrg->setHorizontalHeaderItem(0, new QTableWidgetItem("Abbr"));
ui->twOrg->setHorizontalHeaderItem(1, new QTableWidgetItem("Title"));
ui->twOrg->setHorizontalHeaderItem(2, new QTableWidgetItem("City"));
ui->twOrg->setHorizontalHeaderItem(3, new QTableWidgetItem("INN"));

// Последний столбец растягивается при изменении размера формы
ui->twOrg->horizontalHeader()->setStretchLastSection(true);

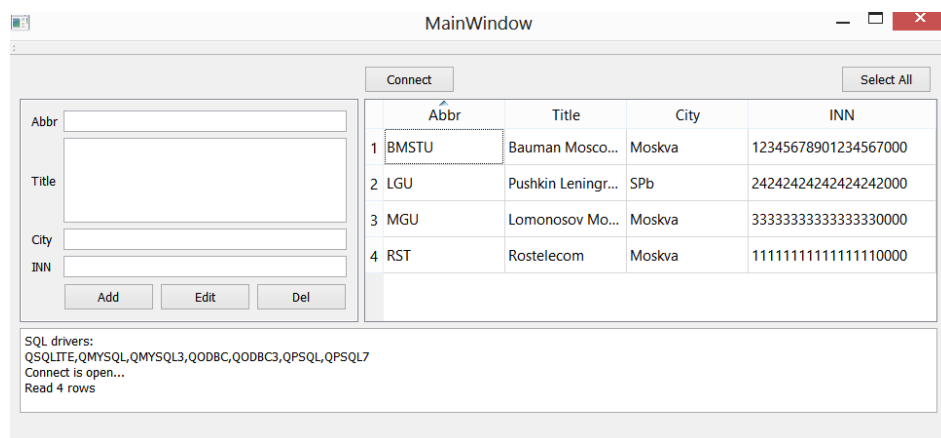
// Разрешаем сортировку пользователю
ui->twOrg->setSortingEnabled(true);
ui->twOrg->sortByColumn(0);

// Запрет на изменение ячеек таблицы при отображении
ui->twOrg->setEditTriggers(QAbstractItemView::NoEditTriggers);
```

Также эти параметры можно указать в редакторе Свойств таблицы в режиме редактирования формы.

Сохраните изменения и запустите программу.

Проверьте работу функции выборки записей:



## Добавление записей базы данных

Добавьте в класс формы функцию-слот для обработки нажатия на кнопку Add (объявление в классе и сигнально-слотовое соединение в конструкторе формы).

В коде функции напишите команды для добавления новой записи в таблицу БД. Значения полей записи считываются из полей редактирования формы:

```
void MainWindow::add()
{
    // Подключиться к БД

    if( !dbconn.isOpen() )
    {
        dbconnect();
        if( !dbconn.isOpen() )
        {
            QMessageBox::critical(this, "Error", dbconn.lastError().text());
            return;
        }
    }

    QSqlQuery query(dbconn);

    // Создать строку запроса

    QString sqlstr = "insert into org(abbr,title,city,inn) values(?,?,?,?)";

    // Подготовить запрос

    query.prepare(sqlstr);

    // Передать параметры из полей ввода в запрос

    query.bindValue(0,ui->leAbbr->text());
    query.bindValue(1,ui->teTitle->toPlainText());
    query.bindValue(2,ui->leCity->text());

    // Если тип поля отличается от строкового, то преобразовать его

    query.bindValue(3,ui->leInn->text().toLongLong());

    // Выполнить запрос

    if( !query.exec() )
    {
        ui->teResult->append( query.lastQuery());
        QMessageBox::critical(this, "Error", query.lastError().text());
        return;
    }

