

МИНИСТЕРСТВО НАУКИ И ВЫСШЕГО ОБРАЗОВАНИЯ РОССИЙСКОЙ
ФЕДЕРАЦИИ

МИРЭА - РОССИЙСКИЙ ТЕХНОЛОГИЧЕСКИЙ УНИВЕРСИТЕТ

М.В. СМЕРНОВ, Р.С. ТОЛМАСОВ

**СОЗДАНИЕ ПРИЛОЖЕНИЙ БАЗ ДАННЫХ НА
ЯЗЫКЕ C++**

УЧЕБНО-МЕТОДИЧЕСКОЕ ПОСОБИЕ

Москва — 2020

УДК 004.65:004.43(075.8)

ББК 32.973.2я73

C50

Смирнов М.В. Создание приложений баз данных на языке C++ [Электронный ресурс]: учебно-методическое пособие / Смирнов М.В., Толмасов Р.С. — М., МИРЭА - Российский технологический университет, 2020 — 1 электрон. опт. диск (CD-ROM)

Аннотация.

В учебно-методическом пособии рассмотрены основные аспекты создания приложений для управления данными, такие как: осуществление подключения к источнику данных, добавление, изменение, удаление и представление данных, формирование простых отчётов на основе данных хранящихся в источнике, а также варианты реализации дополнительных функций, расширяющих возможности приложения и повышающих удобство работы с ним. Предложены вопросы и задания для самостоятельной работы, способствующие более глубокому усвоению материала.

Учебное пособие издается в авторской редакции.

Авторский коллектив: Смирнов Михаил Вячеславович, Толмасов Руслан Сергеевич

Рецензенты:

Волосенков В.О., д.т.н., профессор, гл. науч. сотрудник АО «Концерн «Моринформсистема-Агат»»

Андреева О.Н., д.т.н., доцент, начальник отд. науч. работы АО «Концерн «Моринформсистема-Агат»»

Комиссаров Ю.В., к.т.н., доцент, старший преподаватель МОФ МФЮА, г. Сергиев Посад

Минимальные системные требования:

Поддерживаемые ОС: Windows 2000 и выше.

Память: ОЗУ 128МБ.

Жесткий диск: 20 Мб.

Устройства ввода: клавиатура, мышь.

Дополнительные программные средства: Программа Adobe Reader.

Подписано к использованию по решению Редакционно-издательского совета

МИРЭА – Российского технологического университета от () г.

Объем: 7 Мб

Тираж: 10

ISBN _____

© Смирнов М.В., Толмасов Р.С. 2020

© МИРЭА - Российский технологический университет,
2020

СОДЕРЖАНИЕ

ТРЕБОВАНИЯ К АППАРАТНОМУ И ПРОГРАММНОМУ ОБЕСПЕЧЕНИЮ КОМПЬЮТЕРНОГО КЛАССА	4
1 ПОДКЛЮЧЕНИЕ К ПРИЛОЖЕНИЯ К ИСТОЧНИКУ ДАННЫХ	5
1.1 ОРГАНИЗАЦИЯ ПОДКЛЮЧЕНИЯ К БАЗЕ ДАННЫХ В СРЕДЕ ПРОГРАММИРОВАНИЯ QT CREATOR	5
1.2СБОРКА ДРАЙВЕРА ODBC ДЛЯ СОЕДИНЕНИЯ ПРОГРАММНОГО ПРИЛОЖЕНИЯ И БД	6
КОНТРОЛЬНЫЕ ВОПРОСЫ.....	23
ДОПОЛНИТЕЛЬНЫЕ ЗАДАНИЯ	23
2 ВЫВОД ДАННЫХ НА ЭКРАН.....	24
2.1 МОДЕЛИ И ПРЕДСТАВЛЕНИЯ ДАННЫХ В СРЕДЕ ПРОГРАММИРОВАНИЯ QT CREATOR	24
2.2 ОТОБРАЖЕНИЕ СОДЕРЖИМОГО ТАБЛИЦЫ БД В ПРИЛОЖЕНИИ.	25
КОНТРОЛЬНЫЕ ВОПРОСЫ.....	30
ДОПОЛНИТЕЛЬНОЕ ЗАДАНИЕ	31
3 ФУНКЦИИ ДОБАВЛЕНИЯ, ИЗМЕНЕНИЯ И УДАЛЕНИЯ ЗАПИСЕЙ.....	32
3.1 ОБРАБОТКА ЗАПРОСОВ К БАЗЕ ДАННЫХ ИЗ ПРИЛОЖЕНИЯ.....	32
3.2 ДОБАВЛЕНИЕ НОВОЙ СТРОКИ В ТАБЛИЦУ БД	33
3.3 УДАЛЕНИЕ ВЫБРАННОЙ ЗАПИСИ	39
3.4 ИЗМЕНЕНИЕ ВЫБРАННОЙ ЗАПИСИ	39
3.5 ДОСТУП К ФУНКЦИЯМ ИЗМЕНЕНИЯ И УДАЛЕНИЯ СТРОКИ ТАБЛИЦЫ ЧЕРЕЗ КОНТЕКСТНОЕ МЕНЮ.	40
КОНТРОЛЬНЫЕ ВОПРОСЫ.....	47
ДОПОЛНИТЕЛЬНОЕ ЗАДАНИЕ	47
4 ВЫВОД ДАННЫХ ИЗ ПРИЛОЖЕНИЯ.....	48
4.1 ЭКСПОРТ ТАБЛИЦЫ В СТОРОННЕЕ ПРИЛОЖЕНИЕ	48
4.2 ЭКСПОРТ ПРОСТОГО ОТЧЁТА В ФОРМАТЕ PDF.....	54
КОНТРОЛЬНЫЕ ВОПРОСЫ.....	57
ДОПОЛНИТЕЛЬНОЕ ЗАДАНИЕ	57
5 ДОПОЛНИТЕЛЬНЫЕ ФУНКЦИИ ПРИЛОЖЕНИЯ.....	58
5.1 ДОБАВЛЕНИЕ ИЗОБРАЖЕНИЙ И ПОСТРОЕНИЕ ГРАФИКОВ СРЕДСТВАМИ ПРИЛОЖЕНИЯ.	58
5.2 ПОСТРОЕНИЕ ГРАФИКА ПО ДАННЫМ ТАБЛИЦЫ БД.....	61

5.3	РЕАЛИЗАЦИЯ ВЫБОРА ЗНАЧЕНИЯ АТТРИБУТА ИЗ НИСПАДАЮЩЕГО СПИСКА ...	70
5.4	УПРОЩЕНИЕ ПРОЦЕССА ВВОДА И РЕДАКТИРОВАНИЯ ДАННЫХ	75
	КОНТРОЛЬНЫЕ ВОПРОСЫ.....	77
	ДОПОЛНИТЕЛЬНОЕ ЗАДАНИЕ	77
	СПИСОК РЕКОМЕНДОВАННЫХ ИСТОЧНИКОВ	78
	ПРИЛОЖЕНИЯ	79
A.	ИНСТРУКЦИЯ ПО УСТАНОВКЕ QT CREATOR.....	80
B.	УСТАНОВКА MICROSOFT SQL SERVER	86
C.	УСТАНОВКА MICROSOFT SQL SERVER MANAGEMENT STUDIO	91

ТРЕБОВАНИЯ К АППАРАТНОМУ И ПРОГРАММНОМУ ОБЕСПЕЧЕНИЮ КОМПЬЮТЕРНОГО КЛАССА

Минимальные требования к ПК:

Операционные системы:

Рекомендуется: Windows 7 и выше, Допускается: Linux (Ubuntu, Archi и некоторые другие), Mac OS¹.

Процессор:

Intel или совместимый процессор с тактовой частотой 1,4 ГГц и выше (рекомендуется 2 ГГц и выше)

ОЗУ:

1 ГБ ОЗУ (рекомендуется 2 ГБ и выше)

HDD/SDD:

2,2 ГБ свободного места на диске.

Программное обеспечение:

- 1) IDE Qt Creator – для разработки приложений.
- 2) СУБД MS SQL Server Express² - для создания локального сервера баз данных.
- 3) MS SQL Server Management Studio³ - удобный клиент для управления SQL-сервером.

Инструкции по загрузке и установке указанных программ приведены в приложении.

¹ Mac OS не имеет прямой поддержки MS SQL Server и MS SSMS, тем не менее установка сервера на данную операционную систему возможна. Также следует учитывать тот факт, что операции чтения/записи файлов для MacOS и Linux могут отличаться от указанных в пособии в силу различий в организации файловых систем и политик безопасности.

² Можно выбрать любой выпуск (2008, 2012, 2014 и т.д.) так как в лабораторных работах используются только основные инструкции SQL, одинаково поддерживаемые всеми выпусками. В данном пособии используется SQL Server Express 2017

³ Версия MS SQL SMS должна поддерживать установленную на компьютере СУБД. Может потребоваться аналог SSMS для Linux и Mac OS

1 ПОДКЛЮЧЕНИЕ К ПРИЛОЖЕНИЯ К ИСТОЧНИКУ ДАННЫХ

1.1 Организация подключения к базе данных в среде программирования Qt Creator

Подключение приложения к различным источникам данных сложный процесс, который требует хороших знаний об организации СУБД и навыков в написании низкоуровневого программного кода. Учитывая тот факт, что большинство разрабатываемых бизнес-приложений так или иначе взаимодействует с базами данных, необходимые для выполнения большинства действий методы и классы уже описаны в стандартных библиотеках, что существенно упрощает процесс создания приложений для работы с базами данных.

Для того чтобы получить доступ к библиотекам для работы с базами данных SQL, необходимо предварительно подключить Qt-модуль «*sql*» в главном файле проекта (файл с расширением .pro).

Работа с БД в Qt проходит на трёх разных слоях:

- 1) Слой драйверов БД — данный слой представляет собой низкоуровневый мост между определенными базами данных и слоем более высокого уровня. Данный уровень включает классы *QSqlDriver*, *QSqlDriverCreator*, *QSqlDriverCreatorBase*, *QSqlDriverPlugin*, *QSqlResult*;
- 2) Слой SQL API — данный слой предоставляет доступ к базам данных. Соединения устанавливаются с помощью класса *QSqlDatabase*, на этом же слое осуществляется взаимодействие с базой данных с помощью класса *QSqlQuery*.
- 3) Слой пользовательского интерфейса — данный слой позволяет отразить данные, хранящиеся в базе на специальных визуальных элементах пользовательского интерфейса с помощью экземпляров классов-моделей. К моделям относятся следующие классы: *QSqlQueryModel*, *QSqlTableModel* и *QSqlRelationalTableModel*.

Класс *QSqlDatabase* необходим для установления соединения. В процессе создания объекта этого класса, необходимо определить плагин (или драйвер), который должен быть использован для попытки установления связи с выбранным источником данных. Qt поддерживает возможность работы со множеством БД с помощью следующих плагинов:

- QDB2 — IBM DB2

- QIBASE — Borland InterBase
- QMYSQL / MARIADB — MySQL or MariaDB
- QOCI — драйвер Oracle Call Interface
- QODBC (Open Database Connectivity (ODBC)) - Microsoft SQL Server и другие ODBC-совместимые источники данных (MS Access, Excel и другие).
- QPSQL — PostgreSQL
- QSQLITE2 — SQLite

Полный список поддерживаемых источников данных можно найти в документации к проекту и встроенной справочной системе приложения.

Таким образом, работа с базой данных из приложения выполняется с помощью специальной программной прослойки, которая реализована на трёх уровнях (слоях):

- низкоуровневом, который отвечает за обмен управляющими сигналами и данными между приложением и СУБД;
- высокоуровневом, который отвечает за сбор и передачу данных необходимых для открытия соединения, контроля его состояния, а также для управления взаимодействием с объектами, которые хранятся в СУБД.
- пользовательский, который отвечает за отображение данных, хранящихся в БД в специальных визуальных элементах пользовательского интерфейса.

Далее будем рассматривать процесс сборки драйвера и установки соединения на примере использования плагина QODBC, который содержит драйвер для работы с MS SQL Server.

1.2 Сборка драйвера ODBC для соединения программного приложения и БД

Подготовка к выполнению

Перед началом выполнения лабораторных работ необходимо создать базу данных, к которой будет подключаться разрабатываемое приложение, и таблицу внутри этой базы, с которой приложение будет работать.

Для примера будет создана база данных “*QtCourse*” содержащая таблицу “*product*”(Листинг 1.1). Данные объекты проще всего создать через SQL Server Management Studio, подключившись к нужному серверу и выполнив следующий скрипт:

```

CREATE DATABASE qtcourse
GO
USE qtcourse
GO
CREATE TABLE product
(
    ID int IDENTITY PRIMARY KEY,
    name varchar(30),
    cat_ID varchar(30)
);

```

С целью упрощения контроля состояния подключения и правильности выполнения запросов к БД из создаваемого приложения, необходимо заполнить созданную таблицу тестовыми данными, для этого необходимо выполнить следующий скрипт (Листинг 1.2):

Листинг 1.2 - Добавление новых строк к таблице БД

```

INSERT INTO product VALUES ( 'Яблоко' , 'Фрукты' ) ,
                             ( 'Груша' , 'Фрукты' ) ,
                             ( 'Картофель' , 'Овощи' ) ,
                             ( 'Огурец' , 'Овощи' ) ;

```

Следует обратить внимание, что столбец ID не указывается в списке добавляемых значений, так как при создании таблицы этому столбцу было задано свойство *IDENTITY*.



Поле, для которого задано свойство *IDENTITY[(m,n)]* становится автоинкрементным, то есть в поле автоматически заносится начальное значение *m*, а каждое следующее значение увеличивается на определённую величину *n* автоматически при добавлении новой строки. Если значение *m* и *n* не задано, то по умолчанию они принимаются равными 1. Значения автоинкрементных полей не могут быть отредактированы или заданы пользователем, они так же не могут повторяться в пределах одной таблицы, что позволяет использовать их в качестве суррогатных первичных ключей.

Создание нового приложения

Для создания нового приложения необходимо открыть среду разработки Qt Creator. После запуска, в окне приветствия (Рисунок 1.1), на вкладке «Проекты», необходимо нажать на кнопке «Создать».

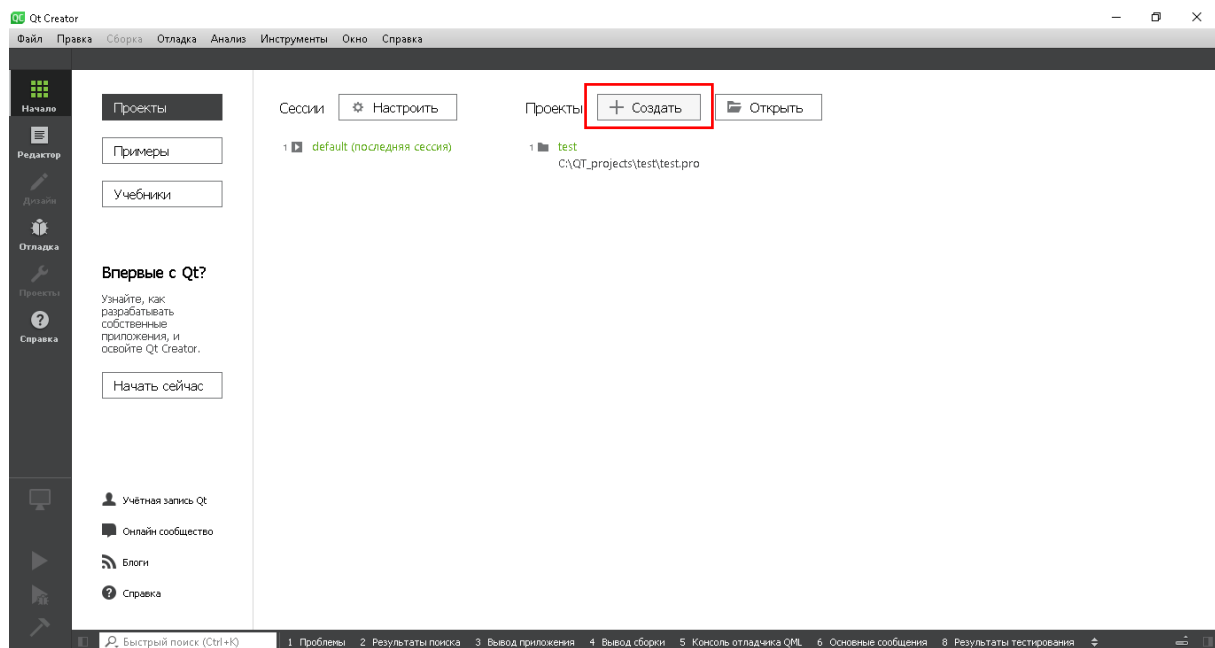


Рисунок 1.1 Окно приветствия среды разработки Qt Creator

В открывшемся диалоговом окне (Рисунок 1.2) можно выбрать необходимый для решения поставленных задач шаблон. В Qt представлен широкий выбор шаблонов. Для выполнения приведённых здесь лабораторных работ следует выбрать шаблон «Приложение Qt», после выбора которого, следует нажать на кнопку «Выбрать...».

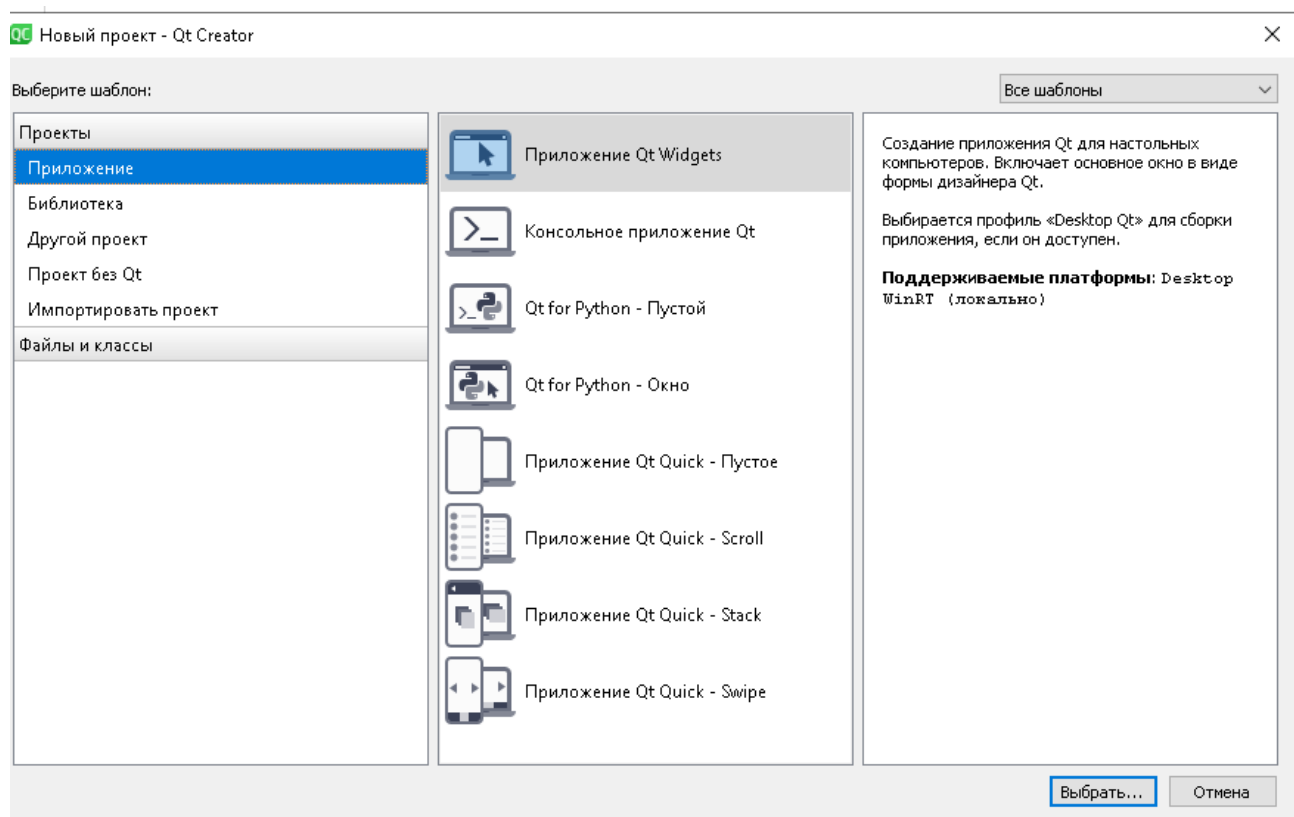


Рисунок 1.2 Выбор шаблона для нового проекта

Далее следует выбрать место, где будет храниться проект и его название (Рисунок 1.3). В названии проекта допускается использование прописных и строчных букв, а также цифр и символа «_». Нельзя называть проект с цифры и символа «_». В случае ошибки продолжение создания проекта будет невозможно, до её исправления.

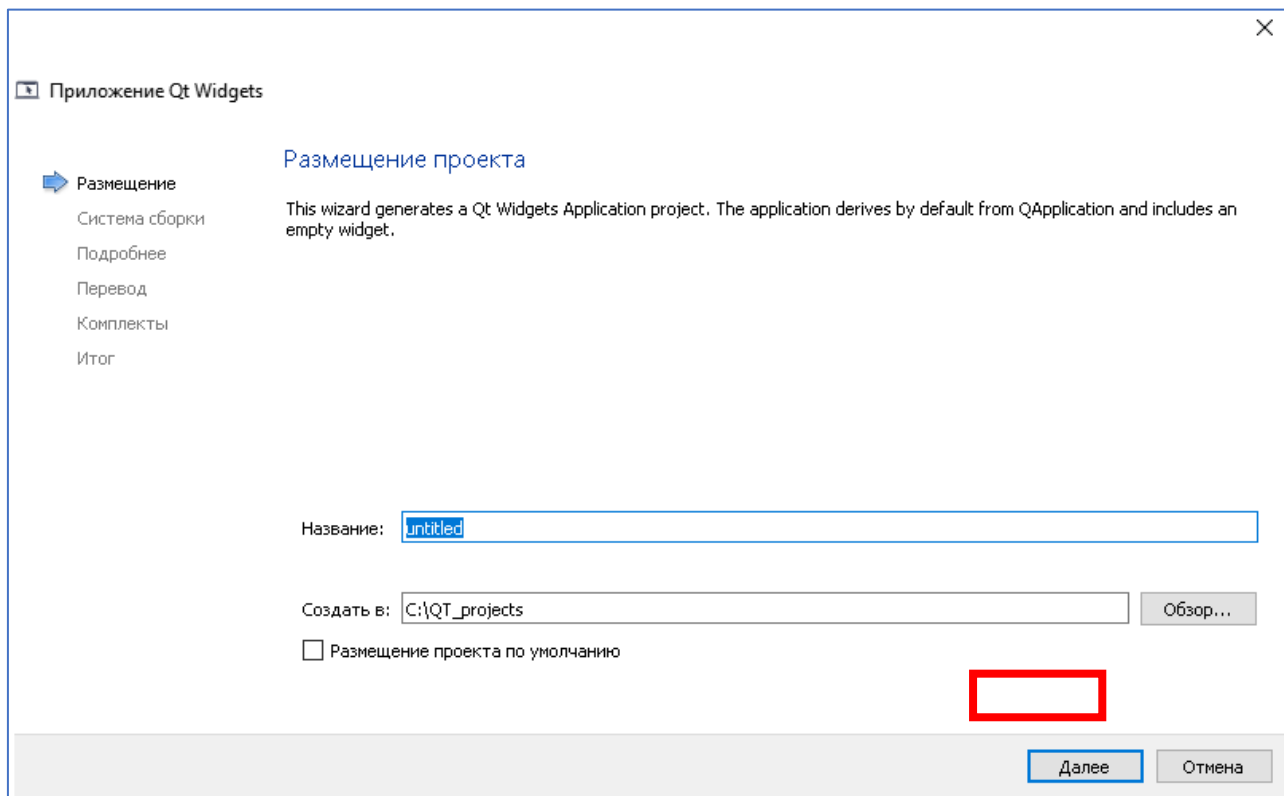


Рисунок 1.3 Выбор места размещения и названия проекта



Следует обратить внимание, что путь к файлам создаваемого проекта **не должен** содержать букв кириллицы, в противном случае возникнут проблемы с компиляцией решения компонентом “*qmake.exe*”

После указания названия проекта и пути к месту хранения его файлов необходимо нажать кнопку «Далее» (Рисунок 1.3).

На следующем этапе создания приложения будет предложено выбрать систему сборки (Рисунок 1.4). Для выполнения приведённых ниже заданий необходимо выбрать систему *qmake*, после чего следует нажать «Далее»

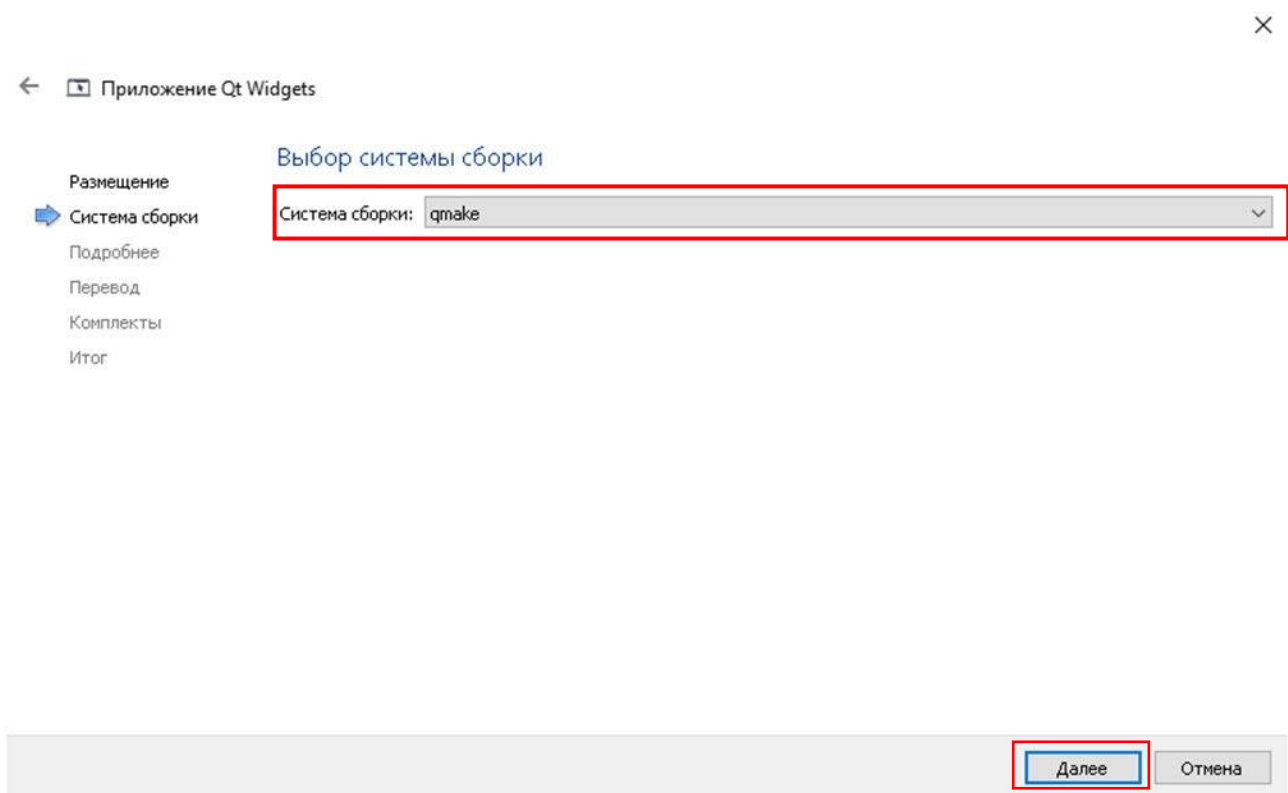


Рисунок 1.4 - Окно выбора системы сборки

В следующем окне пользователю предлагается выбрать имя класса, его базовый класс и имена связанных с ним файлов (Рисунок 1.5). Эти данные будут использованы при создании шаблонов файлов исходных текстов. Оставив указанные данные без изменений и, нажмём кнопку «Далее».

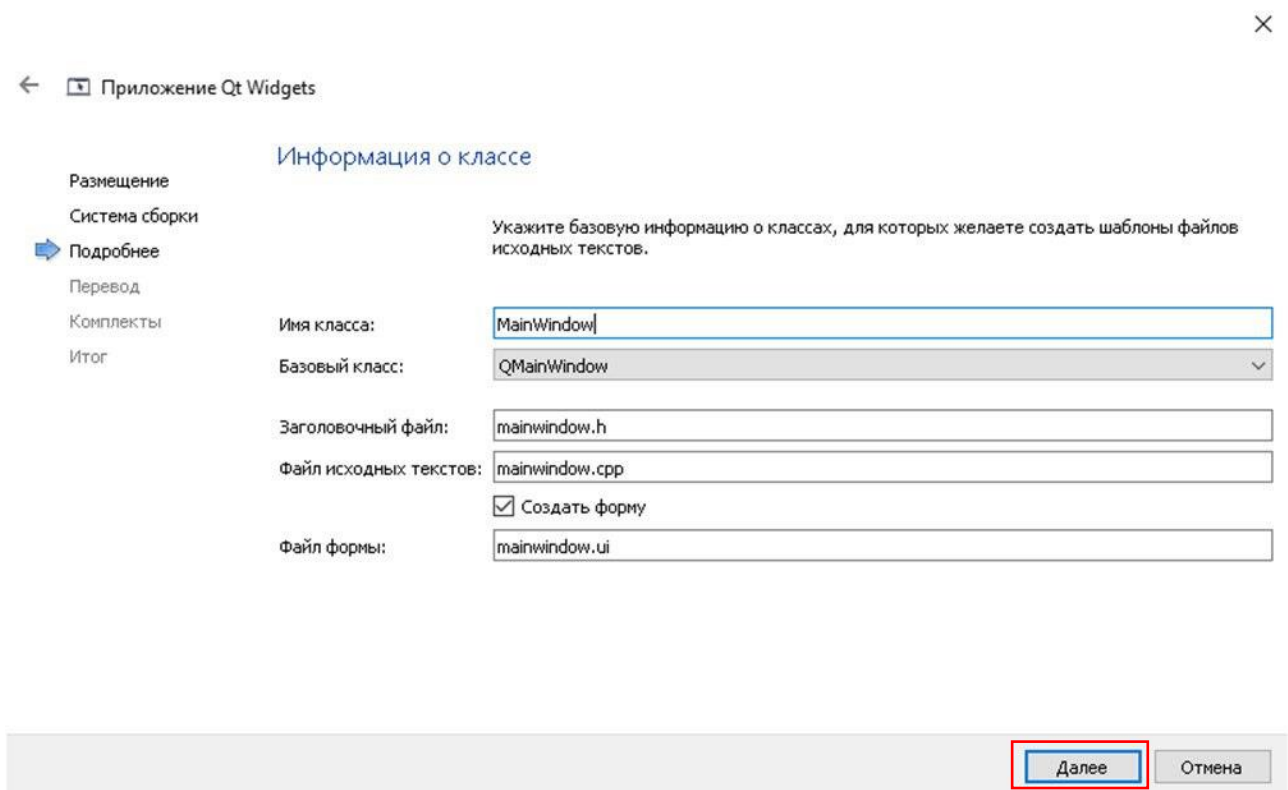


Рисунок 1.5 - Окно ввода информации о классе

Следующее окно служит для указания языков, которые будет поддерживать разрабатываемое приложение, пропустите его нажав «Далее». Последнее окно содержит информацию об используемых комплектах. Оставьте комплекты, выбранные по умолчанию, и нажмите «Далее» после чего завершите создание проекта.

Если в процессе создания не прозойдет никаких ошибок, то диалоговое окно создания проекта и окно приветствия будут закрыты и будет открыто окно работы с проектом (Рисунок 1.6). На рисунке ниже указаны основные элементы рабочего пространства, большую часть которого занимает редактор кода :

1. Навигация по проекту.
2. Панель быстрого доступа.
3. Менеджер файлов.

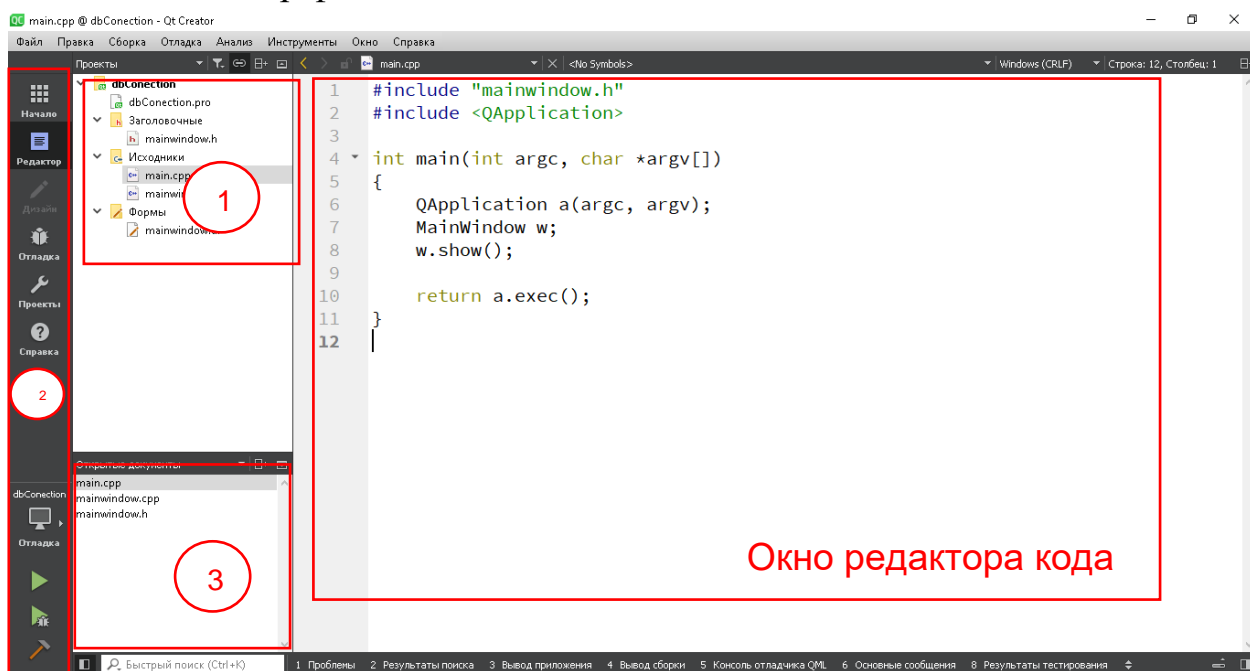


Рисунок 1.6 Главное рабочее окно

1. Добавление новой экранной формы к проекту

Строка подключения, в зависимости от целевой СУБД, используемого языка и особенностей среды программирования, может незначительно отличаться от указанной в настоящем разделе. Тем не менее, независимо от выбранных средств разработки она будет включать следующие элементы:

- явное указание используемого драйвера;
- адрес целевого сервера;
- имя целевой базы данных на данном сервере;
- логин и пароль (может не указываться при соответствующих настройках сервера БД).

Очевидно, что для установления соединения пользовательского приложения с сервером, пользователю необходимо передать приложению как минимум эти четыре параметра. С этой целью, необходимо предусмотреть несколько полей ввода, которые можно расположить и на главной форме, однако, если в дальнейшем данное приложение будет использоваться как составная часть какого-либо проекта, то такое размещение будет только мешать. Более логичным решением выглядит добавление к проекту диалогового окна, где и будут расположены требуемые элементы.

Для добавления новых файлов в проект необходимо:

- 1) Щелкнуть правой кнопкой мыши по корневой папке проекта в окне навигации по проекту.
- 2) В открывшемся меню выбрать пункт «Добавить новый...» (Рисунок 1.7).

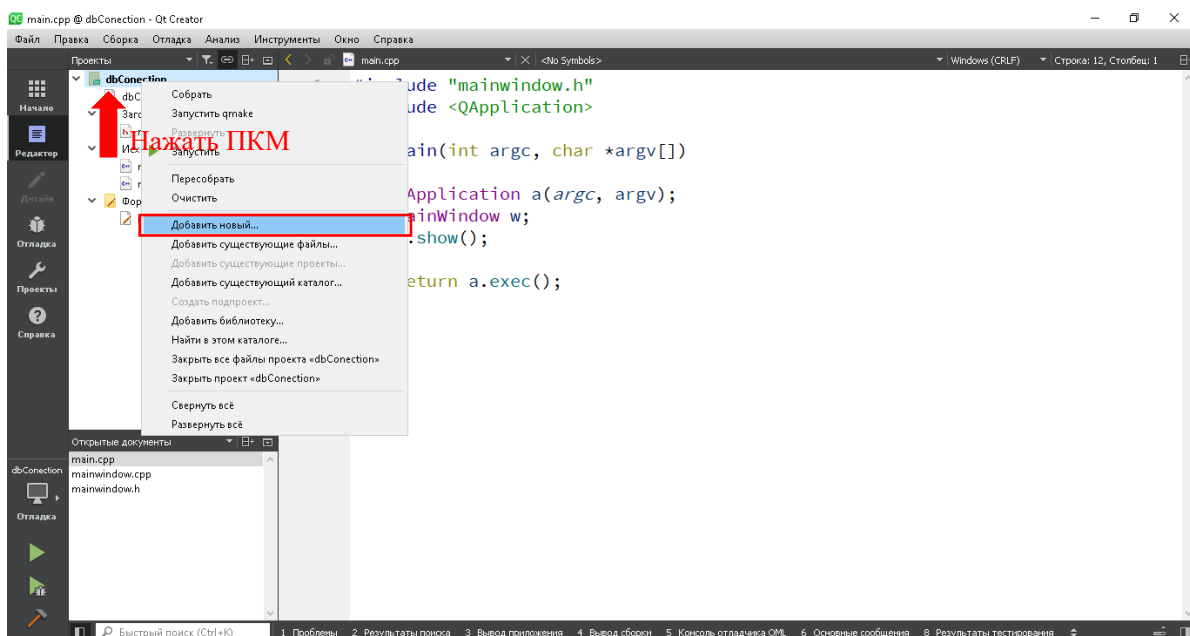


Рисунок 1.7 Добавление новых объектов в проект

- 3) В диалоговом окне необходимо выбрать компонент «Qt \ Класс формы Qt Designer» (Рисунок 1.8).

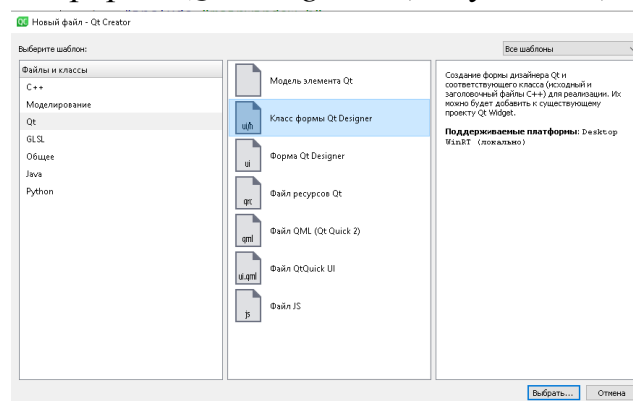


Рисунок 1.8 Добавление нового компонента в проект



Следует выбрать именно

«Класс формы Qt Designer»,

так как при создании этого компонента одновременно создаётся форма, заголовочный и исходный файл. В противном случае, добавлять и редактировать эти файлы в проект придётся вручную.

- 4) В мастере создания компонента необходимо выбрать “*Widget*” в качестве шаблона формы.

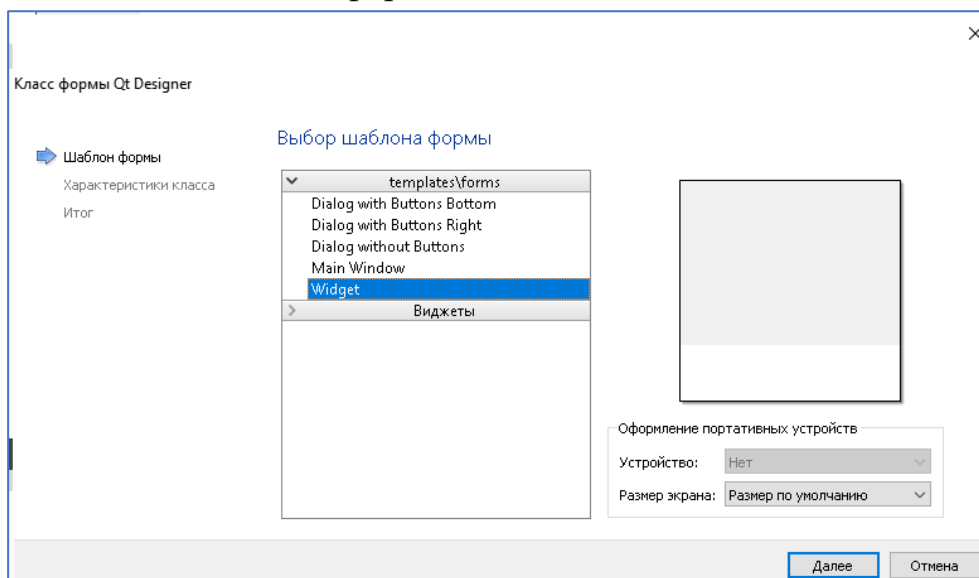


Рисунок 1.9 Выбор шаблона формы

- 5) В следующем окне достаточно ввести имя нового класса и все создаваемые дополнительно файлы будут автоматически переименованы в соответствии с главным классом. Например, “*ConnectionDialog*”.

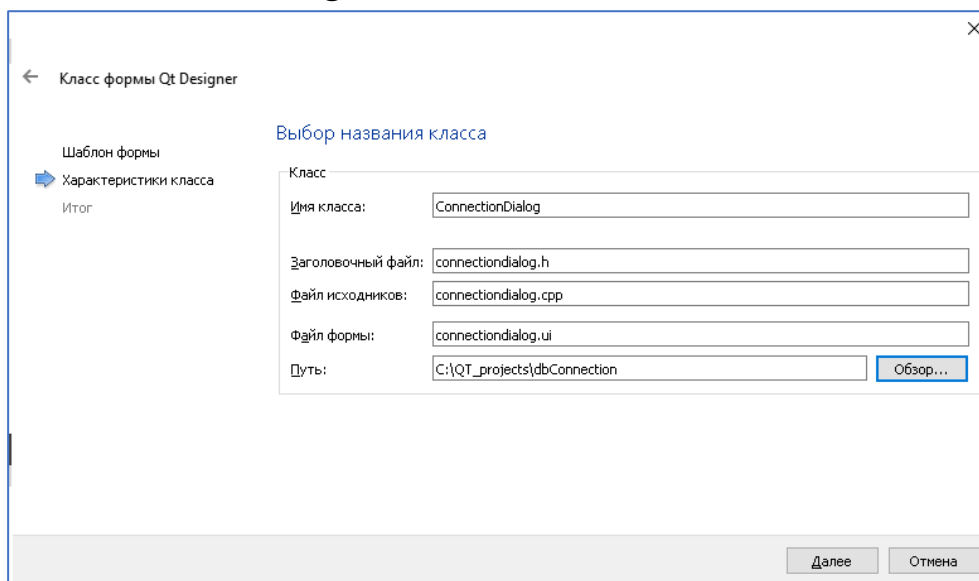


Рисунок 1.10 Ввод имени класса и файлов

б) В следующем диалоговом окне ничего менять не требуется.

После создания новой формы откроется окно «Дизайн», где можно редактировать её внешний вид (Рисунок 1.11).

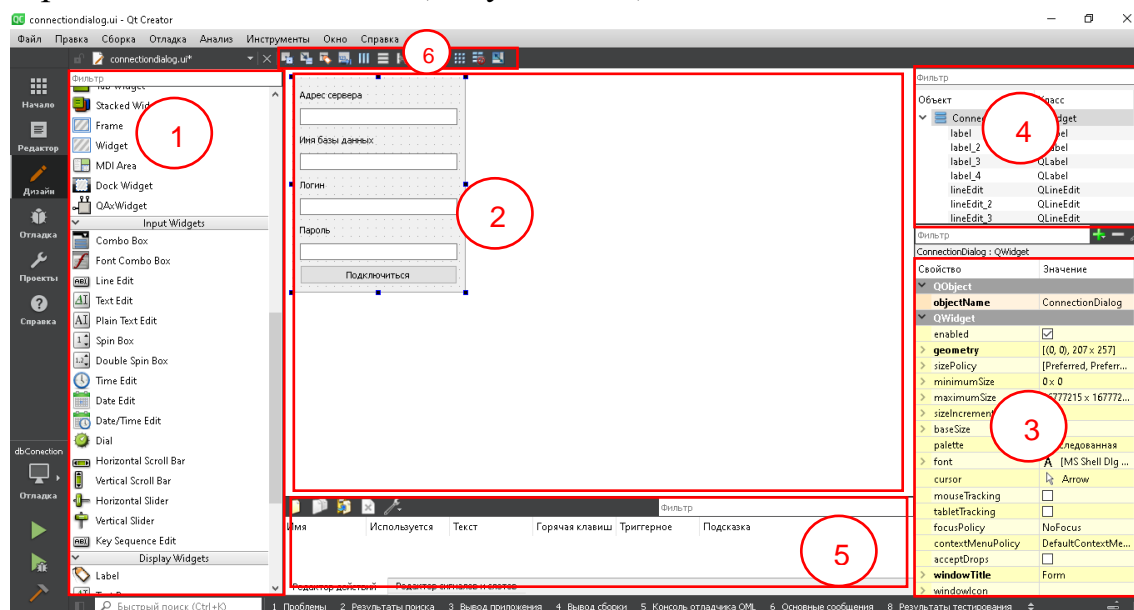


Рисунок 1.11 Окно редактора формы

Необходимо разместить на форме четыре объекта *lineEdit*, кнопку *pushButton*, и четыре *lable*, перетащив их из каталога элементов (1, Рисунок 1.11). Для изменения названий элементов *lable* и *pushButton* следует дважды щелкнуть по ним левой кнопкой мыши.

Редактор форм позволяет не только размещать компоненты на форме, менять её размеры и размеры отдельных компонентов, но также группировать элементы, тем самым ускоряя создание пользовательского интерфейса приемлемого качества.

Для осуществления группировки всех элементов, расположенных на форме, необходимо щелкнуть левой кнопкой мыши на пустом месте формы и выбрать один из вариантов группировки либо в верхней части экрана (6, Рисунок 1.11), либо сочетанием клавиш “CTRL+L” или “CTRL+H” для компоновки по вертикали и горизонтали соответственно.

Визуальная часть компонента *ConnectionDialog* настроена, однако она не выполняет своих прямых функциональных обязанностей, а именно не осуществляет подключения приложения к БД.

Реализация данной возможности будет помещена в код обработчика нажатия на кнопку «Подключиться». Для создания обработчика необходимо

щёлкнуть правой кнопкой мыши на кнопку и выбрать пункт «Перейти к слоту...» в открывшемся меню (Рисунок 1.12).

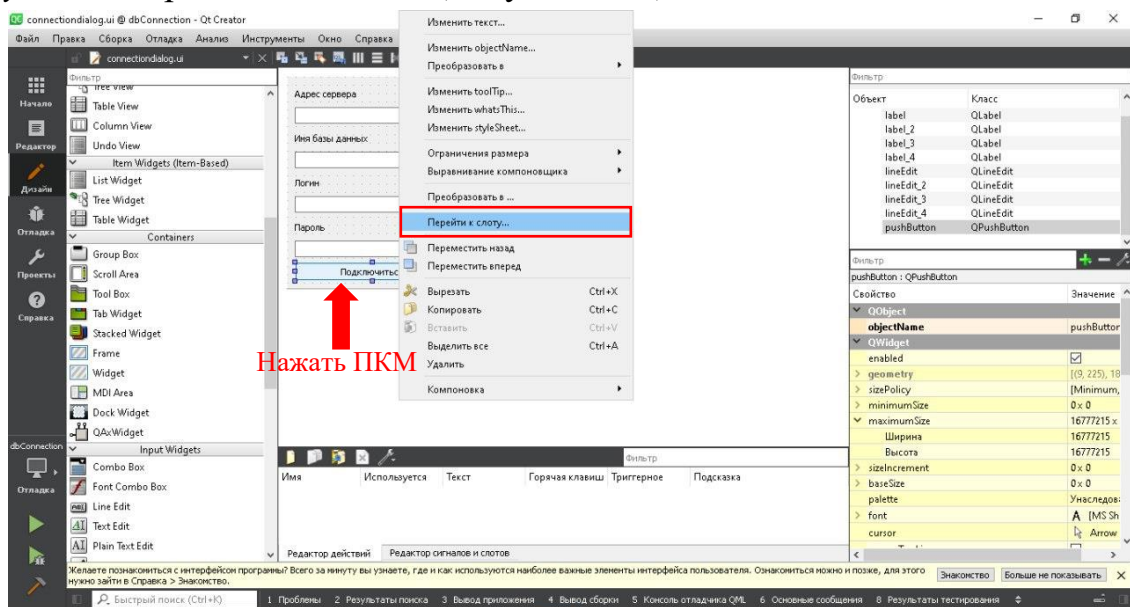


Рисунок 1.12 - Переход к слоту объекта QPushButton



В Qt для коммуникации между объектами используется отличный от многих фреймворков механизм сигналов и слотов.

Сигнал — появляется в качестве реакции на какое-либо событие (нажатие, перетаскивание и т.д.).

Слот — функция (метод), которая вызывается в ответ на определенный сигнал.

В открывшемся диалоговом окне, необходимо выбрать сигнал, при возникновении которого, будет выполнен связанный метод. В данном случае следует выбрать сигнал *clicked()*, так как нас интересует факт нажатия на кнопку (Рисунок 1.13).

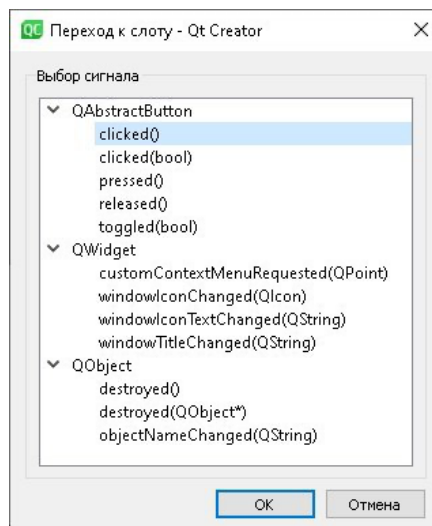


Рисунок 1.13 - Выбор нужного сигнала

В результате откроется окно редактора кода с открытым файлом “connectiondialog.cpp” в конце которого будет добавлен шаблон метода. Он будет вызываться каждый раз при нажатии на эту кнопку. Так же в заголовочном файле “connectiondialog.h” автоматически добавится необходимая информация о созданном слоте (*on_pushButton_clicked()* в разделе *private slots* (Листинг 1.4))

Сборка драйвера ODBC

Перед началом работы с библиотеками и командами, так или иначе связанными с командами языка SQL, в корневом файле проекта (в рассматриваемом примере это *dbConnection.pro*) необходимо добавить “*sql*” в строку (Листинг 1.3):

Листинг 1.3 - Добавление сведений об используемых наборах библиотек

```
QT += core gui sql
```

Для вступления изменений в силу необходимо перестроить проект. Для этого можно воспользоваться одной из 3 кнопок в левом нижнем углу панели быстрого выбора (Рисунок 1.14). Первые две кнопки, с изображением зелёного треугольника запускают приложение (разница в режиме запуска, во втором случае будут учитываться точки останова, установленные пользователем), последняя кнопка с изображением молотка позволяет выполнить сборку проекта без его запуска. Так как запускать приложение сейчас не требуется нажмите на кнопку с изображением молотка и нажать на кнопку.