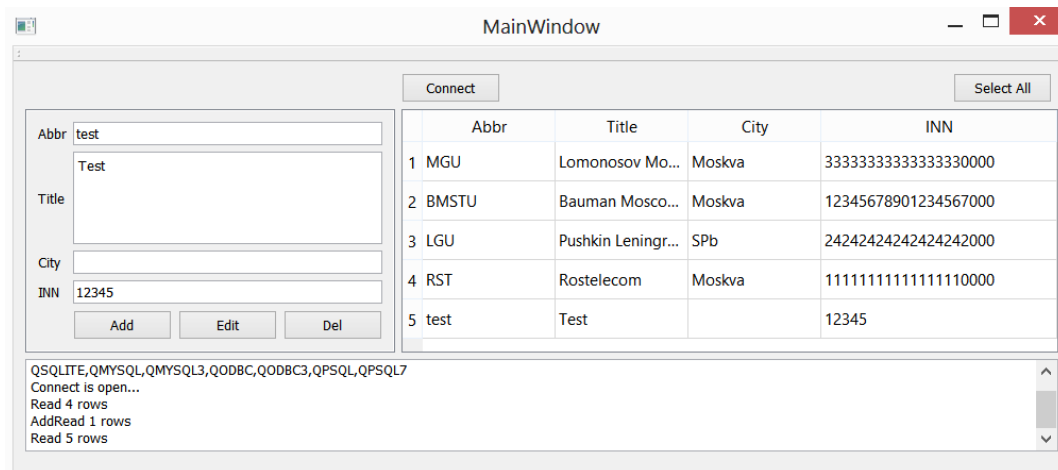
    // Если запрос выполнен, то вывести сообщение о добавлении строки

    ui->teResult->append( QString("AddRead %1
rows").arg(query.numRowsAffected()) );

    // и обновить записи в компоненте таблицы

    selectAll();
}
```

Сохраните изменения и запустите программу. Проверьте работу кнопки Add:



Обратите внимание, что поле city не получило значение по умолчанию, поскольку в запрос была передана пустая строка, которая не эквивалентна NULL значению.

### Удаление записей базы данных

Добавьте в класс формы функцию-слот для обработки нажатия на кнопку Del (объявление в классе и сигнально-слотовое соединение в конструкторе формы).

В коде функции напишите команды для удаления выбранной записи из таблицы БД. Запись выбирается пользователем в компоненте таблицы:

```
void MainWindow::del()
{
    // Подключение к БД

    if( !dbconn.isOpen() )
    {
        dbconnect();
        if( !dbconn.isOpen() )
        {
            QMessageBox::critical(this, "Error", dbconn.lastError().text());
            return;
        }
    }

    // Получить номер выбранной строки в компоненте таблицы

    int currow = ui->twOrg->currentRow();

    // Если он меньше 0 (строка не выбрана), то
    // сообщение об ошибке и выход из функции

    if( currow < 0 )
    {
        QMessageBox::critical(this, "Error", "Not selected row!");
        return;
    }

    // Спросить у пользователя подтверждение удаления записи
    // Используется статический метод QMessageBox::question
    // для задания вопроса, который возвращает код нажатой кнопки
```



```

if( QMessageBox::question(this, "Delete", "Delete row?",
    QMessageBox::Cancel, QMessageBox::Ok) == QMessageBox::Cancel)
    return;

    // Создать объект запроса
    QSqlQuery query(dbconn);

    // Создать строку запроса.
    // Вместо подготовки запроса и передачи параметров значение параметра
    // конкатенируется со строкой запроса
    // Обратите, что строковое значение помещается в одинарные кавычки
    // Значение выбирается из компонента таблицы методом item(row,col)

    QString sqlstr = "delete from org where abbr = '"
        + ui->twOrg->item(currow,0)->text() + "'";

    // Выполнить строку запроса и проверить его успешность

    if( !query.exec(sqlstr) )
    {
        ui->teResult->append( query.lastQuery());
        QMessageBox::critical(this, "Error", query.lastError().text());
        return;
    }

    // Вывести сообщение об удалении строки
    ui->teResult->append( QString("Del %1 rows").arg(query.numRowsAffected()) );

    // Обновить содержимое компонента таблицы
    selectAll();
}

```

Сохраните изменения и запустите программу. Проверьте работу кнопки Del.

## Литература

1. Шлее М. Qt 5.10. Профессиональное программирование на C++. – БХВ-Петербург, 2018.
2. Документация по Qt с примерами на русском языке <http://doc.crossplatform.ru/qt/>

## Приложение 1. Исходный код проекта test1.