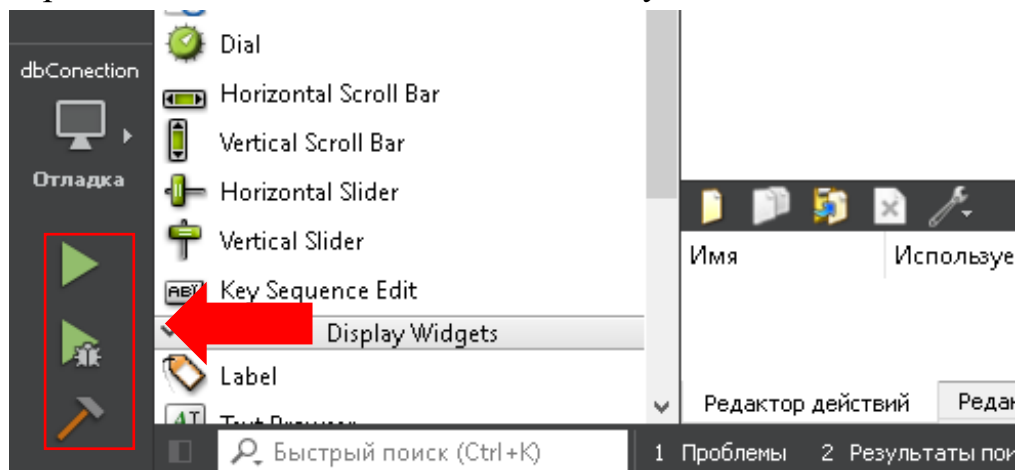


Рисунок 1.14 - Элементы управления запуском приложения

В открывшемся диалоговом окне нужно нажать на кнопку «Сохранить всё» и дождаться окончания сборки.

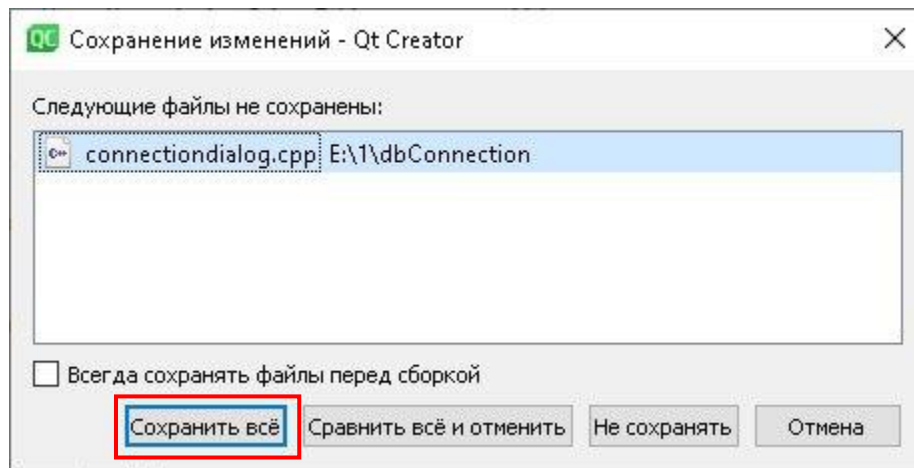


Рисунок 1.15 - Сохранение внесенных изменений

Далее следует перейти в заголовочный файл *connectiondialog.h*.

В этом файле необходимо подключить ряд библиотек и создать несколько переменных. Строки, добавленные в файл выделены полужирным (Листинг 1.4):

Листинг 1.4 - Подключение необходимых библиотек и объявление переменных

```
#include <QWidget>
#include <QtSql>
#include <QMessageBox>

namespace Ui {
class ConnectionDialog;
}

class ConnectionDialog : public QWidget
{
    Q_OBJECT

public:
    explicit Form(QWidget *parent = nullptr);
    ~ConnectionDialog();
    QMessageBox*msg;
private slots:
    void on_pushButton_clicked(); //создано при создании слота

private:
    Ui::ConnectionDialog *ui;
    QSqlDatabase db;
};
```

Для работы с подключениями к БД применяется библиотека *QtSql*, для уведомления пользователей о состоянии подключения в данной лабораторной работе будет использоваться объект класса *QMessageBox*, представляющий собой диалоговое окно с текстом (выделенные жирным строки в разделе *#include*, Листинг 1.4).

В разделе *private* объявлены переменные следующих классов:

- *QSqlDatabase* — класс из библиотеки *QtSql*, содержащий необходимые для установки соединения методы;
- *QMessageBox* — класс содержащий все необходимые методы для работы с простыми текстовыми сообщениями во всплывающих диалоговых окнах.

После внесения изменений в заголовочный файл, можно переходить к написанию обработчика нажатия на кнопку «Подключиться» (Листинг 1.5):

Листинг 1.5 - Обработчик нажатия на кнопку "Подключиться"

```
void ConnectionDialog::on_pushButton_clicked()
{
    //В следующей строке указывается используемый драйвер
    db = QSqlDatabase::addDatabase("QODBC");

    /*Перенос в следующей строке использовался для удобства и не
    является обязательным*/
    //Далее формируется строка подключения
    db.setDatabaseName("DRIVER={SQL Server};SERVER="
        +ui->lineEdit->text()+
        ";DATABASE="+ui->lineEdit_2->text()+"");
    //Указывается логин и пароль соответственно
    db.setUserName(ui->lineEdit_3->text());
    db.setPassword(ui->lineEdit_4->text());

    msg = new QMessageBox(); //создаём объект для вывода сообщения

    if (db.open()) //попытка подключения к БД
    {
        msg->setText("Соединение установлено"); //
    }
    else
    {
        msg->setText("Соединение НЕ установлено");
    }

    msg->show();
}
```

Обеспечение доступа к диалоговому окну

Чтобы пользователь мог ввести данные в созданную ранее форму, он должен иметь возможность перехода к ней с главного окна приложения. Для этой цели создадим объект класса “*QMenu*”.

Объект класса “*QMenu*” уже помещён на главную форму, но он не активен до тех пор, пока не содержит в себе элементов (пунктов меню и подменю). Чтобы активировать объект и отобразить его на главной форме необходимо создать

хотя бы один элемент. Для этого, нужно выбрать файл “mainwindow.ui” и дважды щёлкнув на нём перейти в окно редактора формы «Дизайн». В верхней части макета экранной формы, нужно воспользоваться приглашением «Пишите здесь...» (Рисунок 1.16) и дважды щёлкнув левой кнопкой мыши ввести заголовок «Подключения».

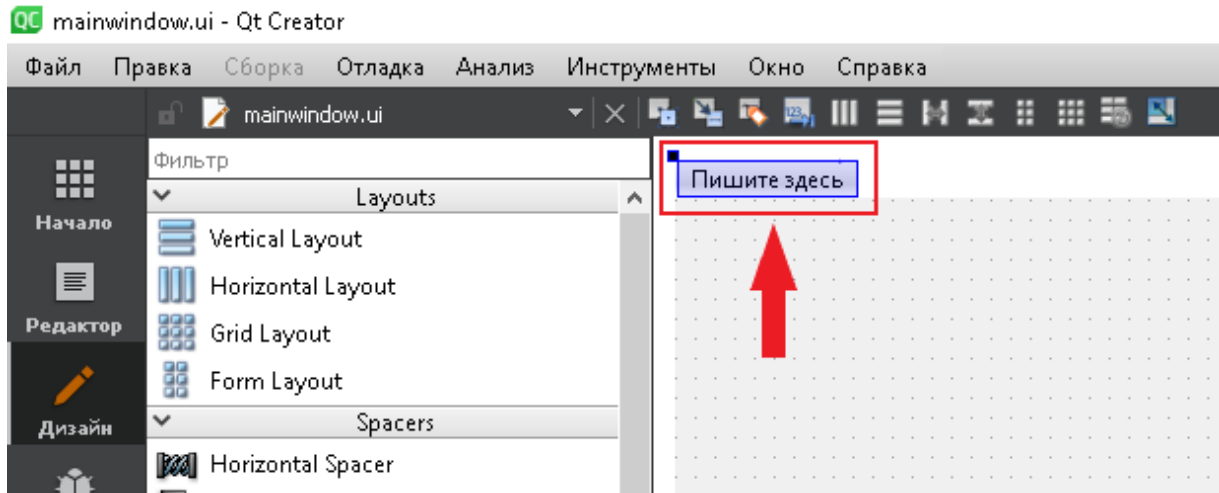


Рисунок 1.16 - Создание главного меню приложения

Далее необходимо создать пункт меню «Подключиться к БД...» для вызова экранной формы “ConnectionDialog” (Рисунок 1.17).

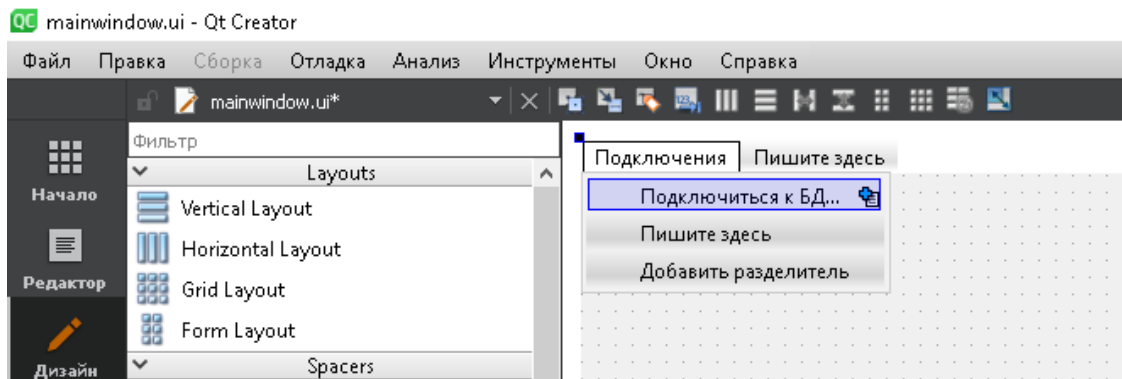


Рисунок 1.17 - Добавление пункта меню

Следующим шагом, создадим метод обработки выбора соответствующего пункта меню. Создание обработчика для объекта “QMenu” несколько отличается от создания обработчика для обычной кнопки. Для того чтобы перейти к доступному слоту необходимо выбрать соответствующий элемент в нижней части экрана и щёлкнув по нему правой кнопкой мыши выбрать «Перейти к слоту...» в контекстном меню.

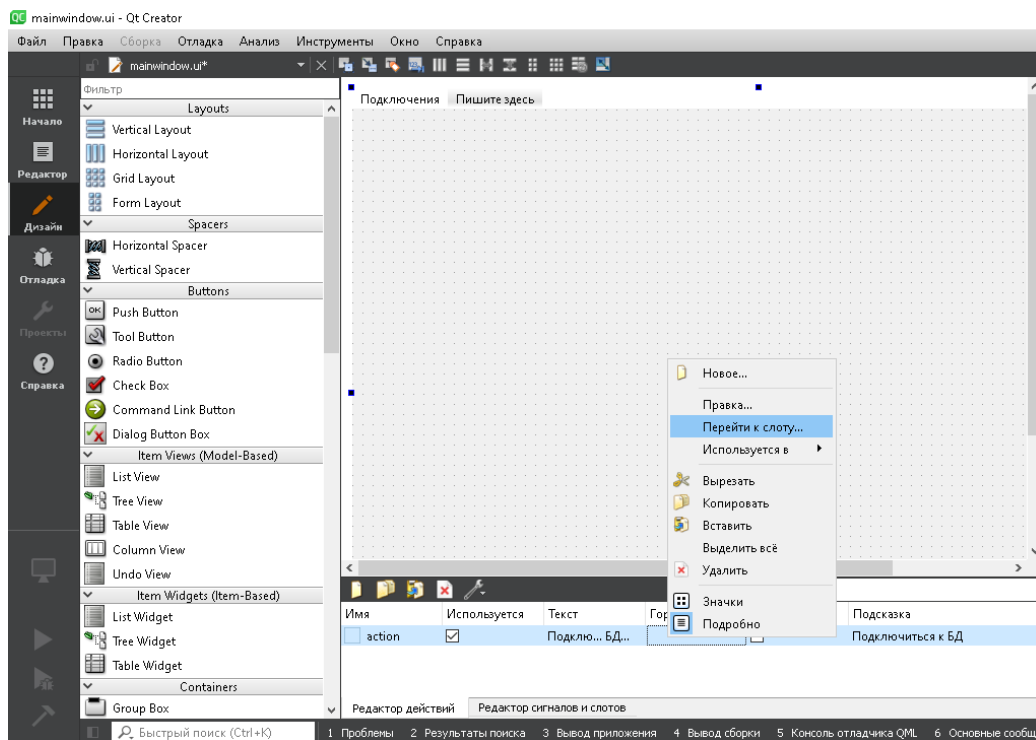


Рисунок 1.18 - Создание слота обработки нажатия на пункт меню



Набор сигналов для пунктов меню несколько отличается от набора для обычной кнопки, например, здесь нет слота “*clicked()*”, при нажатии испускается сигнал “*triggered()*”

Данном случае следует выбрать сигнал *triggered()* без параметров (Рисунок 1.19).

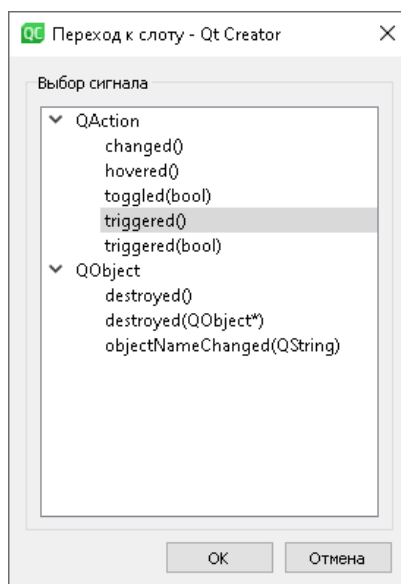


Рисунок 1.19 - Выбор сигнала для пункта меню

После подтверждения выбора будет создана заготовка метода (слота) *on_action_triggered()*, но перед тем, как будет написан код перехода к экранной форме подключения, необходимо внести некоторые изменения в заголовочный

файл *“mainwindow.h”*. Во-первых, необходимо подключить созданные ранее модули. Во-вторых – создать указатель на объект класса *“ConnectionDialog”*, в приведённом ниже листинге, добавляемые строки выделены полужирным шрифтом (Листинг 1.6):

Листинг 1.6 - Подключение новых модулей к заголовочному файлу главной формы

```
#ifndef MAINWINDOW_H
#define MAINWINDOW_H

#include <QMainWindow>
#include "connectiondialog.h"
#include "ui_connectiondialog.h"

QT_BEGIN_NAMESPACE
namespace Ui { class MainWindow; }
QT_END_NAMESPACE

class MainWindow : public QMainWindow
{
    Q_OBJECT

public:
    MainWindow(QWidget *parent = nullptr);
    ~MainWindow();

private slots:

    void on_action_triggered();

private:
    Ui::MainWindow *ui;
    ConnectionDialog *dlg;
};
#endif // MAINWINDOW_H
```



Совет: чтобы избежать ошибок, название модулей можно скопировать из файла с исходным кодом формы подключения (*connectiondialog.cpp*) (первая и вторая строка), в таком случае можно избежать ошибки связанной с неправильным написанием названия модулей.

После добавления названий модулей (*connectiondialog.h* и *ui_connectiondialog.h*) и объявления переменной (*dlg*), можно перейти к автоматически созданной заготовке метода *on_action_triggered()* и описать действия необходимые для вызова формы подключения (**Ошибка! Источник с ссылки не найден.**):

Листинг 1.7 - Метод обработки нажатия на кнопку меню "Подключится к БД"

```
void MainWindow::on_action_triggered()
{
    dlg = new ConnectionDialog();
    dlg->show();
}
```

После этого, можно запустить проект и убедиться в возможности подключения к БД.

Пробный запуск

Для запуска приложения необходимо нажать на зелёный треугольник в левом нижнем углу окна Qt Creator. После успешной сборки откроется главное окно созданного Вами приложения. При помощи меню нужно вызвать форму подключения и заполнить данными для подключения. Параметры подключения, указанные на рисунке, ниже приведены для примера для примера (Рисунок 1.20).

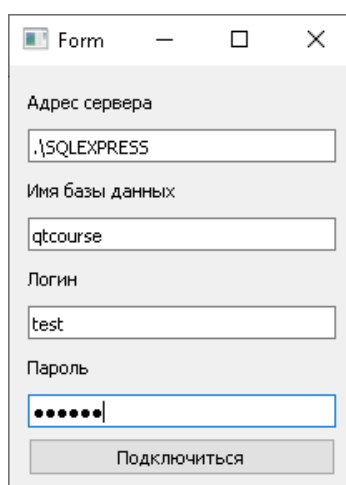


Рисунок 1.20 - Ввод данных в окно подключения



Данные, необходимые для заполнения формы, можно получить через среду SSMS:

- имя сервера – в свойствах сервера;
- имя нужной базы данных – в папке «Базы данных»;
- имя входа – в папке «Безопасность/Имена входа».

Пароль следует запоминать или записывать, узнать его через SSMS невозможно.

В случае подключения к удалённому серверу, данную информацию необходимо уточнять у администратора.

В случае успешного подключения приложение выдаст сообщение (Рисунок 1.21):

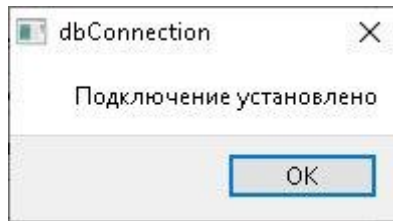


Рисунок 1.21 - Окно подтверждения подключения

Если подключение не будет установлено, следует последовательно выполнить следующие действия:

- 1) Проверить корректность введенных данных.
- 2) Выполнить подключение без указания логина и пароля (в таком случае будет выполнена авторизация средствами Windows).
- 3) Проверить правильность написания строки подключения (имена элементов, отсутствие лишних пробелов и т.д.)

Контрольные вопросы

1. Какой класс необходим для установления соединения с базой данных?
2. На каких трёх слоях (уровнях) Qt осуществляет взаимодействие с базой данных?
3. Назовите назначение каждого слоя (уровня) взаимодействия.
4. Найдите в Интернете описание ODBC и кратко опишите его назначение и перечень поддерживаемых им источников данных.
5. Каким плагином следует воспользоваться, если нужно осуществить подключение к СУБД MySQL?

Дополнительные задания

1. Добавить функцию разрыва текущего соединения с БД.
2. Доработать метод `on_pushButton_clicked()` (Листинг 1.5) таким образом, чтобы в случае ошибки подключения пользователь получал информацию о том, какая именно ошибка произошла.
3. Попробовать осуществить подключение к удалённому серверу используя следующие параметры строки подключения:

Имя сервера	msuniversity.ru
Имя базы данных	test_db
Логин	student
Пароль	student

2 ВЫВОД ДАННЫХ НА ЭКРАН

2.1 Модели и представления данных в среде программирования Qt Creator

Для полноценной работы с приложением, пользователю необходимо предоставить возможность просматривать содержимое таблиц базы данных, подключённой к приложению. В зависимости от используемого средства разработки для этого могут использоваться разные способы. Рассмотрим один из них.

Архитектура модель/представление

Для создания пользовательских интерфейсов часто применяется шаблон проектирования Model-View-Controller (MVC), который состоит из трёх типов объектов⁴:

- модель — объект приложения;
- представление — то, как объект будет представлен на экране;
- контроллер — определяет реакцию пользовательского интерфейса на ввод.

Если объединить представление и контроллер в один элемент, то получим архитектуру *модель-представление*, которая используется в Qt для работы различными источниками данных (Рисунок 2.1).

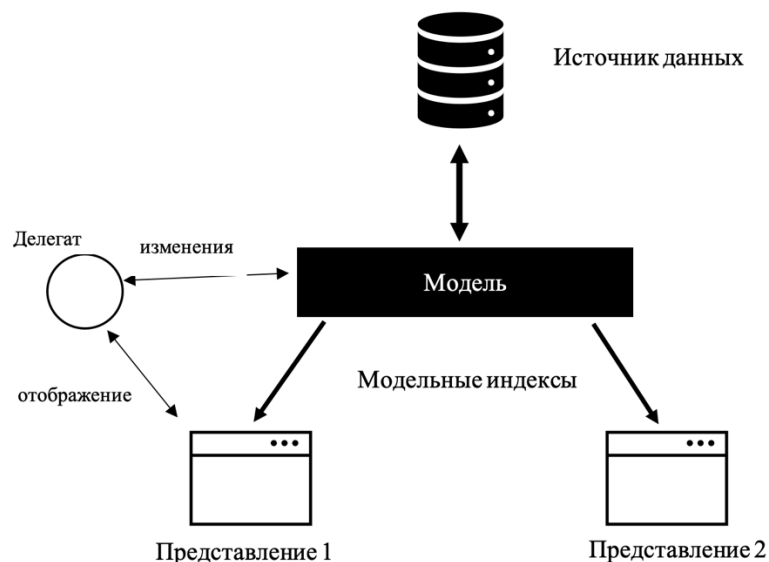


Рисунок 2.1 — Архитектура модель-представление

В данной архитектуре появляется новый объект — делегат, преимущество которого в том, что он даёт возможность для настройки представления и

⁴ Design Patterns – Elements of reusable Object-Oriented Software /Gamma, Helm, Johnson, Vlissides/ ISBN 0-201-63361-2

редактирования элементов данных, обеспечивая гибкость управления пользовательским вводом.

Принцип работы данной архитектуры следующий:

Модель осуществляет соединение с источником данных, формируя интерфейс, необходимый другим объектам архитектуры. Этот интерфейс представляет собой модельные индексы, которые являются ссылкой на данные. С помощью модельных индексов представление может получать данные из источника передавая соответствующие индексы в модели. Делегат в стандартных представлениях, отображает элементы данных, а в случае редактируемого элемента — связывается с моделью непосредственно, используя модельные индексы.

В данном пособии мы рассмотрим два класса, реализующих модели в Qt, это: *QSqlTableModel* и *QSqlQueryModel*.

Модель *QSqlTableModel* — модель, оптимизированная для работы с одной таблицей, но предоставляет возможности выполнения операций чтения-записи, которые можно производить из некоторых компонентов пользовательского интерфейса, например *tableView*.

Теоретически можно использовать эту модель для работы с несколькими таблицами, но удобнее и правильнее делать это с помощью модели *QSqlQueryModel*.

Модель *QSqlQueryModel* — модель позволяющая выбирать данные из базы данных при помощи инструкции «*SELECT*», в котором могут быть использованы практически все предложения, поддерживаемые данной инструкцией (условия, группировка, сортировка и т.д.). Данная модель используется преимущественно при необходимости извлечь данные из нескольких таблиц. Данная модель не предоставляет возможности записи, но она может быть реализована в потомке при изменении соответствующих свойств.

Рассмотрим применение этих классов для вывода данных из таблицы базы на экранную форму используя объект класса-представления *tableView*.

2.2 Отображение содержимого таблицы БД в приложении.

Открытие проекта

Перед тем, как приступить к выполнению задания необходимо открыть проект, созданный в процессе выполнения первой лабораторной работы.

Для открытия проекта необходимо:

1. Запустить Qt Creator

2. В окне приветствия на вкладке «Проекты» нажать на кнопку «Открыть» (Рисунок 2.2)

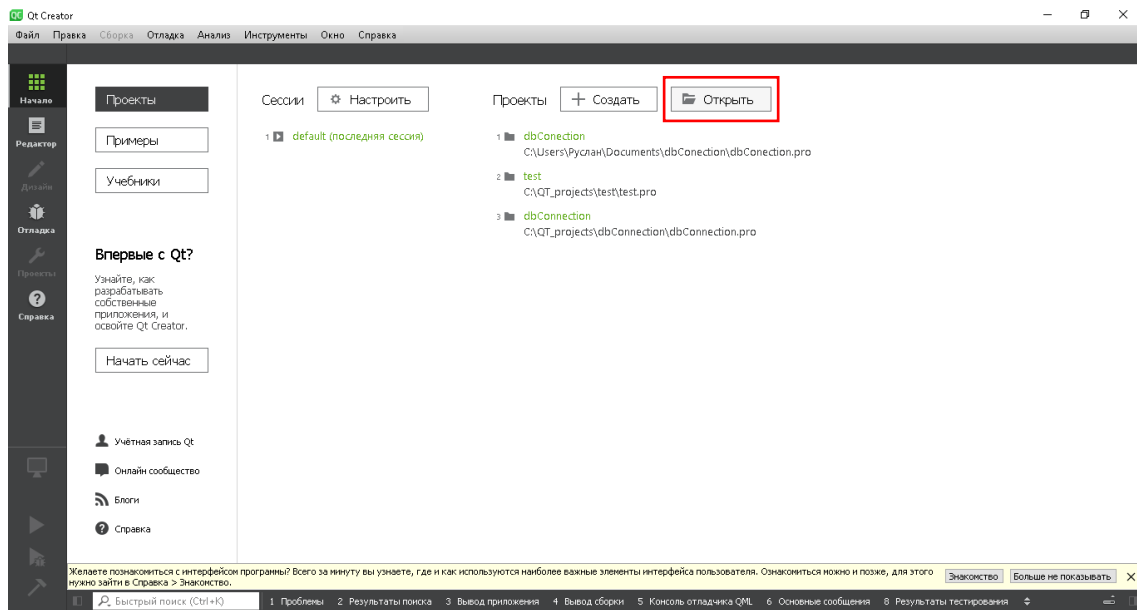


Рисунок 2.2 - Окно приветствия

3. В открывшемся диалоговом окне найти папку с программой и выбрать в ней файл с расширением .pro (Рисунок 2.3) и нажать кнопку «Открыть».

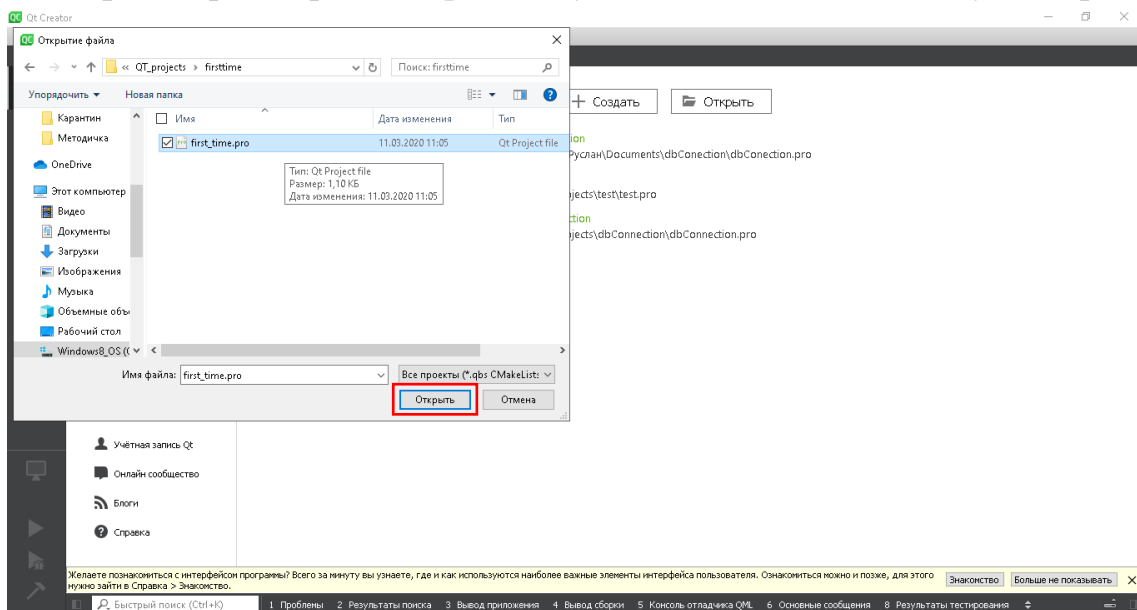


Рисунок 2.3 - Выбор файла проекта

4. В случае успеха откроется окно для работы с проектом (Рисунок 2.4)

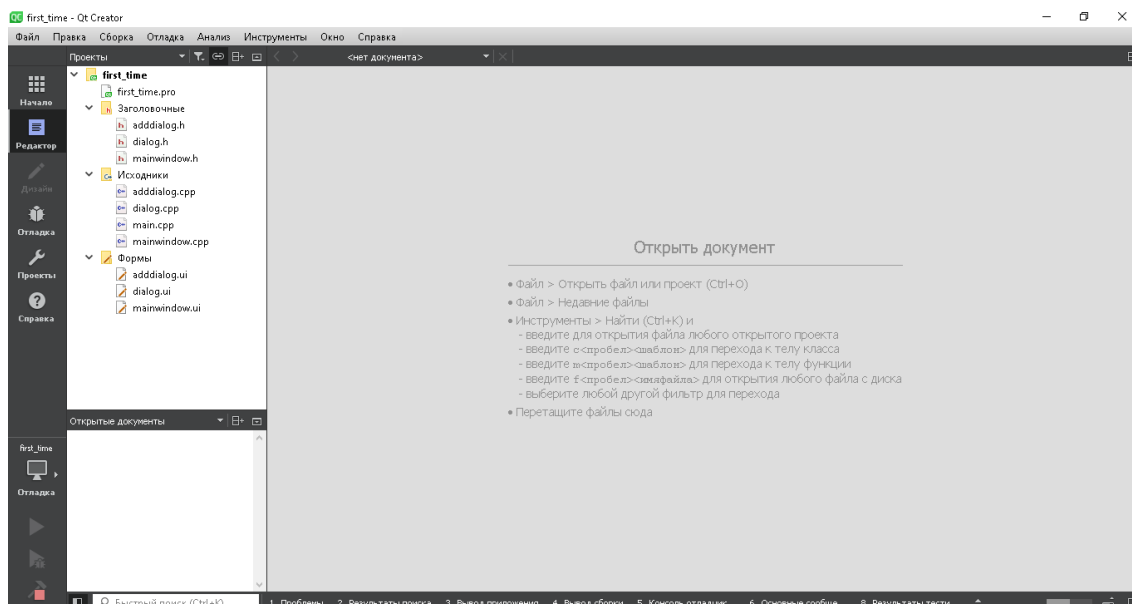


Рисунок 2.4 - Открытый проект



При открытии проекта возможно появление следующих проблем:

1. Предупреждение о несовместимости версий Qt. В большинстве случаев, если Вы используете более новую версию, проект все равно будет открыт и будет работать.
2. При размещении папки с проектом необходимо следить за тем, чтобы путь к ней не содержал букв кириллицы.

После открытия проекта можно приступить к выполнению задания второй лабораторной работы.

Класс *QSqlTableModel*

Объявим переменную класса *QSqlTableModel* в заголовочном файле главного окна (Листинг 2.1 **Ошибка! Источник ссылки не найден.**):

Листинг 2.1 - Изменения в заголовочном файле `mainwindow.h`

```
#ifndef MAINWINDOW_H
#define MAINWINDOW_H

#include <QMainWindow>
#include "connectiondialog.h"
#include "ui_connectiondialog.h"
#include <QSqlTableModel>

QT_BEGIN_NAMESPACE
namespace Ui { class MainWindow; }
QT_END_NAMESPACE
```

Листинг 2.1 (Окончание)

```
class MainWindow : public QMainWindow
{
    Q_OBJECT

public:
    MainWindow(QWidget *parent = nullptr);
    ~MainWindow();

private slots:

    void on_action_triggered();

private:
    Ui::MainWindow *ui;
    ConnectionDialog *dlg;
    QSqlTableModel *tmodel;
};
#endif // MAINWINDOW_H
```

Как уже упоминалось ранее, для представления данных пользователю, применяются специальные классы представлений, одним из которых является *QTableView*, который позволяет отображать таблицы. Размещаем компонент на форме. Для того чтобы данные были загружены в приложение необходимо добавить кнопку «Обновить», при каждом нажатии на которую будет происходить загрузка данных из БД и передача её в объект *TableView* (Рисунок 2.5).

Qt mainwindow.ui @ dbConection - Qt Creator

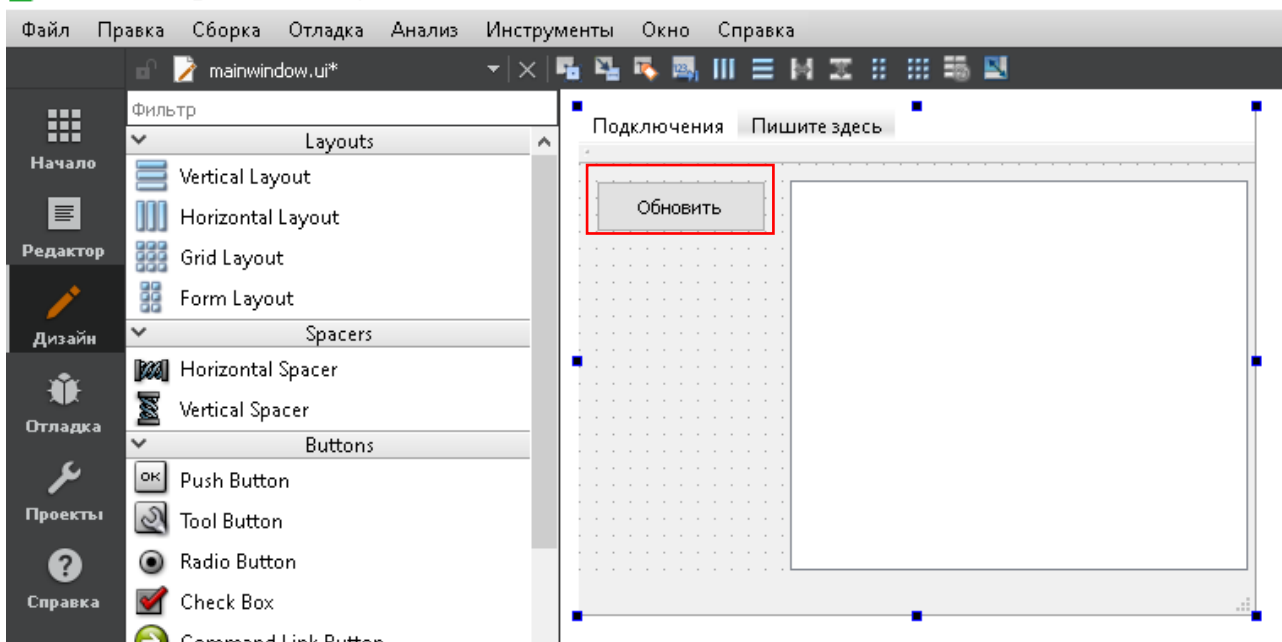


Рисунок 2.5 - Добавление компонентов на главную форму приложения

Далее нужно создать метод обработки щелчка по кнопке «Обновить» (см. Лабораторную работу №1, Рисунок 1.12). В теле метода написать следующий код (Листинг 2.2):

Листинг 2.2 - Обработчик нажатия на кнопку "Обновить"

```
void MainWindow::on_pushButton_clicked()
{
    tmodel = new QSqlTableModel();
    tmodel->setTable("product");
    tmodel->select();
    ui->tableView->setModel(tmodel);
}
```

Пробный запуск.

После успешного запуска приложения и установления подключения к БД, необходимо нажать на кнопку «Обновить», если всё сделано верно – в *tableView* будет отображено содержимое таблицы *product*.

Двойным щелчком по ячейки таблицы можно внести изменения в значения. Ознакомившись с возможностями данного класса, закройте приложение.

Рассмотренная модель позволяет достаточно легко получить доступ к данным, но в ряде случаев её применение нецелесообразно: например, при необходимости работы с несколькими таблицами составляющих один набор. Для этих целей применяется объект другого класса, который будут рассмотрен дальше.

Класс *QSqlQueryModel*

Далее при выполнении последующих ЛР будет использоваться именно *QSqlQueryModel*, следовательно, необходимо внести изменения в заголовочный файл и исходный код главного окна, чтобы исключить объект класса *QSqlTableModel* и все связанные с ним строки (подключённая библиотека, объявленная переменная и т.д.). На листингах ниже удаляемые строки зачёркнуты, а добавленные выделены полужирным (Листинг 2.3 - Листинг 2.4):

```

#ifndef MAINWINDOW_H
#define MAINWINDOW_H

#include <QMainWindow>
#include "connectiondialog.h"
#include "ui_connectiondialog.h"
#include <QSqlTableModel>
#include <QSqlQueryModel>
.....
.....
.....
private:
    Ui::MainWindow *ui;
    ConnectionDialog *dlg;
    QSqlTableModel *tmodel;
    QSqlQueryModel *qmodel;
};

```

Листинг 2.4 - Изменения в исходном коде главного окна

```

void MainWindow::on_pushButton_clicked()
{
    qmodel = new QSqlTableModel();
    qmodel->setQuery("SELECT * FROM product");
    ui->tableView->setModel(qmodel);
}

```

Пробный запуск.

После внесения изменений можно запустить проект на выполнение. Подключившись к БД необходимо нажать на кнопку «Обновить» и сравнить результат с предыдущим запуском, попробовать внести изменения в данные и сделать выводы об отличиях двух рассмотренных моделей.

Контрольные вопросы

1. Какой шаблон проектирования лежит в основе архитектуры модель/представление?
2. Из каких объектов она состоит?
3. В чем отличие архитектуры модель/представление от шаблона MVC?
4. Найти в документации Qt способ создания модели для выборки данных из нескольких таблиц с возможностью редактирования?
5. Какие ещё классы моделей поддерживаются Qt?

Дополнительное задание

1. Реализовать вызов функции обновления таблицы через главное меню приложения.
2. Внести изменения в свойства объекта tableView которые спрячут от пользователя ключевой столбец таблицы product (при этом, его нельзя исключать его из запроса).
3. Добавить возможность фильтрации строк таблицы по одному признаку. Признак студент выбирает самостоятельно. Результат работы фильтра проверить на таблице, содержащей не менее 15 записей.

3 ФУНКЦИИ ДОБАВЛЕНИЯ, ИЗМЕНЕНИЯ И УДАЛЕНИЯ ЗАПИСЕЙ

3.1 Обработка запросов к базе данных из приложения

Ранее были рассмотрены классы, которые позволяли подготавливать данные для представления их на экранных формах, однако, некоторые действия с БД не требуют отображения результата выполнения на экране, например инструкции *DDL* и *DML*, следовательно для их использования рассмотренные ранее классы не подходят.

Для работы со всеми инструкциями указанных языков необходимо использовать класс *QSqlQuery*. Этот класс представляет собой интерфейс для выполнения *SQL*-запросов, а также навигации по результирующей выборке. При работе с *QSqlQuery* возможно два сценария:

- 1) выполнение запроса без подготовки;
- 2) выполнение заранее подготовленные запросы.

Выполнение запроса без подготовки следует применять в запросах выполнение которых не зависит от изменения каких-либо параметров в приложении. Текст такого запроса передаётся в приложение в качестве параметра метода *exec()*.

Листинг 3.1 — Пример использования метода *exec()*

```
QSqlQuery *query = new QSqlQuery();  
query->exec("SELECT * FROM table");
```

Если необходимо выполнить запрос, в который предварительно необходимо передать один или несколько параметров, то можно воспользоваться вторым сценарием: выполнение заранее подготовленного запроса. В этом случае текст запроса необходимо передать в метод *prepare()*, но вместо конкретных значений параметров необходимо указывать специальные переменные вида `"<название переменной>"`, а после связать указанные переменные с необходимыми элементами интерфейса или переменными с помощью метода *bindValue()*. После того, как запрос сформулирован, а все входящие в него переменные определены, следует вызвать метод *exec()* без параметров, чтобы выполнить запрос, например:

Листинг 3.2 — Пример использования *prepare()*

```
QSqlQuery *query = new QSqlQuery();  
query->prepare("DELETE FROM table WHERE ID = :ID");  
query->bindValue(":ID", ui->lineEdit->text());  
query->exec();
```

С помощью объекта класса *QSqlQuery* можно извлекать данные из БД и передавать их в элементы интерфейса или переменные. Извлечение данных происходит с помощью инструкции *SELECT*. В результате выполнения инструкции *SELECT* и до выполнения следующего запроса, объект класса *QSqlQuery* содержит результирующую выборку (таблицу) к каждой строке которой можно обратиться, если указатель указывает на неё. Указателем можно управлять при помощи следующих методов:

- *next()* — указатель переходит к следующей записи;
- *previous()* — переход к предыдущей записи;
- *first()* — переход к первой записи;
- *last()* — переход к последней записи.

Изначально указатель показывает перед первой записью, и чтобы прочитать первую строку, указатель необходимо передвинуть на первую запись с помощью *next()*. Для обращения к конкретной ячейке используется метод *value(int)* куда необходимо передать индекс поля, которое необходимо прочитать. Индексы присваиваются от 0 каждому полю в списке выбора слева на право. Например в запросе ниже () поле *col_a* будет иметь индекс 0, а поле *col_c* — 3. Передадим значение поля *col_c* элементу пользовательского интерфейса:

Листинг 3.3 — Пример получения значение конкретные поля результирующей выборки

```
query->exec("SELECT col_a, col_b, col_c FROM table");  
ui->lineEdit->setText(query->value(3).toString());
```

Обратите внимание, что необходимо все значения явно приводить к нужному типу данных.

Рассмотрим применение объекта *QSqlQuery* на конкретных примерах.

3.2 Добавление новой строки в таблицу БД

В рассматриваемом случае добавление данных будет осуществляться в единственную таблицу, естественно, что при работе с другой таблицей или таблицами ввод данных необходимо будет организовать иначе.

Для начала определим количество элементов (переменных) в водимой строке, в примере таблица содержит три столбца (автоинкрементный *ID*, текстовый *Name*, текстовый *Category*). Так как автоинкрементные поля заполняются СУБД автоматически при создании новой строки, то для ввода значений новой строки достаточно двух полей. Разместить эти поля можно и на главной форме, но сейчас мы создадим новое диалоговое окно для ввода данных, переход к которому будет осуществляться нажатием на кнопку. Разместите на главной форме *PushButton* «Добавить...» (Рисунок 3.1).

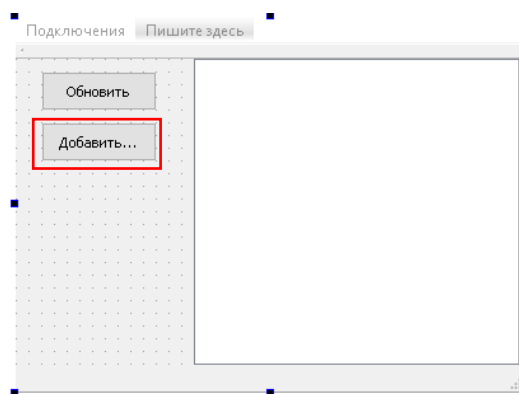


Рисунок 3.1 - Изменения на главной форме

Процесс создания формы для добавления строки аналогичен описанному ранее в ЛР№1 при создании окна подключения (см. стр. 11). Назовём новый класс “AddDialog”. В окне дизайнера формы добавим необходимые компоненты: два поля ввода и обычную кнопку. Так как значения в ячейке имеет ограничение в 40 символов (задано при создании таблицы, см. Листинг 1.1), в качестве полей ввода можно использовать объекты класса *QLineEdit*, чтобы пользователю было понятно их назначение, подпишем их используя объекты класса *QLabel*. После размещения компонентов, их можно сгруппировать по вертикали, тогда форма приобретет аккуратный вид (Рисунок 3.2).



Рисунок 3.2 - Макет формы добавления строки

Опишем реакцию приложения на нажатие на кнопку «Добавить». При нажатии на кнопку должен выполняться запрос на вставку введенных пользователем данных в таблицу БД. Для этого нам потребуется использовать объект класса *QSqlQuery*. Этот класс уже был упомянут при знакомстве с моделями для представления данных в предыдущей лабораторной работе (см. стр. 29). Данный класс позволяет выполнять SQL-запросы непосредственно из приложения. Для того чтобы его можно было использовать необходимо подключить соответствующую библиотеку. Подключение библиотек не обязательно производить в заголовочном файле, в этот раз, библиотека будет подключена прямо в исходном файле *adddialog.cpp*. Кликом правой кнопкой

мыши по кнопке «Добавить» переходим к слоту *clicked()* и перед тем, как запишем в него необходимый код, подключим новую библиотеку (Листинг 3.4):

Листинг 3.4 - Фрагмент файла *adddialog.cpp*

```
#include "adddialog.h"
#include "ui_adddialog.h"
#include <QSqlQuery>
#include <QMessageBox>

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

После, возвращаемся к заготовке метода *on_pushButton_clicked()* (в конце этого файла) и записываем следующий код (Листинг 3.5):

Листинг 3.5 - Метод, добавляющий новую строку в таблицу

```
Для/mainwindow
#include <QSqlQueryModel>
#include "adddialog.h"
#include "ui_adddialog.h"

private:
    Ui::MainWindow *ui;
    connectiondialog *dlg;
    AddDialog *add;
    QSqlQueryModel *qmodel;

void MainWindow::on_pushButton_2_clicked()
{
    add = new AddDialog();
    add->show();
}

{
    QSqlQuery *query = new QSqlQuery();
    query->prepare("INSERT INTO product (name, cat_ID)
VALUES (:name, :cat_ID)");
    query->bindValue(":name", ui->lineEdit->text());
    query->bindValue(":cat_ID", ui->lineEdit_2->text());
    QMessageBox *mess= new QMessageBox();
    if(!query->exec())
    {
        mess->setText("Запрос составлен неверно!");
        mess->show();
    }
}

void AddDialog::on_pushButton_clicked()
```

```

{
    QSqlQuery *query = new QSqlQuery();

    query->prepare("INSERT INTO product VALUES"
                  ":name,:category");
    query->bindValue(":name", ui->lineEdit->text());
    query->bindValue(":category", ui->lineEdit_2->text());

    query->exec();

    close(); //закрытие диалогового окна
}

```

Следующим шагом нужно создать обработчик нажатия на кнопку «Добавить...» расположенную на главной форме, аналогичный обработчик был написан для перехода к экранной форме подключения к БД (см. лабораторную работу №1 стр. 18).

После выполнения указанных действий, выполнить тестовый запуск программы и проверить правильность добавления строки. Чтобы внесённые изменения отразились в TableView важно не забывать нажимать на кнопку «Обновить».

Внесение изменений и удаление строки таблицы

Пользователям периодически нужна функция редактирования данных. В этой части лабораторной работы эта функция будет добавлена в приложение.

Для начала добавим необходимые компоненты на форму (компоненты, необходимые для редактирования записей будут размещены на главной форме). В отличии от ввода данных, при внесении изменений или удаления записи из БД, пользователю необходимо знать и передать в соответствующий запрос её первичный ключ, поэтому, в рассматриваемом примере, на форму следует поместить три объекта класса QLineEdit и три метки класса QLabel, чтобы пояснить назначение полей ввода. Также нужно добавить две кнопки: «Изменить» и «Удалить». В результате изменений главная форма примет вид⁵ (Рисунок 3.3):

⁵ Красные прямоугольники отображают сгруппированные объекты. Группировка не является обязательной, но её использование визуально может сделать проект аккуратнее и понятнее. Группировка уже упоминалась в ЛР1 при добавлении новой экранной формы (см. стр. 7)

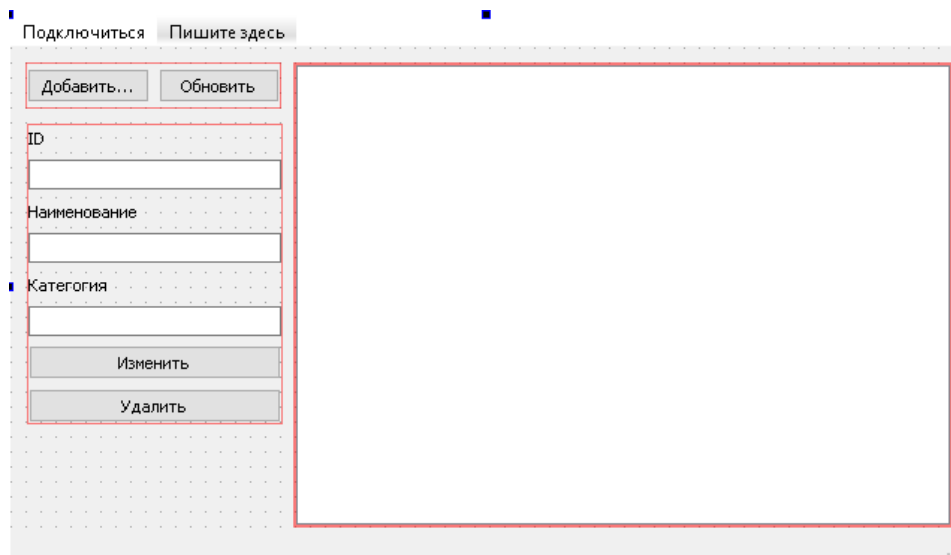


Рисунок 3.3 - Главная форма после добавления компонентов

После добавления новых объектов, необходимо изменить свойства некоторых объектов. Во-первых – поле ID необходимо сделать доступным только для чтения, так как внесение изменений в первичный ключ может повлечь нарушение целостности БД. Во-вторых – так как модель *QSqlQueryModel* не поддерживает изменение данных через объект *tableView*, а действия с элементами строки будет производиться при помощи других компонентов, значит выбор отдельных ячеек пользователю не нужен. В данном случае более подходящим способом выбора данных в таблице будет выбор всей строки сразу.

Изменение свойств объектов возможно либо при помощи программного кода, либо через «Редактор свойств» в правой части экрана (Рисунок 1.11, под цифрой 3). В данном примере изменим свойства объекта при помощи редактора свойств.

Изменение свойств *lineEdit* (который будет отвечать за соответствие ID):

- 1) Выделите объект щёлкнув по нему левой кнопкой мыши.
- 2) В редакторе свойств найдите “*ReadOnly*”
- 3) Измените значение на *true*.

Как обсуждалось ранее, нужно оставить пользователю возможность выбора только одной строки и запретить выбирать отдельные ячейки.

Изменение свойств *tableView*:

- 1) Выделить объект, нажав левой кнопкой мыши.
- 2) В редакторе свойств установить значение *SingleSelection* свойству *SelectionMode*.
- 3) Установить свойству *SelectionBehavior* значение *SelectRows*.

Осуществим пробный запуск приложения и убедимся в невозможности внесения данных в поле ID и в выборе отдельной ячейки в таблице.

Для реализации функций изменения и удаления необходимо получить первичный ключ записи, которую выделил пользователь. Для этого, щелчком правой кнопкой мыши на объекте *TableView* и в контекстном меню выберем пункт «Перейти к слоту...». В открывшемся диалоговом окне нужно выбрать слот *clicked()*. Этот метод будет вызываться каждый раз, когда пользователь будет наводить курсор на *TableView* и нажимать левую кнопку мыши.

В созданном шаблоне запишем следующий код (Листинг 3.6):

Листинг 3.6 - Метод обработки нажатия по объекту *TableView*