```
#-----
#
# test1.pro
#
#-----

QT      += core gui

greaterThan(QT_MAJOR_VERSION, 4): QT += widgets

TARGET = test1
TEMPLATE = app

SOURCES += main.cpp\
           mainwindow.cpp

HEADERS  += mainwindow.h

FORMS    += mainwindow.ui

#-----
#
# mainwindow.h
#
#-----

#ifndef MAINWINDOW_H
#define MAINWINDOW_H

#include <QMainWindow>
#include <QPushButton>
#include <QLineEdit>

namespace Ui {
class MainWindow;
}

class MainWindow : public QMainWindow
{
    Q_OBJECT

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

private slots:
    void on_btnHello_clicked();

    void funHi(bool);

private:
    Ui::MainWindow *ui;

    QPushButton *btnHi;
    QLineEdit *leFio;
};

#endif // MAINWINDOW_H
```

```

#-----
#
#mainwindow.cpp
#
#-----

#include "mainwindow.h"
#include "ui_mainwindow.h"
#include "QMessageBox"
#include <QFormLayout>
#include <QLabel>
#include <QDebug>

MainWindow::MainWindow(QWidget *parent) :
    QMainWindow(parent),
    ui(new Ui::MainWindow)
{
    ui->setupUi(this);

    btnHi = new QPushButton("Hi");
    leFio = new QLineEdit();

    QLayout *layout = this->layout();
    if( layout==0 )
    {
        layout = new QFormLayout();
        this->setLayout(layout);
    }
    QWidget *w = new QWidget(this);
    QGridLayout *formLayout = new QGridLayout();

    formLayout->addWidget(new QLabel(tr("Hello!")),0,0 );
    formLayout->addWidget(ui->btnHello,0,1 );
    formLayout->addWidget(new QLabel(tr("Fio:")),1,0 );
    formLayout->addWidget(leFio,1,1 );
    formLayout->addWidget(new QLabel(tr("Hi!:")),2,0 );
    formLayout->addWidget(btnHi,2,1 );

    w->setLayout(formLayout);
    w->setFixedSize(200,300);
    formLayout->setSizeConstraint(QLayout::SetDefaultConstraint);
    layout->addWidget(w);

    connect(btnHi,SIGNAL(clicked(bool)),this,SLOT(funHi(bool)));
}

MainWindow::~MainWindow()
{
    delete leFio;
    delete btnHi;
    delete ui;
}

void MainWindow::on_btnHello_clicked()
{
    QMessageBox::about(this,"info","Hello!");
}

void MainWindow::funHi(bool flag)
{
    QString str = "Hello, " + leFio->text() + "!";
    QMessageBox::about(this,"hi",str);
}

```

```

#-----
#
# main.cpp
#
#-----
#include "mainwindow.h"
#include <QApplication>

int main(int argc, char *argv[])
{
    QApplication a(argc, argv);
    MainWindow w;
    w.show();

    return a.exec();
}

#-----
#
# mainwindow.ui
#
#-----

<?xml version="1.0" encoding="UTF-8"?>
<ui version="4.0">
    <class>MainWindow</class>
    <widget class="QMainWindow" name="MainWindow">
        <property name="geometry">
            <rect>
                <x>0</x>
                <y>0</y>
                <width>381</width>
                <height>322</height>
            </rect>
        </property>
        <property name="windowTitle">
            <string>MainWindow</string>
        </property>
        <widget class="QWidget" name="centralWidget">
            <widget class="QPushButton" name="btnHello">
                <property name="geometry">
                    <rect>
                        <x>20</x>
                        <y>20</y>
                        <width>93</width>
                        <height>28</height>
                    </rect>
                </property>
                <property name="text">
                    <string>Hello!</string>
                </property>
            </widget>
        </widget>
        <widget class="QMenuBar" name="menuBar">
            <property name="geometry">
                <rect>
                    <x>0</x>
                    <y>0</y>
                    <width>381</width>
                    <height>26</height>
                </rect>
            </property>
        </widget>
        <widget class="QToolBar" name="mainToolBar">

```

```
<attribute name="toolBarArea">
  <enum>TopToolBarArea</enum>
</attribute>
<attribute name="toolBarBreak">
  <bool>false</bool>
</attribute>
</widget>
<widget class="QStatusBar" name="statusBar"/>
</widget>
<layoutdefault spacing="6" margin="11"/>
<tabstops>
  <tabstop>btnHello</tabstop>
</tabstops>
<resources/>
<connections/>
</ui>
```