```
void MainWindow::on_tableView_clicked(const QModelIndex &index)
{
    int temp_ID;
    temp_ID = ui->tableView->model()->data(ui->tableView->
model()->index(index.row(),0)).toInt(); //в одну строку
    QSqlQuery *query = new QSqlQuery();
    query->prepare("SELECT name, category FROM product WHERE ID =
:ID");
    query->bindValue(":ID",temp_ID);

    ui->lineEdit->setText(QString::number(temp_ID));
    if (query->exec())
    {
        query->next();
        ui->lineEdit_2->setText(query->value(0).toString());
        ui->lineEdit_3->setText(query->value(1).toString());
    }
}
```

В данном методе переменная *tempID* получает значение, которое хранится в первой ячейке строки (а это поле ID, в котором и хранится значение первичного ключа).

Зная первичный ключ, можно создать запрос, который вернёт нужную строчку. Для этого воспользуемся уже знакомым *QSqlQuery*. Объявлять эту библиотеку в этом модуле или в заголовочном файле не требуется, он уже объявлен в файле *addialog.cpp*, с который подключён к этому модулю. Нужно сформировать запрос на выборку двух оставшихся полей из таблицы, отфильтрованных по значению первичного ключа.

Для того чтобы присвоить полученные значения в поле *Text* объекта *lineEdit* нужно воспользоваться методом *value()*, где в скобках указывается

порядковый номер поля (начиная с нуля) в том порядке в котором они указаны в запросе. Данное свойство не будет работать пока мы не сместим курсор на первую строку выборки при помощи метода *next()*.

Для проверки, можно запустить приложение и выбирая строки, контролировать изменение значений в объектах *QLineEdit*.

3.3 Удаление выбранной записи

Напишем обработчик для кнопки «Удалить» (см. создание обработчика нажатия на кнопку в лабораторной работе №1 стр. 11).

Листинг 3.7 - Обработчик нажатия на кнопку "Удалить"

```
void MainWindow::on_pushButton_4_clicked()
{
    QSqlQuery *query = new QSqlQuery();
    query->prepare("DELETE FROM product WHERE ID = :ID");
    query->bindValue(":ID", ui->lineEdit->text());
    query->exec();
    //очищаем поля ввода
    ui->lineEdit->setText("");
    ui->lineEdit_2->setText("");
    ui->lineEdit_3->setText("");
    //вызов обработчика кнопки «Обновить»
    MainWindow::on_pushButton_clicked();
}
```

3.4 Изменение выбранной записи

Для передачи обновленных значений в базу данных, опишем обработчик нажатия на кнопку «Изменить» (Листинг 3.8):

Листинг 3.8 - Обработчик нажатия на кнопку "Изменить"

```
void MainWindow::on_pushButton_3_clicked()
{
    QSqlQuery *query = new QSqlQuery();
    query->prepare("UPDATE product SET name = :name, "
                  "category = :category WHERE ID = :ID");
    query->bindValue(":ID", ui->lineEdit->text());
    query->bindValue(":name", ui->lineEdit_2->text());
    query->bindValue(":category", ui->lineEdit_3->text());
    query->bindValue(":image", Img);
    query->exec();
    ui->lineEdit->setText("");
    ui->lineEdit_2->setText("");
    ui->lineEdit_3->setText("");
}
```



```

    ui->label_4->setText("");
    MainWindow::on_pushButton_clicked();
}

```

3.5 Доступ к функциям изменения и удаления строки таблицы через контекстное меню.

Реализуем доступ к функциям изменения и удаления записей таблицы через контекстное меню, вызываемое нажатием правой кнопкой мыши по *TableView*. Этот способ более сложен, но изучив принцип его реализации можно существенно упростить интерфейс разрабатываемых приложений, распределить функции по объектам, к которым они относятся и т.д.

Рассмотрим процесс создания контекстного меню пошагово:

- 1) Необходимо разрешить вызов контекстного меню для объекта *tableView* (Рисунок 3.4):

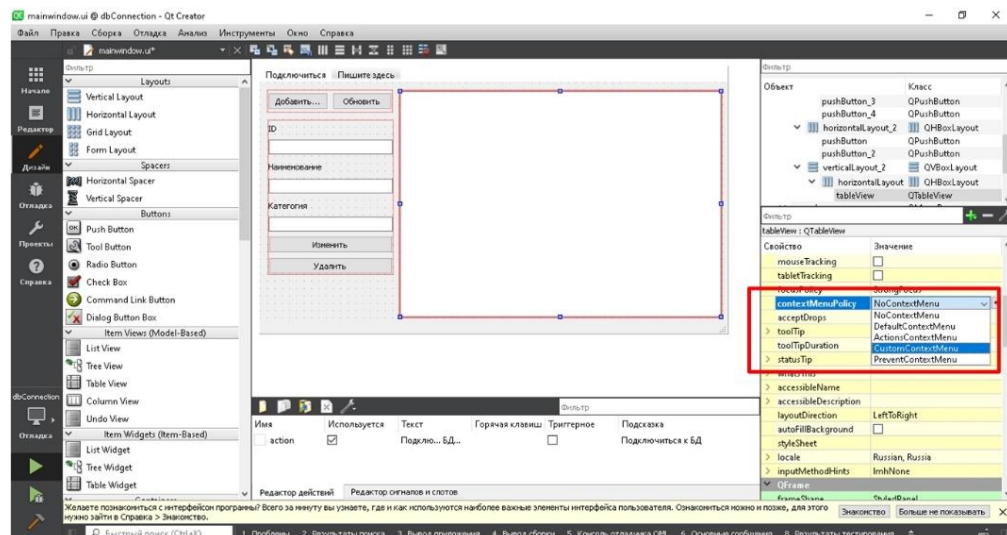


Рисунок 3.4 – Добавление возможности использования контекстного меню

- 2) Введем переменную – флаг состояния, который будет иметь значение *0* если объект *TableView* не отображает данные и *1* если данные отображаются. Переменную объявим в заголовочном файле *mainwindow.h*, это позволит обращаться к ней из любого метода, описанного в *mainwindow.cpp*.
- 3) Введём глобальную переменную, куда будет помещаться ID выбранной записи. Для этого, в заголовочном файле, в разделе *private* объявим переменную *GlobID* целочисленного типа.
- 4) Для обработки генерации контекстного меню создадим слот самостоятельно, для чего в файле *mainwindow.cpp* напишем следующий фрагмент кода (Листинг 3.9):

Листинг 3.9 - Пользовательский метод обработки создания контекстного меню

```

void MainWindow::CustomMenuReq(QPoint pos)

```

```

{
    if (fl == 1)
    {
        QModelIndex index = ui->tableView->indexAt(pos);
        GlobID = ui->tableView->model()->data(ui->tableView-
>model()->index(index.row(),0)).toInt();
        //Создаём меню и два действия
        QMenu *menu = new QMenu(this);
        QAction *ModRec = new QAction("Изменить...", this);
        QAction *DelRec = new QAction("Удалить", this);
        //Соединяем действие с соответствующим сигналом и слотом

        //(который нужно создать позже)
        connect(ModRec, SIGNAL(triggered()), this, SLOT(ModRecAction()));
        connect(DelRec, SIGNAL(triggered()), this, SLOT(DelRecAction()));

        //Добавление действий, созданных ранее
        menu->addAction(ModRec);
        menu->addAction(DelRec);

        menu->popup(ui->tableView->viewport()->mapToGlobal(pos));
    }
}

```

5) Опишем слот *DelRecAction()* (Листинг 3.10):

Листинг 3.10 – Слот для пункта контекстного меню "Удалить"

```

void MainWindow::DelRecAction()
{
    QSqlQuery *query = new QSqlQuery();
    query->prepare("DELETE FROM product WHERE ID = :ID");
}

```

Листинг 3.10 (Окончание)

```

query->bindValue(":ID",GlobID);
query->exec();
MainWindow::on_pushButton_clicked();

```

- 6) На примере обработчика нажатия на кнопку «Изменить...» рассмотрим процесс передачи значений переменных между модулями.

Для этого добавим новое диалоговое окно *ModfyDialog* и расположим на нем два объекта класса *QLineEdit*, два – класса *QLabel* и один *pushButton* (Рисунок 3.5):

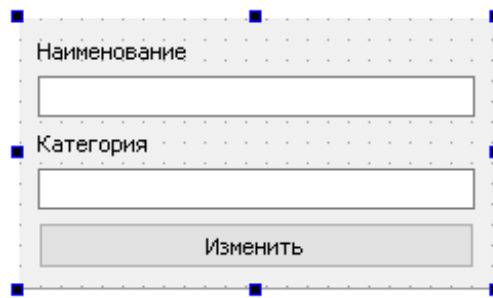


Рисунок 3.5 - Новое диалоговое окно

В этом окне будут отображаться данные, выбранные в таблице на главной форме. При изменении значений в *lineEdit* и нажатии на кнопку «Изменить» данные в выбранной строке БД также будут заменены.

Объявим глобальную целочисленную переменную *TempID* в которую будет помещаться перенесённый из *mainwindow.cpp* идентификатор записи, после чего опишем обработчик нажатия на кнопку «Изменить» (Листинг 3.11), который аналогичен описанному в ЛР3:

Листинг 3.11 - Обработчик нажатия на кнопку "Изменить"

```
void ModifyDialog::on_pushButton_clicked()
{
    QSqlQuery *query = new QSqlQuery();
    query->prepare("UPDATE product SET name = :name, "
                  "category = :category, WHERE ID = :ID");
    query->bindValue(":ID", tempID);
    query->bindValue(":name", ui->lineEdit->text());
    query->bindValue(":category", ui->lineEdit_2->text());
    if (query->exec())
    {
        close();
    }
}
```

- 7) Ранее мы создавали слоты при помощи контекстного меню, через команду «Перейти к слоту...», которая не только создавала шаблон будущего метода, но и вносила соответствующие изменения в заголовочный файл. Слоты *CustomMenuReq(QPoint)*, *ModRecAction()* и *DelRecAction()*, упомянутые выше (Листинг 3.9, Листинг 3.10) созданы вручную, а значит изменения в заголовочный файл следует также внести вручную, иначе они не будут являться частью класса *MainWindow*, что сделает невозможным их использование в качестве слотов. Поэтому, следующим шагом необходимо включить в заголовочный файл упомянутые выше слоты, подключить

модули новой экранной формы, а также новую переменную, необходимую для вызова этой формы (Листинг 3.12):

Листинг 3.12 – Заголовочный файл `mainwindow.h` (вносимые изменения выделены)

```
#ifndef MAINWINDOW_H
#define MAINWINDOW_H

#include <QMainWindow>
#include "connectiondialog.h"
#include "ui_connectiondialog.h"
#include "adddialog.h"
#include "ui_adddialog.h"
#include <QSqlQueryModel>
#include "modifydialog.h"
#include "ui_modifydialog.h"

QT_BEGIN_NAMESPACE
namespace Ui { class MainWindow; }
QT_END_NAMESPACE

class MainWindow : public QMainWindow
{
    Q_OBJECT

public:
    MainWindow(QWidget *parent = nullptr);
    ~MainWindow();

    int GlobID;
    int fl;
```

Листинг 3.12 (Окончание)

```
private slots:
    void on_pushButton_clicked();
    void on_pushButton_2_clicked();

    void on_action_triggered();

    void on_tableView_clicked(const QModelIndex &index);

    void on_pushButton_3_clicked();

    void on_pushButton_4_clicked();

    void CustomMenuReq(QPoint);

    void DelRecAction();

    void ModRecAction();

private:
    Ui::MainWindow *ui;
    ConnectionDialog *dlg;
    AddDialog *adlg;
    ModifyDialog *mdlg;
    QSqlQueryModel *qmodel;
};
#endif // MAINWINDOW_H
```

- 8) Для того, чтобы созданные слоты срабатывали в нужный момент, их необходимо соединить с нужным нам сигналом при помощи процедуры *connect*. Так как слот *CustomMenuReq(QPoint)*, может быть использован сразу же после запуска приложения, то можно внести изменения прямо в конструктор главной формы (Листинг 3.13):

Листинг 3.13 - Внесение изменений в конструктор главной формы

```
MainWindow::MainWindow(QWidget *parent)
    : QMainWindow(parent)
    , ui(new Ui::MainWindow)
{
    ui->setupUi(this);
    connect(ui->tableView,
            SIGNAL(customContextMenuRequested(QPoint)),
            SLOT(CustomMenuReq(QPoint)));
    fl = 0; //установка начального значения
}
```

Последняя строчка устанавливает *0* в качестве значения переменной *fl*, что сделает невозможным вызов контекстного меню *TableView* до того, как будет нажата кнопка «Обновить».

- 9) Чтобы меню можно было вызвать контекстное меню необходимо изменить значение переменной *fl* с *0* на *1*. Добавим соответствующую строчку в обработчик нажатия на кнопку «Обновить» (Листинг 3.14) и таким образом, обеспечим возможность вызова меню только в том случае, если в объекте отображаются данные:

Листинг 3.14 - Изменения в обработчике нажатия на кнопку "Обновить"

```
void MainWindow::on_pushButton_clicked()
{
    fl = 1;

    qmodel = new QSqlQueryModel();
    qmodel->setQuery("SELECT * FROM product");

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

- 10) После создания новой экранной формы и подключения его в заголовочном файле главной формы, можно приступить к написанию слота *ModRecAction()*, который свяжет сигнал *sendID(int)*, испускаемый объектом *tableView*, со слотом *sendingID(int)* объекта *mdlг*. Передаст в неё первичный ключ выбранной записи (при помощи процедуры *emit*) и выведет на экран форму модификации записи (Листинг 3.15,).

Листинг 3.15 - Описание слота *ModRecAction()*

```
void MainWindow::ModRecAction()
{
    mdlг = new ModifyDialog();
    connect(this, SIGNAL(sendID(int)), mdlг, SLOT(sendingID(int)));
    emit sendID(GlobID);
    mdlг->show();
    disconnect(this, SIGNAL(sendID(int)), mdlг,
SLOT(sendingID(int)));
}
```

В случае если связь между сигналом и слотом больше не требуется, то её можно разорвать при помощи процедуры *disconnect*, с теми же передаваемыми параметрами (Листинг 3.15, последняя строка).



Важно: сигнатура (*тип возвращаемого значения, количество и тип переменных*) испускаемого сигнала и принимающего его слота должны совпадать.

Реализацию работы сигнала *sendID(int)*, аналогично другим методам, описывать не нужно, но его обязательно необходимо объявить в классе, который может испускать этот сигнал, в разделе *signals*. В данном случае это необходимо сделать в *mainwindow.h* (Листинг 3.16). Если раздел отсутствует, то следует его добавить вручную.

Листинг 3.16 - Изменения в *mainwindow.h*

```
//Сигналы можно описывать сразу после описания слотов

private slots:

    /*строки пропущены*/

signals:
    void sendID(int);
```

- 11) Теперь необходимо добавить запись о слоте *sendingID* в заголовочный файл *modifydialog.h*, так как он должен принадлежать классу *ModifyDialog* (Листинг 3.17). В разделе *private slots* необходимо добавить следующую строку:

Листинг 3.17 - Изменения в заголовочном файле *modifydialog.h*

```
private slots:
    void on_pushButton_clicked();
    void sendingID(int);
```

- 12) В файле *modifydialog.cpp* необходимо прописать реализацию для слота *sendingID* (Листинг 3.18):

Листинг 3.18 - Добавление функции в *modifydialog.cpp*

```
void ModifyDialog::sendingID(int aa)
{
    tempID = aa;
    QSqlQuery *query = new QSqlQuery();
    query->prepare("SELECT name, category FROM product"
                  "WHERE ID = ?");
    query->bindValue(0, aa);
    if (query->exec())
    {
        query->next();
        ui->lineEdit->setText(query->value(0).toString());
        ui->lineEdit_2->setText(query->value(1).toString());
    }
}
```

- 13) После необходимо произвести тестовый запуск приложения и проверить работоспособность всех добавленных функций. После подключения к БД, выполните обновление данных и кликните правой кнопкой мыши по любой записи в таблице. Проверьте корректность работы функции изменения и удаления записи.

Контрольные вопросы

1. В чём главное отличие объекта QSqlQuery от QSqlQueryModel?
2. Какие инструкции можно выполнить при помощи QSqlQuery?
3. Напишите метод, который бы считывал данные из результирующей выборки, количество записей в которой заранее не известно.
4. Какие сценарии работы с QSqlQuery вы знаете?

Дополнительное задание

1. Реализовать вызов созданных функций с помощью горячих клавиш.
2. Используя SSMS добавить в базу данных две таблицы содержащих не менее 3 столбцов, один из которых является автоинкрементным первичным ключом. Таблицы должны отражать одну выбранную студентом предметную область. Примеры пар таблиц: «Студент» - «Группа», «Товар» - «Категория» и т.д.
3. Используя объект класса QComboBox реализовать возможность просмотра, удаления и изменения всех столбцов выбранной таблицы в главной форме приложения. Модель данных, имя таблицы, количество и тип столбцов выбирать самостоятельно.
4. Реализовать автоматическое обновление tableView при установлении подключения и возврату к главному окну.
5. Добавить контекстное меню для объекта QLineEdit, через которое пользователь сможет осуществлять очистку поля ввода.
6. Реализовать возможность выбора и правки любой таблицы БД на отдельной форме.

4 ВЫВОД ДАННЫХ ИЗ ПРИЛОЖЕНИЯ

Важной функцией информационных систем является функция предоставления отчётности. Требуемый отчёт может быть выведен на печать или экспортирован в другое приложение для выполнения с ним каких-либо действий.

Отчеты бывают простыми, влиять на формирование которых пользователь не может, и интерактивными, когда пользователь может влиять на формирование отчёта в процессе выполнения программы, например, указывая какие-либо фильтры.

В этой главе рассматриваются варианты создание простых отчётов в формате .doc и .pdf.

4.1 Экспорт таблицы в стороннее приложение

Перед началом выполнения необходимо создать пустой файл с расширением .html.

Создание новой экранной формы.

Создадим новую экранную форму, которую назовём *PrintDialog.*, назначение которого - выбор ранее созданного html-файла, в который будет экспортироваться таблица *product*.

На форме необходимо разместить на форме *lineEdit* для записи пути к файлу, *pushButton* для запуска процесса экспорта. Существенно повысит удобство использования приложения наличие стандартного диалогового окна открытия файла, с помощью которого можно избежать ввода пути к файлу в ручную, поэтому, для вызова диалогового окна открытия файла, добавим на форму кнопку класса *QToolButton*. Расположим элементы на форме и сгруппируем интерфейс по горизонтали (Рисунок 4.1).



Рисунок 4.1 - Форма экспорта таблицы в Word

Подключение библиотек для работы с файлами

В задачи созданного модуля будет входить перенос данных из таблицы БД в html-файл. Для решения этой задачи необходимы средства работы с файлами. Среди стандартных библиотек Qt есть ряд библиотек, специально созданных для работы с файлами: *QFile*, *QFileDialog* и т.д.

Для организации передачи содержимого таблицы в html-файл необходимо подключить к приложению следующие библиотеки:

- библиотека, содержащая необходимый класс для выполнения действия над файлами (*QFile*);
- библиотека, содержащая классы и методы для организации потокового вывода текста (*QStringDialog*).

Кроме того, для получения данных из БД, понадобится уже знакомая библиотека *QSqlQuery* а для упрощения выбора нужного файла – *QFileDialog*. Подключим нужные библиотеки в заголовочном файле *printdialog.h* (Листинг 4.1):

Листинг 4.1 - Библиотеки подключённые к *printdialog.h*

```
#include <QDialog>
#include <QSqlQuery> //создание запросов
#include <QFileDialog> //диалоговое окно работы с файлами
#include <QFile> //работа с файлами
#include <QTextStream> //организация текстовых потоков
```

Получение пути к файлу

Пользователь может указать путь к html-файлу вручную или выбрать файл в диалоговом окне открытия файла. Рассмотрим второй вариант. Создадим слот для *toolButton* и свяжем его с сигналом *clicked()*, в теле метода *on_toolButton_clicked()* необходимо написать следующую строку (Листинг 4.2):

Листинг 4.2 - Обработчик нажатия на кнопку «...»

```
void PrintDialog::on_toolButton_clicked()
{
    ui->lineEdit->setText(QFileDialog::getOpenFileName(0, "Выберете файл", ".\\", "*.html"));
}
```

При нажатии на *toolButton* откроется диалоговое окно выбора файла, а само приложение перейдёт в режим ожидания. После того как пользователь выберет нужный файл, строка, содержащая путь к нему, будет возвращена в приложение и помещена в объект *lineEdit*.

Организация вывода данных таблицы в html-файл

Создадим слот для кнопки «Сформировать» связанный с сигналом *clicked()* при выполнении которого будет установлена связь с файлом, после чего в него будут скопированы данные таблицы.

Алгоритм записи информации в файл состоит из трёх этапов: открытие файла, передача данных, закрытие файла. Для открытия файла необходимо:

- 1) Создать объект класса *QFile*.

- 2) Связать его с выбранным файлом на диске.
- 3) Открыть файл для записи.

Чтобы организовать передачу данных в файл воспользуемся объектом класса *QTextStream*, который позволяет осуществлять чтение и запись символьных строк в консоль (файлы). Для организации вывода нужно создать объект этого класса и связать с открытым файлом. Чтобы передать строку в консоль (файл), нужно написать оператор следующего вида (Листинг 4.3):

Листинг 4.3 - Формат записи оператора вывода строки

```
<Имя объекта> << "строка вывода";
```

Полный код метода *on_pushButton_clicked()* приведен ниже (Листинг 4.4):

Листинг 4.4 - Экспорт таблицы в HTML-файл

```
void PrintDialog::on_pushButton_clicked()
{
    QFile *file = new QFile();
    file->setFileName(ui->lineEdit->text());
    file->open(QIODevice::WriteOnly);

    QTextStream in(file);

    in << "<html><head></head><body><center>"
        +QString("Пример создания отчёта");
    in << "<table border=1><tr>";
    in << "<td>" +QString("ID") + "</td>";
    in << "<td>" +QString("Наименование") + "</td>";
    in << "<td>" +QString("Категория") + "</td></tr>";

    QSqlQuery *query = new QSqlQuery();
    query->exec("SELECT * FROM product");
    query->next();
    while (query->next())
    {
        in << "<tr>";
        in << "<td>" +QString(query->value(0).toString()) + "</td>";
        in << "<td>" +QString(query->value(1).toString()) + "</td>";
        in << "<td>" +QString(query->value(2).toString()) + "</td>";
        in << "</tr>";
    }

    in << "</table>";
    in << "</center></body></html>";

    file->close();
}
```

Настройка навигации между экранными формами

Настройка доступа к окну выполняется также, как и в предыдущих лабораторных работах.

Добавляем названий заголовочных файлов новой экранной формы и объявляем новую переменную в заголовочном файле `mainwindow.h` (Листинг 4.5):

Листинг 4.5 - Изменения в файле `mainwindow.h`

```
//в подключённых библиотеках
#include <QMainWindow>
#include "connectiondialog.h"
#include "ui_connectiondialog.h"
#include "adddialog.h"
#include "ui_adddialog.h"
#include <QSqlQueryModel>
#include "modifydialog.h"
#include "ui_modifydialog.h"
#include "printdialog.h"
#include "ui_printdialog.h"

//в объявленных переменных
private:
    Ui::MainWindow *ui;
    ConnectionDialog *dlg;
    AddDialog *adlg;
    ModifyDialog *mdlg;
PrintDialog *pdlg;
    QSqlQueryModel *qmodel;
```

На главную форму необходимо добавить кнопку (*pushButton*), при нажатии на которую пользователю будет показана форма *PrintDialog*. Для улучшения внешнего вида, сгруппируем добавленную кнопку и *tableView* по вертикали. Присвоим кнопке название «Экспорт в Word», дважды щелкнув левой кнопкой мыши на ней.

После добавления кнопки, главная форма будет выглядеть как показано на рисунке (Рисунок 4.2):

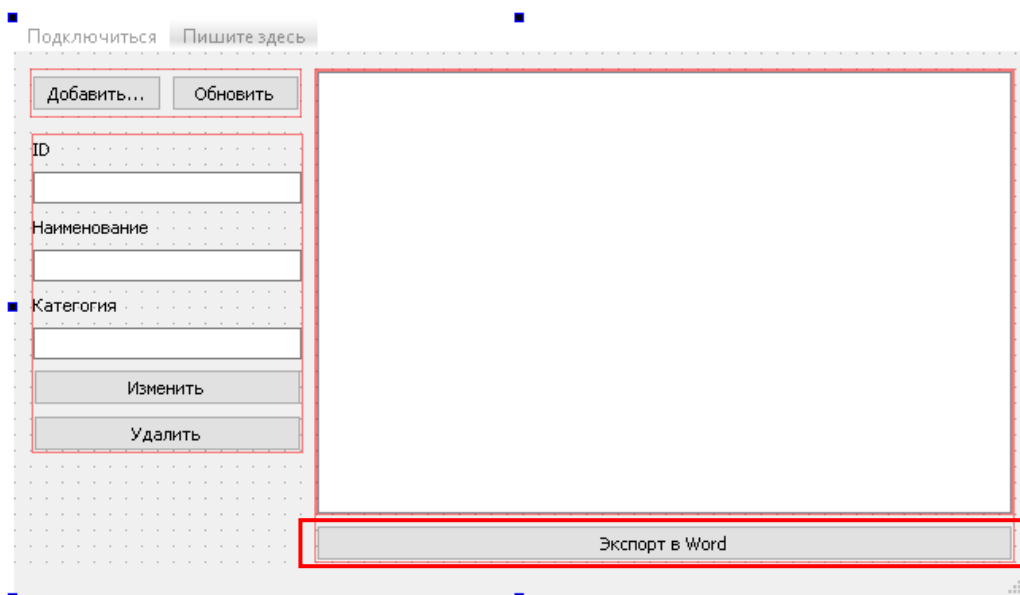


Рисунок 4.2 - Добавленная кнопка "Экспорт в Word"

Завершающим этапом для кнопки создадим слот `on_pushButton_5_clicked()`, где напишем следующий код (Листинг 4.6):

Листинг 4.6 - Слот `on_pushButton_5_clicked()` кнопки "Экспорт в Word"

```
void MainWindow::on_pushButton_5_clicked()
{
    pdlg = new PrintDialog();
    pdlg->show();
}
```

Пробный запуск.

Запустите приложение, сохранив все внесённые изменения. Убедитесь, что сборка проекта завершилась без ошибок.

Соедините приложение с БД, обновите данные и нажмите на кнопку «*Экспорт в Word*». В открывшемся диалоговом окне вводим путь к созданному в начале лабораторной работы файлу (или используем диалоговое окно открытия файла) и нажимаем кнопку «*Сформировать*». При описании реакции на нажатие этой кнопки, информирование пользователя об окончании процесса (или любого другого уведомления о ходе процесса) запрограммировано не было, поэтому чтобы проверить корректность экспорта данных, необходимо открыть файл самостоятельно при помощи любого браузера. Если всё отображается корректно, переходим к настройке экспорта html-файла в Word, иначе необходимо внимательно проверить метод `PrintDialog::on_pushButton_clicked()` (Листинг 4.4).

Подключение набора библиотек для работы с компонентами ActiveX

Microsoft Word использует ActiveX компоненты, благодаря которым, стороннее приложение может запустить приложение и открыть нужный документ. Если дать не строгое определение, то ActiveX – это фреймворк, позволяющий создавать программные компоненты, которые могут быть запущены программами, написанными на разных языках программирования.

Qt Creator поддерживает возможность работы с ActiveX при помощи фреймворка ActiveQt.



Поддержка Qt фреймворка ActiveX и COM, позволяет:

- ◆ получать доступ и использовать элементы управления ActiveX и объекты COM предоставляемые любым из ActiveX серверов в Qt приложении.
- ◆ делать доступными Qt приложения, как COM серверы.

Фреймворк ActiveQt состоит из двух модулей:

- 1) Модуль QAxContainer — это статическая библиотека, реализующая производные от QObject и QWidget классы QAxObject и QAxWidget, которые реализованы как контейнеры для COM объектов и элементов управления ActiveX.
- 2) Модуль QAxServer — это статическая библиотека, реализующая функциональность для внутривещественных (DLL) и исполняемых COM-серверов. Данный модуль предоставляет классы QAxAggregated, QAxBindable и QAxFactory.

Так как наша цель получить доступ к ActiveX объектам, а не создать таковой внутри приложения, то необходимо подключить модуль *axcontainer* в корневом файле проекта *dbconnection.pro* (Листинг 4.7), по аналогии с добавлением модуля *sql* (см. лабораторную работу №1, стр. 16), так как именно он содержит необходимые библиотеки:

Листинг 4.7 - Подключение модуля к проекту

```
QT += core gui sql axcontainer
```

Для того, чтобы среда программирования начала корректно определять библиотеки и классы входящие в этот модуль, необходимо пересобрать решение без запуска, нажав на изображение молотка в левом нижнем углу редактора кода (Рисунок 1.14). После окончания сборки можно приступить к подключению нужной библиотеки.

Переходим к заголовочному файлу *printdialog.h*. В нем необходимо подключить библиотеку *QAxObject* для работы с *ActiveX* и *COM* объектами (Листинг 4.8):

Листинг 4.8 - Подключение библиотеки для работы с ActiveX Object

```
#include <QDialog>
#include <QSqlQuery>
#include <QFileDialog>
#include <QFile>
#include <QTextStream>
#include <QAxObject>
```

Открытие документа в MS Word через приложение

Возвращаемся к обработчику нажатия на кнопку «Сформировать», описанного в файле *printdialog.cpp* и в конце метода добавляем следующие строки (Листинг 4.9):

Листинг 4.9 - Создание ActiveX объектов

```
QAxObject *word = new QAxObject("Word.Application", this);
word->setProperty("DisplayAlerts", false);
word->setProperty("Visible", true);
QAxObject *doc = word->querySubObject("Documents");
doc->dynamicCall("Open(QVariant)", "C:\\\\QT_projects\\\\new.html");
```

Тестовый запуск

Запускаем приложение и проверяем его работоспособность как описано ранее, но в этот раз, после нажатия на кнопку будет запущен MS Word, в котором будет открыт созданный ранее файл.

4.2 Экспорт простого отчёта в формате PDF

Рассмотрим способ вывода простого отчёта в PDF-файл. В этот раз мы не будем создавать новых экранных форм и промежуточных файлов, а всю процедуру генерации отчёта оформим в теле единственного метода. Таким образом, создаваемый нами отчёт полностью лишен возможности редактирования и предварительного выбора контента.

Для реализации этого способа понадобится подключить несколько библиотек, среди которых: *QPrinter* и *QTextDocument*, которые не могут быть подключены без использования модуля *PrintSupport*, подключим данный модуль в корневом файле проекта *dbconnection.pro* (Листинг 4.10):

Листинг 4.10 - Подключение модуля PrintSupport

```
QT += core gui sql axcontainer printsupport
```

Чтобы изменения вступили в силу, решение нужно перестроить. Теперь можно подключить необходимые библиотеки в заголовочном файле *mainwindow.h* (Листинг 4.11):

Листинг 4.11 - Добавляемые к заголовочному файлу библиотеки

```
#include <QPrinter>
#include <QTextDocument>
#include <QFileDialog>
```



Библиотека *QPrint* содержит описание класса, отвечающего за настройки печати такие как: ориентация листа, размер бумаги, выходной формат в случае печати в файл и т.д.

QTextDocument – это контейнер для форматированного текста, который может содержать различные элементы, например, изображения, таблицы и т.д.

Переходим к редактированию главной формы. Необходимо добавить дополнительную простую кнопку (*pushButton*) и переименовать её в «Экспорт в PDF» (Рисунок 4.3):

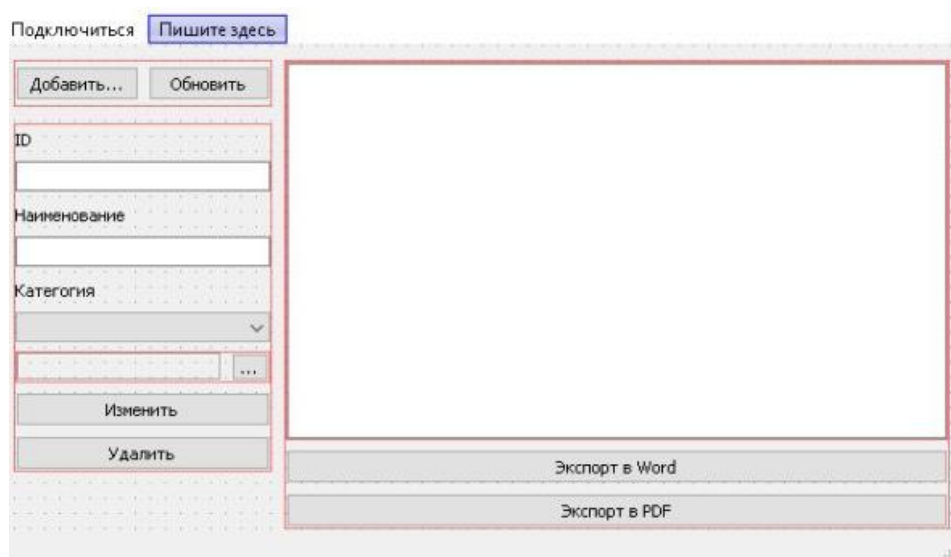


Рисунок 4.3 - Главная форма с добавленной кнопкой экспорта

Создадим слот для новой кнопки, связанный с сигналом *clicked()* и опишем процесс генерации текста отчёта (Листинг 4.12):

Листинг 4.12 - Генерация текста отчёта

```
void MainWindow::on_pushButton_6_clicked()
{
    QString str;
    str.append("<html><head></head><body><center>"
              +QString("Пример создания отчёта"));
    str.append("<table border=1><tr>");
```


Листинг 4.12 (Окончание)

```

    str.append("<td>" + QString("ID") + "</td>");
    str.append("<td>" + QString("Наименование") + "</td>");
    str.append("<td>" + QString("Категория") + "</td></tr>");
    QSqlQuery *query = new QSqlQuery();
    query->exec("SELECT * FROM product");
    query->next();
    while(query->next())
    {
        str.append("<tr>");
        str.append("<td>"
            + QString(query->value(0).toString()) + "</td>");
        str.append("<td>"
            + QString(query->value(1).toString()) + "</td>");
        str.append("<td>"
            + QString(query->value(2).toString())
            + "</td></tr>");
    }
    str.append("</table>");
    str.append("</center></body></html>");
}

```

Структура формируемого текста полностью повторяет структуру, приведенную ранее при формировании промежуточного html-файла (Листинг 4.4) с тем отличием, что в данном случае строки записываются не в файл, а в специальную переменную *str*.

Далее создадим объект класса *QPrint* и с его помощью настроим параметры создаваемого документа, для этого добавим в метод *MainWindow::on_pushButton_6_clicked()* следующие строки (Листинг 4.13):

Листинг 4.13 - Настройка параметров документа

```

QPrinter printer;
printer.setOrientation(QPrinter::Portrait);
printer.setOutputFormat(QPrinter::PdfFormat);
printer.setPaperSize(QPrinter::A4);

```

Следующим шагом настроим путь к будущему файлу отчёта и при помощи объекта класса *QTextDocument* соберём текст, настройки и будущее месторасположение воедино. Напишем следующие строки далее в этом же методе (Листинг 4.14):

Листинг 4.14 - Создание файла отчёта

```

QString path =
QFileDialog::getSaveFileName(NULL, "Сохранить",
                            "Отчёт", "PDF (*.pdf)");
if (path.isEmpty()) return;
printer.setOutputFileName(path);

```

Листинг 4.14 (Окончание)

```
QTextDocument doc;  
doc.setHtml(str);  
doc.print(&printer);
```

Теперь можно запустить приложение и проверить корректность работы создания отчётов.

Контрольные вопросы

1. Что такое простой отчёт? Приведите пример.
2. Какие ещё виды отчётов существуют?
3. Какая технология использовалась при выполнении экспорта данных в MS Word?
4. Опишите фреймворк AxtiveX существующий в Qt.

Дополнительное задание

1. Доработать функцию экспорта отчёта в PDF, добавив возможность выбора таблицы, которая будет выведена в отчёте.
2. Обеспечить возможность выполнения функции без предварительного создания промежуточного файла.
3. Добавить возможность применять ранее созданные фильтры к отчёту.

5 ДОПОЛНИТЕЛЬНЫЕ ФУНКЦИИ ПРИЛОЖЕНИЯ

В этой главе приводятся примеры различных функций, наличие которых в приложении не является строго обязательной, но их реализация поможет расширить функционал предоставляемый ИС и/или сделать работу с приложением проще и безопаснее.

5.1 Добавление изображений и построение графиков средствами приложения.

Рассмотрим ситуацию, когда одновременно с информацией о названии товара, его типе, размере и т.д., необходимо сохранить информацию о внешнем виде. Одним из способов сделать это – сохранить изображение объекта: фотографию, чертеж, схему и т.д. В первой части лабораторной работы будет рассмотрен способ хранения информации о прикрепленном к записи изображении в БД с помощью программного средства, то есть в таблицу БД будет помещаться не само изображение, а ссылка на него.

В таблице *product* не предусмотрено поле для хранения ссылки на изображение. Поэтому, перед началом работы, необходимо добавить к таблице новый столбец. Ссылка на изображение представляет собой строку, максимальную длину которой довольно сложно предугадать, поэтому в качестве типа данных будем использовать тип данных с максимально возможной длиной строки (*varchar(MAX)*), то есть пользователь сможет вводить до $(2^{31} - 1)$ символов, а так как это тип переменной длины, то в случае использования более коротких строк она будет усечена до нужного размера. Таким образом, используя этот тип данных можно хранить любые строковые ссылки. Для добавления нового столбца, в MS SSMS нужно подключиться к своей БД и при помощи редактора SQL-скриптов (ctrl+N) выполнить следующий скрипт (Листинг 5.1):

Листинг 5.1 - Добавление нового столбца к таблице

```
ALTER TABLE product ADD ImagePath varchar(MAX) ;
```

Добавление объекта для демонстрации прикрепленного изображения

Для демонстрации изображения можно использовать объект класса *QLabel* (через свойство *Pixmap*). Для выбора изображения объект добавим *toolButton*. Сгруппируем добавленные элементы между собой по горизонтали. В результате экранная форма будет иметь вид как показано на рисунке ниже (Рисунок 5.1)

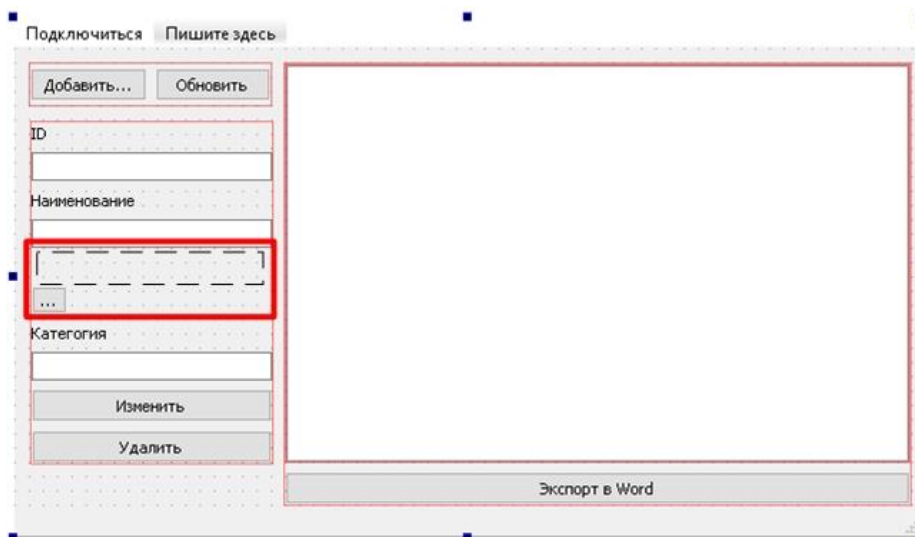


Рисунок 5.1 - Добавленные элементы на главной форме

Добавление изображения в table при нажатии на toolButton

- 1) В заголовочном файле *mainwindow.h* добавим переменную *Img* типа *QString*.
- 2) Настроим обработчик нажатия на кнопку выбора изображения (Листинг 5.2):

Листинг 5.2 - Обработчик нажатия на "..."

```
void MainWindow::on_toolButton_clicked()
{
    Img = QFileDialog::getOpenFileName(0, "Открыть файл",
                                       Img, "*.jpg");
    ui->label_4->setPixmap(Img);
}
```

Теперь, при нажатии на кнопку «...» будет открыто диалоговое окно выбора файла, где пользователь сможет выбрать нужное ему изображение. После нажатия на кнопку «Открыть», изображение будет добавлено в *table* при помощи метода *setPixmap()*.

Сохранение пути к изображению в БД

Так как появились новые поля, то, в первую очередь, необходимо внести изменения в методы, которые отвечают за их обновление, а именно: обработчик щелчка левой кнопки мыши по таблице (*MainWindow::on_tableView_clicked*), обработчик нажатия на кнопку «Изменить» (*MainWindow::on_pushButton_3_clicked()*) и аналогичные методы других, созданных ранее экранных форм.

Изменим описание метода выбора строки в объекте *tableView*. Вносимые изменения выделены полужирным (Листинг 5.3):

Листинг 5.3 - Изменение в обработчике щелчка по таблице

```

void MainWindow::on_tableView_clicked(const QModelIndex &index)
{
    int temp_ID;
    temp_ID = ui->tableView->model()->data(ui->tableView-
>model()->index(index.row(),0)).toInt();
    QSqlQuery *query = new QSqlQuery();
    query->prepare("SELECT name, category, ImagePath FROM product
WHERE ID = :ID");
    query->bindValue(":ID",temp_ID);

    ui->lineEdit->setText(QString::number(temp_ID));
    if (query->exec())
    {
        query->next();
        ui->lineEdit_2->setText(query->value(0).toString());
        ui->lineEdit_3->setText(query->value(1).toString());
        Img = query->value(2).toString();
        ui->label_4->setPixmap(Img);
    }
}

```

Теперь при выборе строки в *tableView* из БД также будет извлекаться значение пути к изображению, который, после приведения к типу *QString*, будет передан в качестве аргумента в свойство *Pixmap* и таким образом, изображение будет показано в объекте.

Далее необходимо внести изменение в обработчик нажатия на кнопку «Изменить» (Листинг 5.4):

Листинг 5.4 - Модернизация обработчика нажатия на кнопку "Изменить"

```

void MainWindow::on_pushButton_3_clicked()
{
    QSqlQuery *query = new QSqlQuery();
    query->prepare("UPDATE product SET name = :name, "
                  "category = :category, ImagePath =
:image WHERE ID = :ID");
    query->bindValue(":ID",ui->lineEdit->text());
    query->bindValue(":name",ui->lineEdit_2->text());
    query->bindValue(":category",ui->lineEdit_3->text());
    query->bindValue(":image",Img);
    query->exec();

    ui->lineEdit->setText("");
    ui->lineEdit_2->setText("");
    ui->lineEdit_3->setText("");
    ui->label_4->setText("");

    MainWindow::on_pushButton_clicked();
}

```

Пробный запуск

Осуществим пробный запуск программы и проверим правильность добавления изображений к существующим записям.

5.2 Построение графика по данным таблицы БД

Различного рода графики и средства их построения часто можно встретить в различных программных средствах. Такой способ представления информации прост для понимания, позволяет быстрее понять суть отчёта или доклада. Далее мы рассмотрим процесс создания простого графика средствами среды программирования Qt Creator.

В нашей БД отсутствует таблица, содержащая нужные для построения простого графика данные, добавим её.

В SSMS нужно подключиться к БД и выполнить следующий скрипт:

Листинг 5.5 - Добавление новой таблицы

```
CREATE TABLE chart
(
    Month int IDENTITY,
    Proceed float
);
```

Добавьте в таблицу 12 строк с произвольными числами в диапазоне от -100 до 100

Для продолжения работы необходимо скачать архив с библиотекой QCustomPlot (ссылка на скачивание доступна под видео №10 [1]). Разархивируйте его и скопируйте два выделенных на рисунке файла (Рисунок 5.2) в корневую папку приложения (Рисунок 5.3)

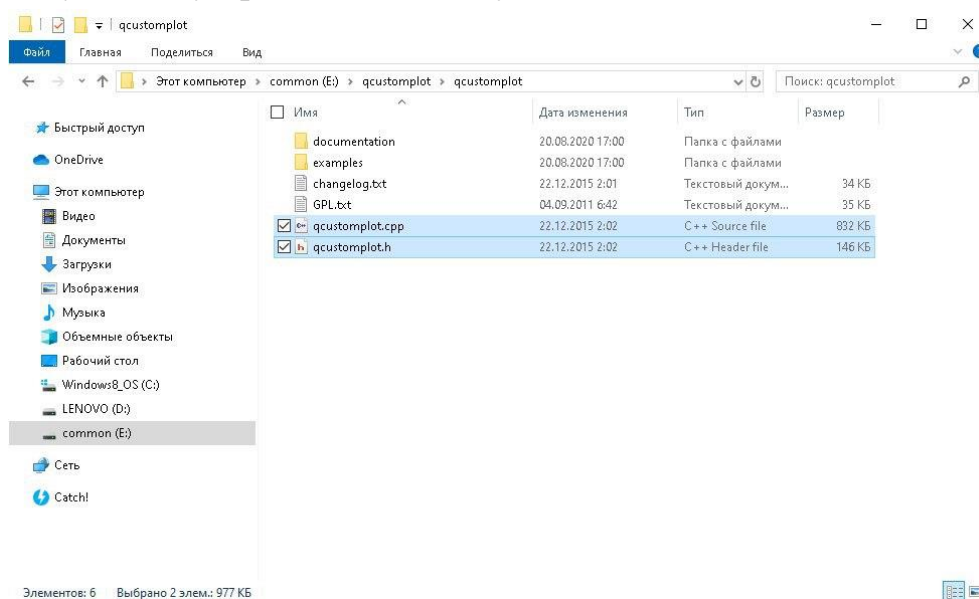


Рисунок 5.2 - Распакованный архив с QCustomPlot

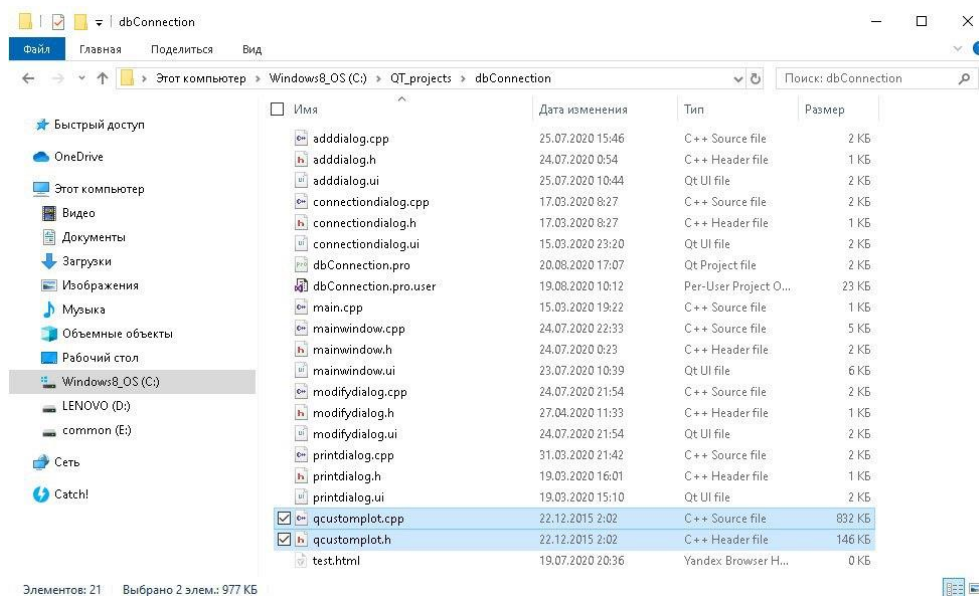


Рисунок 5.3 - Перемещение файлов в корневой каталог

Открыв проект в среде программирования Qt Creator нужно добавить скопированные файлы к проекту. Для этого щелкните правой кнопкой мыши по папке dbconnection в окне «Проекты» и выбрать пункт «Добавить существующие файлы...» (Рисунок 5.4)

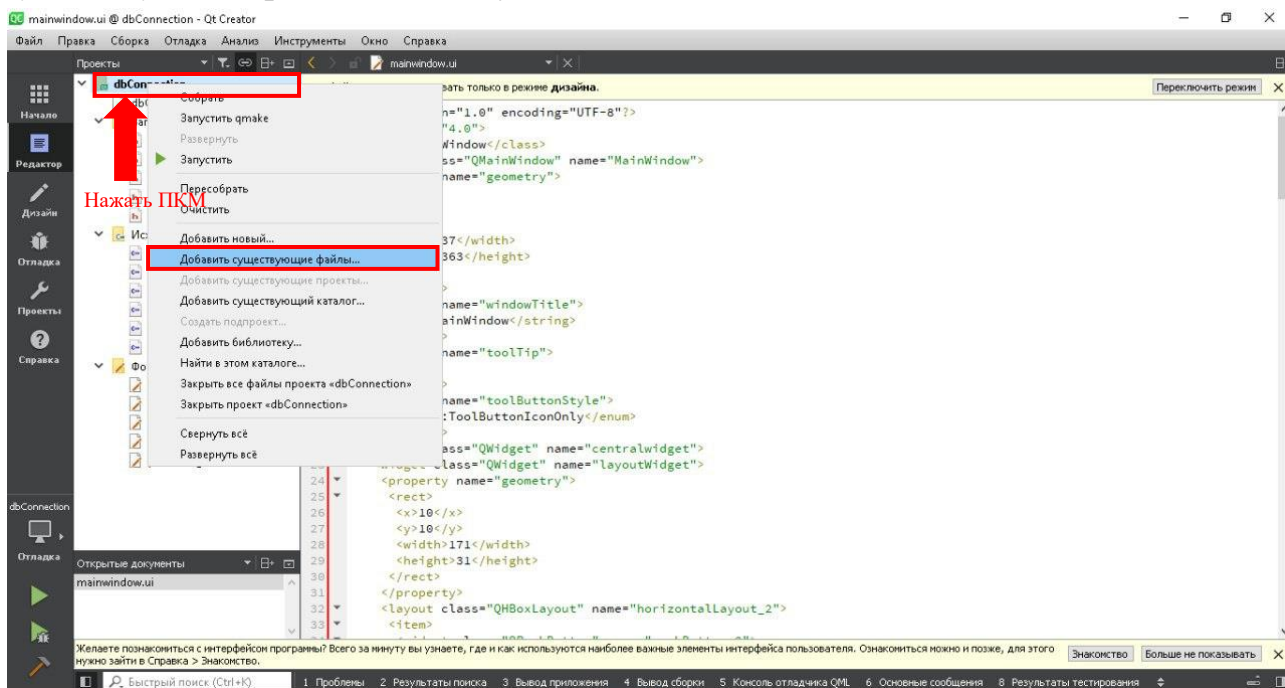


Рисунок 5.4 - Добавление существующих файлов к проекту

В открывшемся диалоговом окне нужно выбрать файлы, скопированные ранее (Рисунок 5.5):

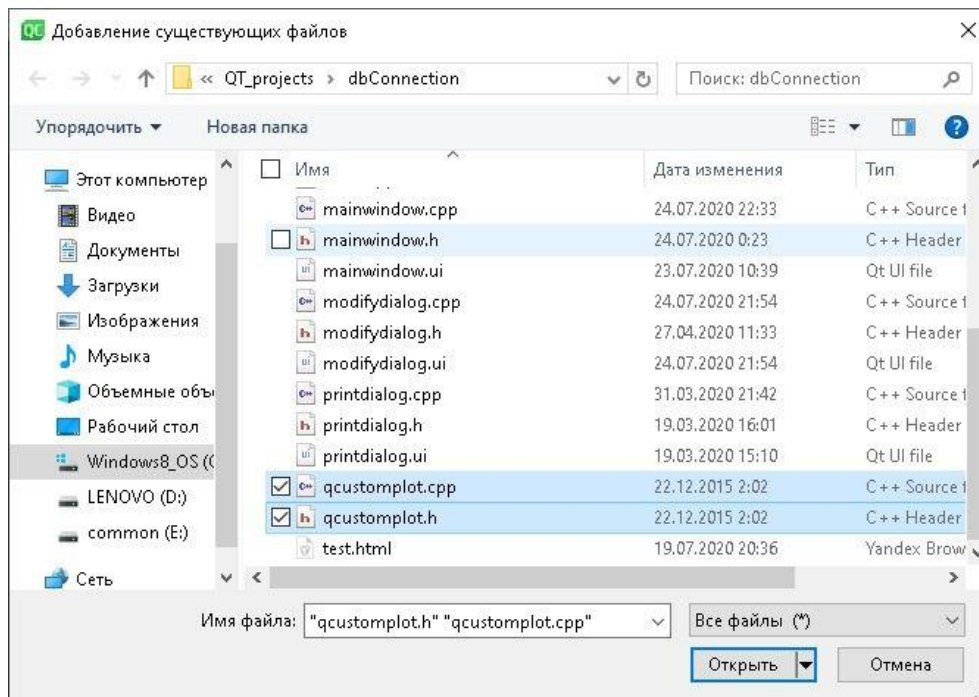


Рисунок 5.5 - Выбор файлов для добавления к проекту

После добавления новых файлов, создайте новую экранную форму (процесс добавления новой экранной формы описан в лабораторной работе №1, см. стр. 11), назовите класс *printGraf*, имена файлов оставьте без изменений. На экранную форму добавьте объект *Grid Layout*. Нажмите левой кнопкой мыши на пустом месте формы и нажмите CTRL+N чтобы Grid Layout занял всё пространство формы.

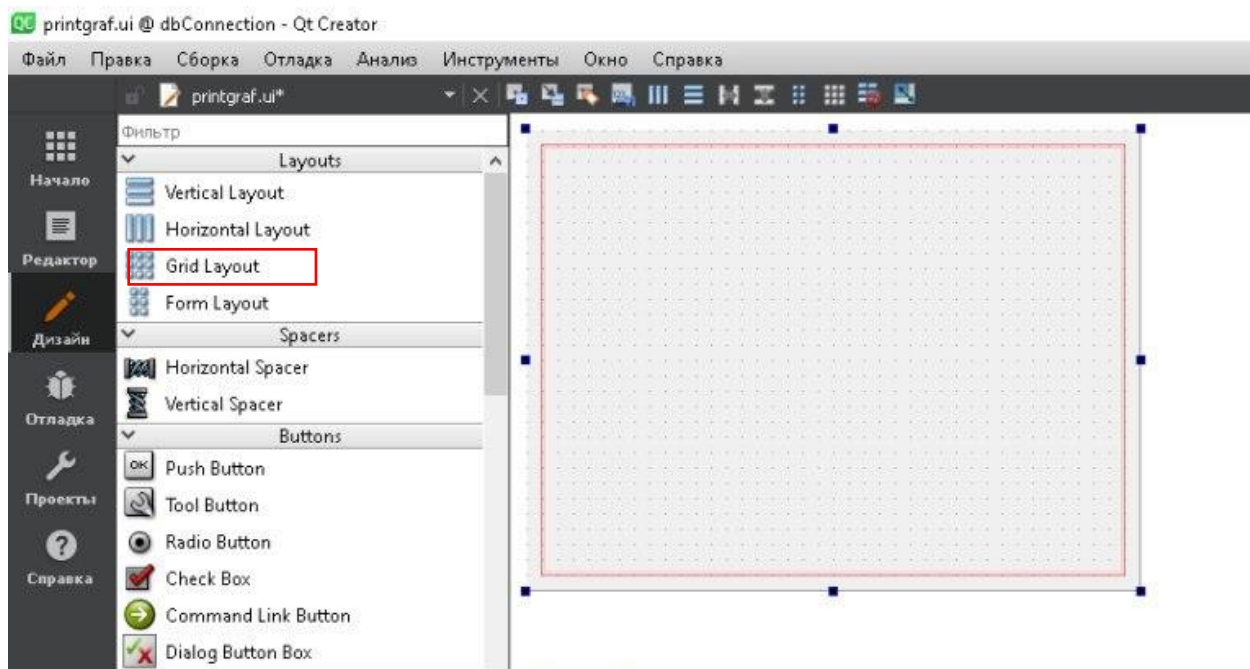


Рисунок 5.6 - Добавление Grid Layout



Qt включает в себя набор классов по управлению компоновкой (размещением), которые используются для описания расположения виджетов (объектов-наследников класса *QWidget*) в пользовательском интерфейсе. Класс *QGridLayout* один из классов управления компоновкой и отвечает за размещение виджетов в узлах сетки.

Подробнее с классами компоновки можно ознакомиться в документации к проекту

Необходимо добавить объект *widget* (Рисунок 5.7), но поместить его не на форму (как обычный компонент), а на объект *Grid Layout*, следует учесть что *widget* не имеет визуального, поэтому визуального изменения формы не будет.

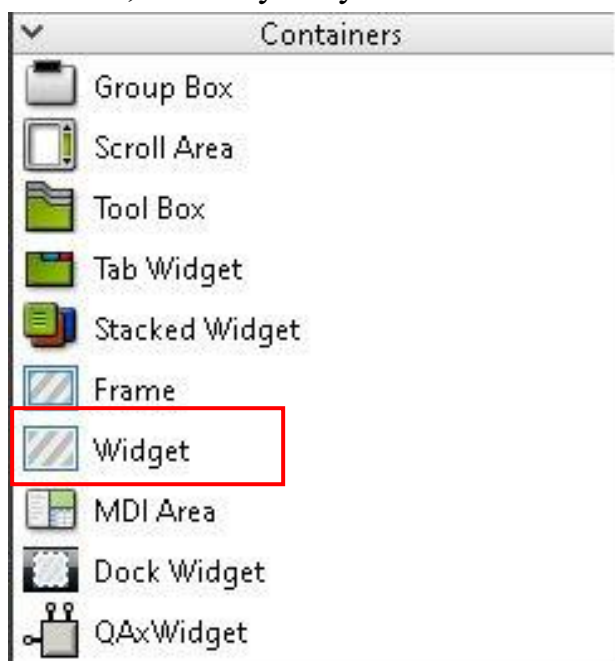


Рисунок 5.7 - *Widget* на панели компонентов

Теперь необходимо преобразовать стандартный виджет к объекту класса *QCustomPlot*, в котором уже содержатся все необходимые для рисования методы. Для выполнения преобразования выполните следующие действия:

- 1) Нажать правой кнопкой на объекте и, в открывшемся контекстном меню, выбрать пункт «Преобразовать в...» ()

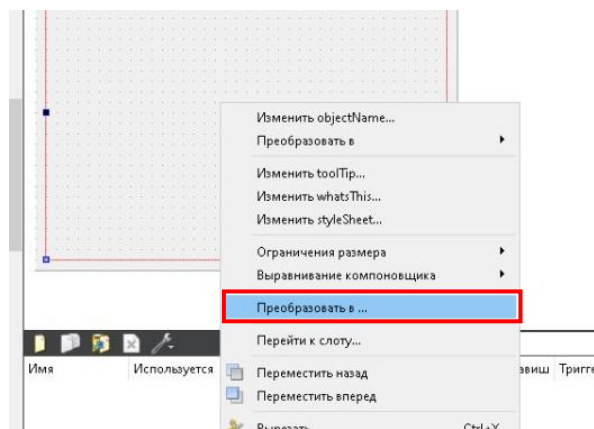


Рисунок 5.8 - Контекстное меню виджета

- 2) В открывшемся диалоговом окне, в нижней части, нужно заполнить поля как показано на рисунке ниже (1, Рисунок 5.9), после чего, нажать на кнопку «Добавить» (2) и завершить процесс нажатием на кнопку «Преобразовать» (3)

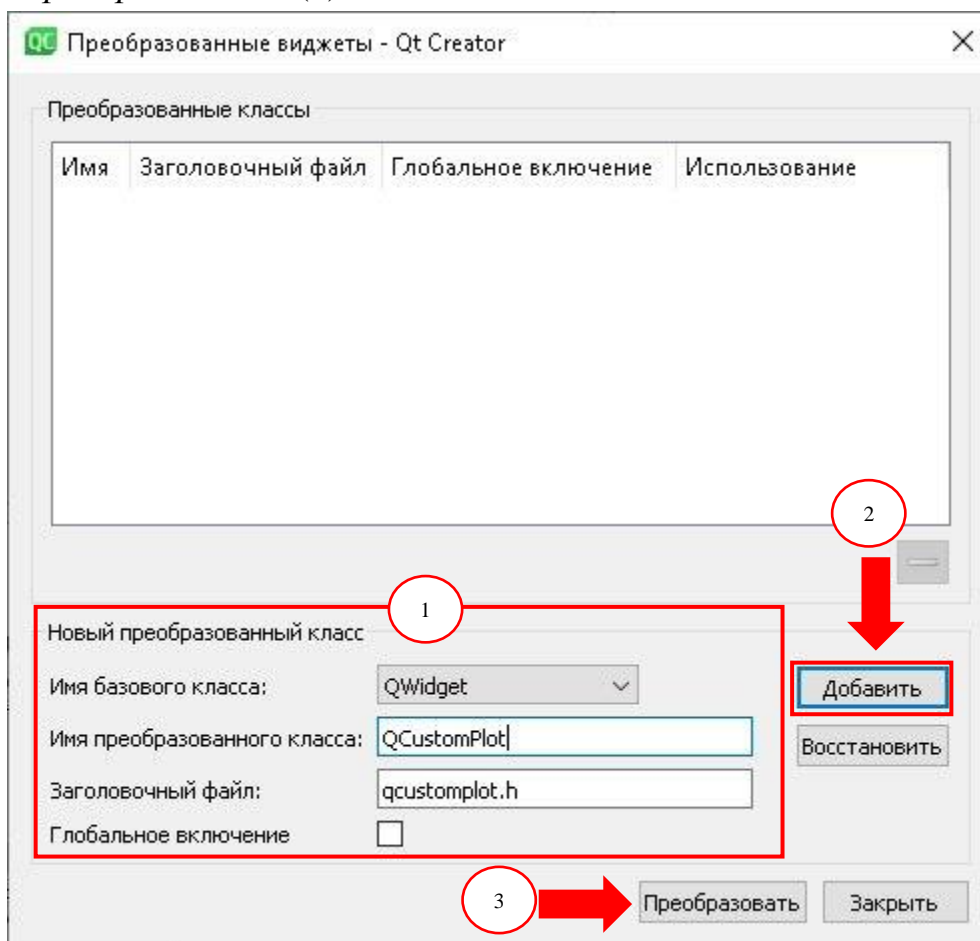


Рисунок 5.9 - Процесс преобразования виджета к нужному классу



Для корректной работы библиотеки `QCustomPlot` требуется предварительно подключить модуль `PrintSupport` в файле `dbconnection.pro`, но так как данный модуль был подключен в ходе предыдущей лабораторной работы, подключать его снова не нужно.

Перед тем как перейти к написанию кода построения графика, подключим уже знакомую библиотеку `QSqlQuery` в заголовочном файле `printgraf.h`, после – перейти в файл `printgraf.cpp`.

При создании формы, мы не добавили на неё никаких кнопок или иных элементов управления, так как график будет отображаться на форме сразу при её открытии, поэтому весь код подготовки и построения графика будет помещён в конструктор формы.

Процесс построения графика состоит из трех этапов:

- 1) Задание начальных значений.
- 2) Чтение координат точки из базы данных.
- 3) Размещение точек на графике.
- 4) Настройка параметров отображения.

На первом этапе нужно задать заголовок для будущего графика, для этого, после автоматически созданного кода необходимо написать следующий код (Листинг 5.6)

Листинг 5.6 - Добавление названия графика

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

    ui->widget->plotLayout()->insertRow(0);
    ui->widget->plotLayout()->addElement(0, 0,
                                         new QCPPlotTitle(ui->widget, "График выручки"));
}
```

Далее необходимо объявить и инициализировать ряд переменных и объектов, которые будут использоваться при построении: переменные для хранения координат точки, максимальное и минимальное значение координат (для определения границ графика) и объект класса `QSqlQuery` для извлечения данных из базы (Листинг 5.7).

Листинг 5.7 - Объявление и инициализация необходимых переменных

```
QVector <double> dx, dy; //для хранения координат точки
double minX, minY, maxX, maxY; //для определения границ
```

Листинг 5.7 (Окончание)

```

minX = 0;
minY = 0;
maxX = 0;
maxY = 0;
QSqlQuery *query = new QSqlQuery();

```

Так как запрос выполняется без передачи в него каких-либо параметров, то можно выполнять его без предварительной подготовки, но чтобы исключить вероятность обработки данных в случае ошибки в выполнении метода `exec()`, поместим его в условный оператор. Запрос должен вернуть 12 строк, перебирать которые удобно в цикле.

Сразу после получения очередной строки, она сравнивается с максимальными и минимальными значениями, если новое значение подходит под условия – оно заменяет предыдущее. После проверки, значение координаты x (первый столбец) заносится в вектор dx , те же действия выполняются и для координаты y (второй столбец).

Получив нужные значения, необходимо настроить координатную сетку, цвет и толщину линий элементов графика, а также данные, которые необходимо показать на графике. Указанные настройки можно произвести при помощи объекта класса *QCPBars*. В завершении создадим добавим подписи осям, границы отображения, настроим масштаб и шаг. Код выполняющий описанные выше действия приведены ниже (Листинг 5.8):

Листинг 5.8 - Получение данных и построение графика

```

if (query->exec("SELECT * FROM chart"))
{
    while (query->next())
    {
        //Поиск максимального и минимального значения координат
        if (minX>=query->value(0).toDouble())
            minX = query->value(0).toDouble();
        if (minY>=query->value(1).toDouble())
            minY = query->value(1).toDouble();
        if (maxX<=query->value(0).toDouble())
            maxX = query->value(0).toDouble();
        if (maxY<=query->value(1).toDouble())
            maxY = query->value(1).toDouble();
        //Добавление координат в вектора
        dx << query->value(0).toDouble();
        dy << query->value(1).toDouble();
    }
    //Создание объекта с привязкой к осям виджета
    QCPBars *bar = new QCPBars(ui->widget->xAxis,
                               ui->widget->yAxis);
    bar->setName("Значение");
}

```

Листинг 5.8 (Окончание)

```
//Цвет заливки графика (R,G,B,Transparency)
bar->setBrush(QColor(255,0,0,255));
bar->setData(dx,dy); //координаты точек
bar->setWidth(0.25); //толщина линий

//Добавление подписей осей
ui->widget->xAxis->setLabel("Месяц");
ui->widget->yAxis->setLabel("Выручка");

//Установка границ отображения графика
ui->widget->xAxis->setRange(minX,maxX+0.20);
ui->widget->yAxis->setRange(minY,maxY+1);

//Запрет автоматического определения шага шкалы
ui->widget->xAxis->setAutoTickStep(false);
ui->widget->yAxis->setAutoTickStep(false);

//Ручная установка шага
ui->widget->xAxis->setTickStep(1);
ui->widget->yAxis->setTickStep(1);
}
```

Для перехода к окну вывода графического отчёта, добавим новый пункт в главное меню, который назовём «Отчёт» и добавим в него подпункт «Графический отчёт...» (Рисунок 5.10)

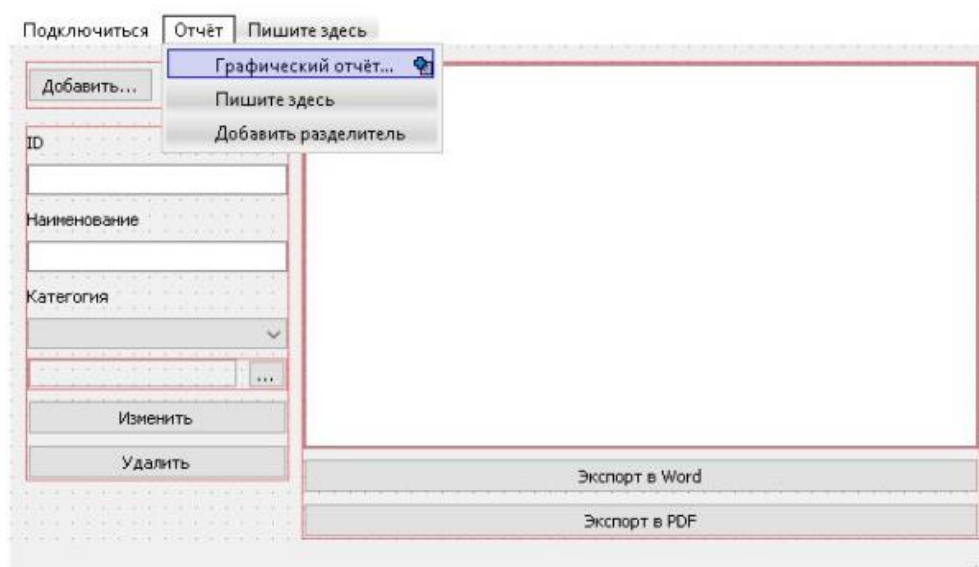


Рисунок 5.10 - Новый пункт меню

Создайте слот для «Графический отчёт...» связанный с сигналом `triggered()`, но до того, как приступить к его описанию нужно добавить файлы `printgraf.h` и `ui_printgraf.h` в заголовочный файл `mainwindow.h` () и добавить переменную класса `printGraf` (Рисунок 5.9):

Листинг 5.9 - Подключение новых библиотек и объявление переменной

```
//в разделе подключения библиотек
#include "printgraf.h"
#include "ui_printgraf.h"
```

Листинг 5.9 (Окончание)

```
...
//в разделе объявления переменных
private
    Ui::MainWindow *ui;
...
printGraf *pg;
```

Возвращаемся к созданному слоту, куда запишем следующий код:

Листинг 5.10 - Обработка реакции на выбор пункта "Графический отчёт..."

```
void MainWindow::on_action_triggered()
{
    dlg = new ConnectionDialog();
    dlg->show();
}
```

Пробный запуск

Выполните пробный запуск и подключитесь к БД. Выберите графический отчёт и убедитесь в корректной работе функции. Если все выполнено правильно на форме будет изображен график как на рисунке ниже (Рисунок 5.11)

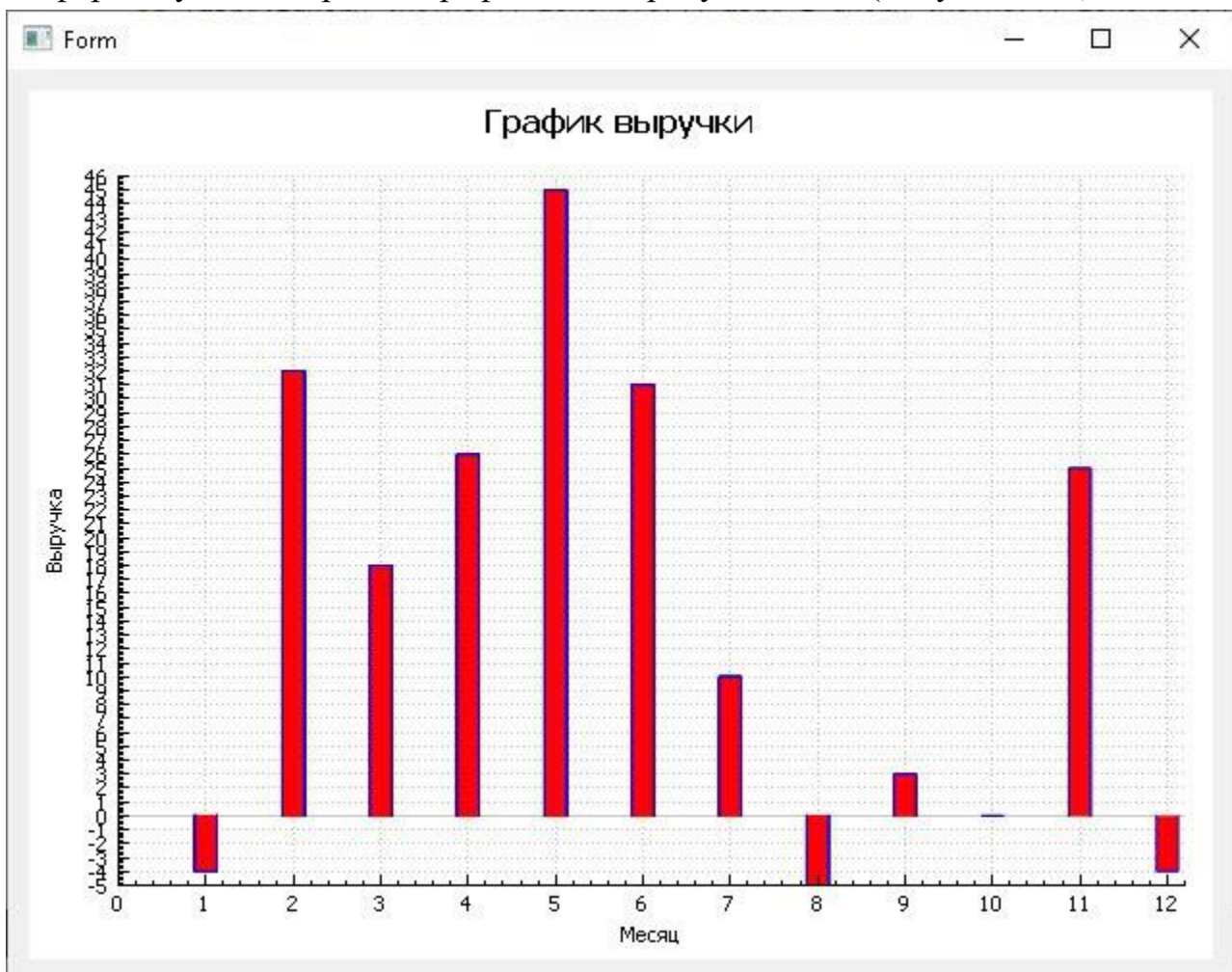


Рисунок 5.11 - График, построенный с помощью QCustomPlot

5.3 Реализация выбора значения атрибута из ниспадающего списка

Внесение изменений в БД

Перед началом выполнения работы необходимо внести изменения в используемую БД. Необходимо провести процесс нормализации таблицы *product*, который приведет к образованию новой таблицы, связанной с нормализуемой таблицей. Безусловно можно удалить существующую таблицу *product* и создать уже нормализованную структуру, но данный способ приведёт к потере данных, которая в реальной ситуации может дорого стоить. Во избежание потери данных в среде SSMS, после подключения к целевой БД, выполним следующий скрипт (Листинг 5.11):

Листинг 5.11 - Скрипт разделения таблицы *product*

```
--выделение category в отдельную таблицу
SELECT DISTINCT category AS name -- только уникальные значения
INTO category                    -- помещаем в новую таблицу
FROM product
ORDER BY name;

--добавление столбца - первичного ключа
ALTER TABLE category ADD id INT IDENTITY PRIMARY KEY;

--добавление столбца в таблицу product
ALTER TABLE product ADD catID INT;

--заполнение столбца catID значениями
UPDATE product SET catID = (SELECT id FROM category
WHERE product.category = name);

--удаление столбца category из таблицы product
ALTER TABLE product DROP COLUMN category;

--добавление связи между новой таблицей и product
ALTER TABLE product
ADD CONSTRAINT FK_product_category
FOREIGN KEY (catID) REFERENCES category (id);
--проверка
SELECT *
FROM product a inner join category b
ON a.catID = b.id;
```

Теперь мы имеем в распоряжении две связанные таблицы. Механизм связей не позволит использовать в дочерней таблице (в нашем случае *product*) значений, которых нет в родительской таблице (*category*).

Внесение изменений в экранные формы

Созданная таблица содержит информацию о категории к которой относится продукт. Такую информацию удобнее выбирать из списка, чем вводить каждый раз заново. Во-первых, это снижает вероятность ошибки, а во-вторых – данные о категориях обычно обновляются гораздо реже, чем о продуктах, относящихся к этим категориям.

Для реализации выбора одного значения из таблицы category воспользуемся объектом класса *QComboBox*, которым заменим все *lineEdit* используемые ранее для указания категории. Начнём с диалогового окна внесения изменений.

Таким образом, форма будет выглядеть как на рисунке ниже (Рисунок 5.12):

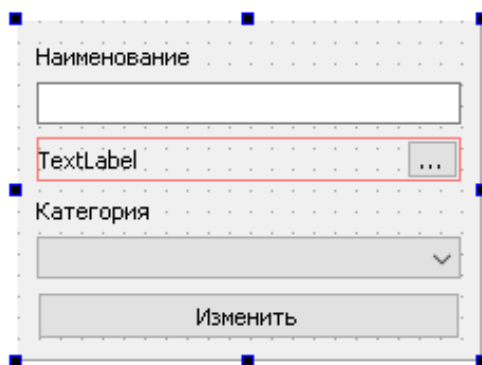


Рисунок 5.12 – Изменённая форма модификации записи

Заполнение списка выбора

Список выбора заполняется названиями категорий из соответствующей таблицы БД, следовательно, к моменту открытия формы – список должен быть заполнен. Разместим фрагмент кода, отвечающего за заполнение списка в конструктор формы, это позволит получить заполненный список выбора сразу после открытия формы (Листинг 5.12, добавленные строки выделены полужирным).

Листинг 5.12 – Внесение изменений в конструктор формы

```
ModifyDialog::ModifyDialog(QWidget *parent) :  
    QDialog(parent),  
    ui(new Ui::ModifyDialog)  
{  
    ui->setupUi(this);  
    ui->label_3->setScaledContents(true);
```


Листинг 5.12 (Окончание)

```
QSqlQuery *queryCombo = new QSqlQuery();
queryCombo->exec("SELECT name FROM category");
while (queryCombo->next())
{
    ui->comboBox->addItem(queryCombo->value(0).toString());
}
}
```

В предыдущих лабораторных работах, при выполнении запросов к БД при помощи объектов класса *QSqlQuery*, перед вызовом метода *exec()*, мы вызывали метод *prepare()*, которому в качестве параметра передавали строку запроса. В данном случае предварительная подготовка запроса не нужна, так как мы не передаём в СУБД никаких параметров, поэтому мы сразу можем указать текст запроса в качестве параметра метода *exec()*.

Отображение первоначальных данных.

Назначение формы – внесение изменений в запись. Это значит, что форма должна быть предзаполнена данными до того как будет представлена пользователю и метод такого предзаполнения был реализован в лабораторной работе №4 (стр. 46), но так как был заменен объект *lineEdit* на *comboBox*, то метод также следует изменить (Листинг 5.13):

Листинг 5.13 - Внесение изменений в сигнал *sendingID*

```
void ModifyDialog::sendingID(int aa)
{
    tempID = aa;
    QSqlQuery *query = new QSqlQuery();
    query->prepare("SELECT name, catID, ImagePath FROM product
WHERE ID = ?");
    query->bindValue(0, aa);
    if (query->exec())
    {
        query->next();
        ui->lineEdit->setText(query->value(0).toString());
        ui->comboBox->setCurrentIndex(query->value(1).toInt()-1);
        Img = query->value(2).toString();
        ui->label_3->setPixmap(Img);
    }
}
```

Для того чтобы выбрать нужный элемент в *comboBox* нужно знать его индекс и передать его через свойство *setCurrentIndex*. Здесь следует обратить внимание на тот факт, что поле *ID* в таблице *category* нумеруется начиная с 1, а

индекс списка выбора *comboBox* с 0, поэтому при записи значения через *setCurrentIndex* нужно вычесть из *catID* единицу, а при чтении индекса – прибавить.

Сохранение изменений в БД

В отличие от Лабораторной работы №4, в поле «*catID*» должен быть занесено не текстовое значение (имя категории), а целочисленный код, который соответствует выбранной категории. Для этого нужно организовать чтение индекса выбранной записи при помощи свойства *currentIndex* (Листинг 5.14)

Листинг 5.14 - Обработчик нажатия на кнопку "Изменить"

```
void ModifyDialog::on_pushButton_clicked()
{
    QSqlQuery *query = new QSqlQuery();
    query->prepare("UPDATE product SET name = :name, "
        "catID = :category, ImagePath = :image WHERE ID = :ID");
    query->bindValue(":ID", tempID);
    query->bindValue(":name", ui->lineEdit->text());
    query->bindValue(":category", ui->comboBox->currentIndex() + 1);
    query->bindValue(":image", Img);
    if(query->exec())
    {
        close();
    }
}
```

Аналогично вносятся изменения в остальных методах и формах приложения.

Отображение данных из нескольких таблиц

После нормализации, таблица, отображаемая на главной форме, стала менее понятной для пользователя, так как словесные обозначения категорий теперь заменены кодами.

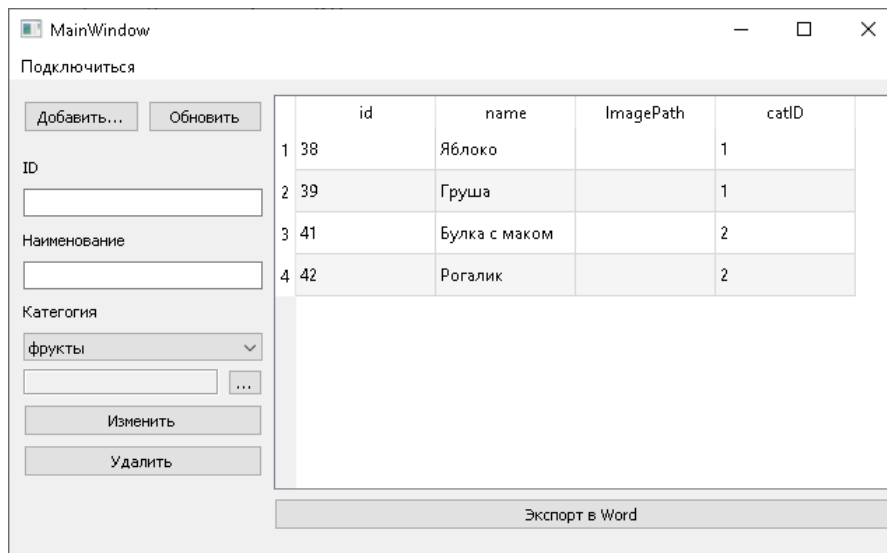


Рисунок 5.13 - Вид главной формы после нормализации таблицы product

Для исправления этого недочёта необходимо внести изменение в запрос, который используется в обработчике нажатия на кнопку «Обновить».

Листинг 5.15 – Внесение изменений в функцию обновления tableView

```
void MainWindow::on_pushButton_clicked()
{
    fl = 1;
    qmodel = new QSqlQueryModel();
    qmodel->setQuery("SELECT *"
"FROM product a inner join category b on a.catID = b.ID");
    ui->tableView->setModel(qmodel);
}
```

Проверяем правильность работы запроса (Рисунок 5.14):

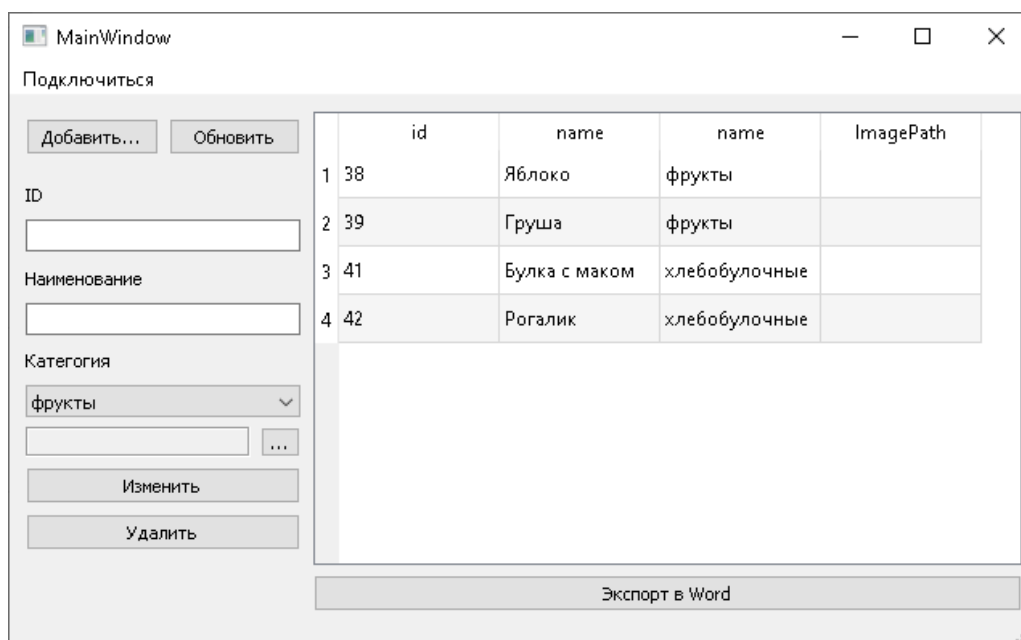


Рисунок 5.14 - Содержание tableView после изменения запроса

5.4 Упрощение процесса ввода и редактирования данных

Подготовка БД

С помощью изученного и широко применяемого в ходе выполнения предыдущих лабораторных работ класса *QLineEdit*, можно ввести практически любые данные, которые предполагается хранить и обрабатывать информационной системой, но использование специальных компонентов может упростить процесс ввода и редактирования данных за счёт предоставления дополнительных возможностей, например, поле ввода даты снабжается небольшим календарём, который появляется при нажатии на специальную кнопку встроенную в компонент, панели и другие способы группировки элементов. Достаточно просто пролистать палитру компонентов и убедиться что

Для выполнения этой лабораторной работы необходимо добавить в таблицу *product* новое поле типа *date*.

Листинг 5.16 - Добавление столбца

```
ALTER TABLE product ADD prodDate date;
```

Добавление объекта *dateEdit*

Перейдя на форму для добавления новой строки, нужно разместить компонент *dateEdit* с панели компонентов и один *label* для поясняющей надписи. Для того, чтобы пользователь мог выбрать дату из ниспадающего календаря, а не вводить вручную, нужно при помощи редактора свойств задать значение *true* для свойства *calendarPopup* (раздел *QDateTimeEdit*). После внесения всех изменений форма добавления строки будет иметь вид (Рисунок 5.15):

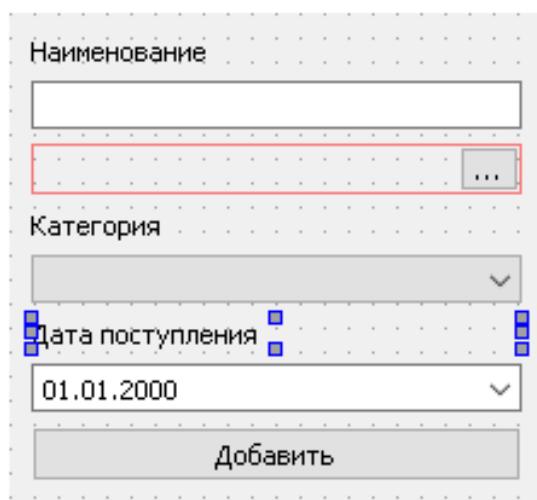


Рисунок 5.15 - Измененная форма добавления строки

Добавление даты в БД через объект *dateEdit*

Далее переходим к редактированию кода в файле `adddialog.cpp` (Листинг 5.17). В конструкторе формы пропишем текущую дату в качестве начальной. Для получения текущей даты (которая хранится в операционной системе), нужно воспользоваться методом `currentDate()` класса `QDate`. Записать полученное значение в объект *dateEdit* можно при помощи свойства `setDate()`:

Листинг 5.17 - Конструктор `AddDialog` (Фрагмент)

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

    ...
    ui->dateEdit->setDate(QDate::currentDate());
}
```

Чтение даты из объекта *dateEdit*

Внесем изменения в обработчик нажатия на кнопку «Добавить»:

Листинг 5.18 - Обработчик нажатия на кнопку "Добавить"

```
void AddDialog::on_pushButton_clicked()
{
    QSqlQuery *query = new QSqlQuery();

    query->prepare("INSERT INTO product (name, catID, "
                  "ImagePath, prodDate) "
                  "VALUES (:name, :catID, :image, :date)");
    query->bindValue(":name", ui->lineEdit->text());
    query->bindValue(":catID", catID);
    query->bindValue(":image", Img);
    query->bindValue(":image", ui->dateEdit->text());
    query->exec();

    close();
}
```

Для получения данных из компонента *dateEdit* в формате строки (класс `QString`) можно воспользоваться методом `text()`, аналогично методу класса `QLineEdit`.



Применение в метода *text()* объектом класса *QDateEdit* возможно благодаря механизму наследования. Этот класс имеет общего предка с *QLineEdit* в котором это свойство и было описано и не заблокировано в потомках. Использование этого свойства в данной ситуации оправданно. Оно позволяет избежать дополнительного преобразования типа (из даты в текст).

Тем не менее для записи даты в объект следует использовать метод *setDate()*, который позволит избежать ошибочного ввода даты. Более подробно о поддерживаемых методах см. документацию к проекту.

Изменения коснулись и самого запроса, так как был добавлен новый столбец, он также быть включён в текст запроса.

После применения изменений (нажатие на молоток в левом нижнем углу (Рисунок 1.14)), запускаем проект для проверки правильности работы приложения.

Контрольные вопросы

1. При помощи какой группы операторов таблица *product* была разделена на две таблицы?
2. Какой тип данных используется для хранения информации об изображении и почему?
3. Можно ли хранить изображение непосредственно в базе данных?
4. Где должно храниться изображение, чтобы оно было доступно всем пользователям информационной системы?

Дополнительное задание

1. Добавить функции добавления, модификации и удаления категорий.
2. Выявить недостаток предложенного метода заполнения списка выбора *ComboBox* и предложить вариант его исправления.
3. Реализовать заполнение списка выбора таблиц, хранящихся в БД, с помощью запроса.
4. Внести изменения в оставшиеся обработчики.
5. Установить текущую дату в качестве минимально возможной.
6. Придумать и реализовать вариант использования объекта класса *QListView*.

СПИСОК РЕКОМЕНДОВАННЫХ ИСТОЧНИКОВ

Видео инструкции по выполнению лабораторных работ:

1. Смирнов М. В. Qt 5.5.0+MS SQL Server 2008 Express (Видео курс) [В Интернете].- 2016 г..- <https://www.youtube.com/watch?v=8MT20sMEmRc&list=PLbE0Wv5gl4SteyKx9hOKnfJ5w1cfaErzF>.

Литература по Qt и C++:

2. Страуструп Бьерн Язык программирования C++ [Книга]. - [б.м.] : Бином, 2011.
3. М. Шлее Qt 5.10 Профессиональное программирование на C++ [Книга]. - СПб : БХВ-Петербург, 2018.

Официальная техническая документация:

4. Qt Company [В Интернете] // Qt Documentation – Qt Company - <https://doc.qt.io>
5. Microsoft [В Интернете] // Техническая документация Microsoft SQL Server. - Microsoft. - 20 03 2020 г.. - <https://docs.microsoft.com/ru-ru/sql/sql-server/?view=sql-server-ver15>.

Разное:

6. Qt документация [В Интернете] // CrossPlatform.ru. - <http://doc.crossplatform.ru/qt/>.
7. Сигналы и слоты в Qt [В Интернете] // Habr.ru. - 1 02 2009 г.. - 12 04 2020 г.. - <https://habr.com/ru/post/50812/>.

+

ПРИЛОЖЕНИЯ

Приложение А. Установка Qt Creator

Приложение Б Установка SQL Server Express

Приложение В Установка SQL Server Management Studio

А Инструкция по установке Qt Creator

Qt Creator – кроссплатформенная интегрированная среда разработки приложений. Её версии существуют практически для любой операционной системы. Среда разработки доступна в нескольких версиях в том числе и свободно распространяемой версии Community, которую можно загрузить с официального сайта проекта.

Подготовка к установке:

Для начала установки необходимо скачать менеджер установки с официального сайта проекта, выбрав интересующую версию в нижней части страницы. (Рисунок А.1).

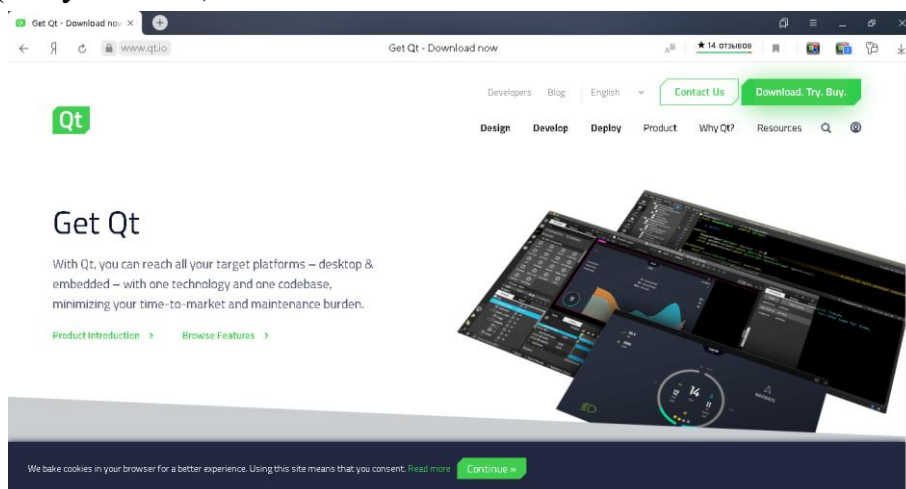


Рисунок А.1 — Официальный сайт проекта (www.qt.io)

Ход установки:

1. Запустите файл установки, в открывшемся окне (Рисунок А.2) нажмите «Next»

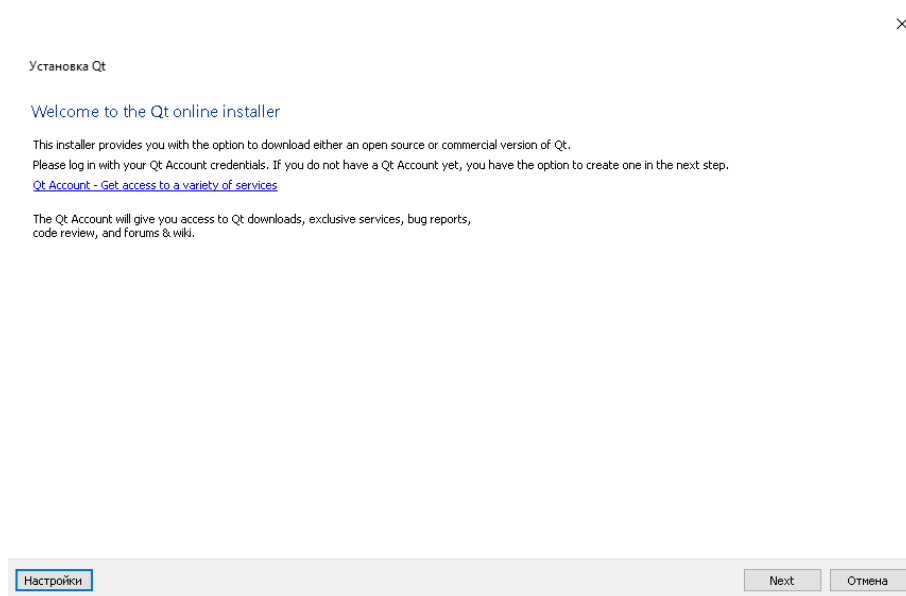


Рисунок А.2 — Окно приветствия

2. На следующей форме ((Рисунок А.3) необходимо ввести логин и пароль от учётной записи Qt Account или создать новый⁶.

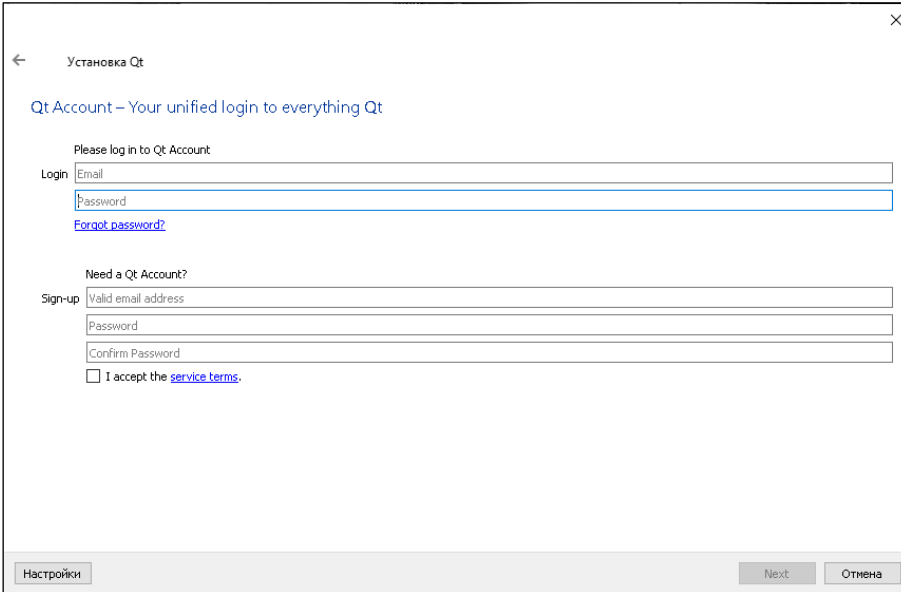
The screenshot shows the 'Qt Account' window during installation. It has a title bar with a back arrow, 'Установка Qt', and a close button. The main heading is 'Qt Account – Your unified login to everything Qt'. Below it, the text 'Please log in to Qt Account' is followed by 'Login' fields for 'Email' and 'Password', with a 'Forgot password?' link. The 'Sign-up' section is titled 'Need a Qt Account?' and includes fields for 'Valid email address', 'Password', and 'Confirm Password', along with a checkbox for 'I accept the service terms.' At the bottom, there are buttons for 'Настройки', 'Next', and 'Отмена'.

Рисунок А.3 — Ввод данных аккаунта

3. В следующем окне (Рисунок А.4) приведена информация о GPL лицензиях, для продолжения необходимо принять его условия поставив отметку в соответствующем поле окна, после чего необходимо нажать «Далее»

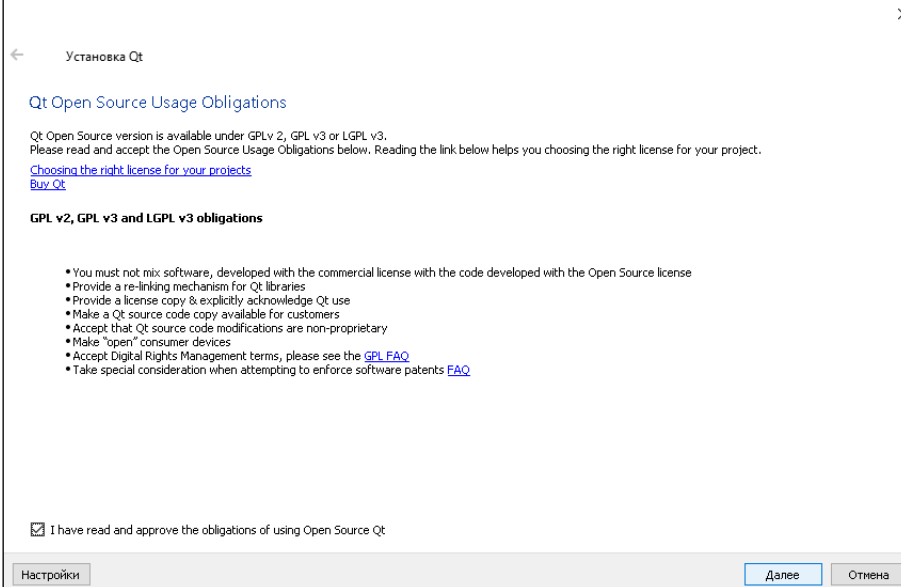
The screenshot shows the 'Qt Open Source Usage Obligations' window. It has a title bar with a back arrow, 'Установка Qt', and a close button. The heading is 'Qt Open Source Usage Obligations'. The text explains that Qt Open Source is available under GPL v2, GPL v3, or LGPL v3 and asks the user to read and accept the obligations. It provides links for 'Choosing the right license for your projects' and 'Buy Qt'. A section titled 'GPL v2, GPL v3 and LGPL v3 obligations' lists several requirements: not mixing commercial and open source code, providing a re-linking mechanism, providing a license copy, making source code available, making 'open' consumer devices, accepting Digital Rights Management terms, and considering software patents. At the bottom, there is a checkbox labeled 'I have read and approve the obligations of using Open Source Qt' which is checked. Buttons for 'Настройки', 'Далее', and 'Отмена' are at the bottom.

Рисунок А.4 — Информация об условиях GPL лицензий

Если какой-либо пункт указанных лицензий Вы по той или иной причине не можете принять, то следует отказаться от установки и дальнейшего использования продукта.

⁶ Ранее этот этап можно было пропустить, но свежие версии не допускают такой возможности.

4. Следующее окно (Рисунок А.5) информирует Вас о начале установки QT Creator на Ваш компьютер, нажмите «Далее» для продолжения.

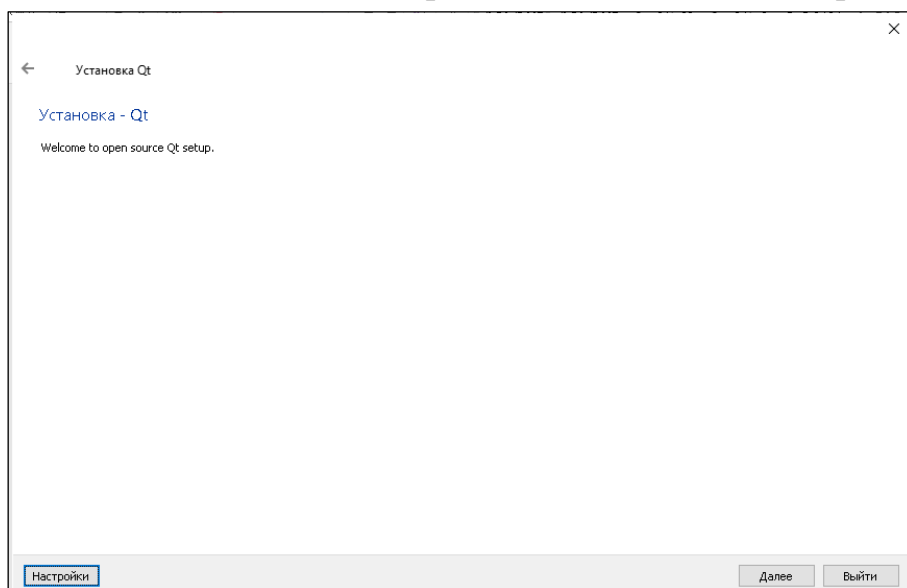


Рисунок А.5 — Окно начала установки Qt Creator

5. Вам необходимо выбрать один из двух вариантов: отправлять или не отправлять статистику разработчикам (Рисунок А.6). Ваш выбор не влияет на функциональность и следует выбрать вариант исходя из собственных предпочтений. После чего следует нажать кнопку «Далее».

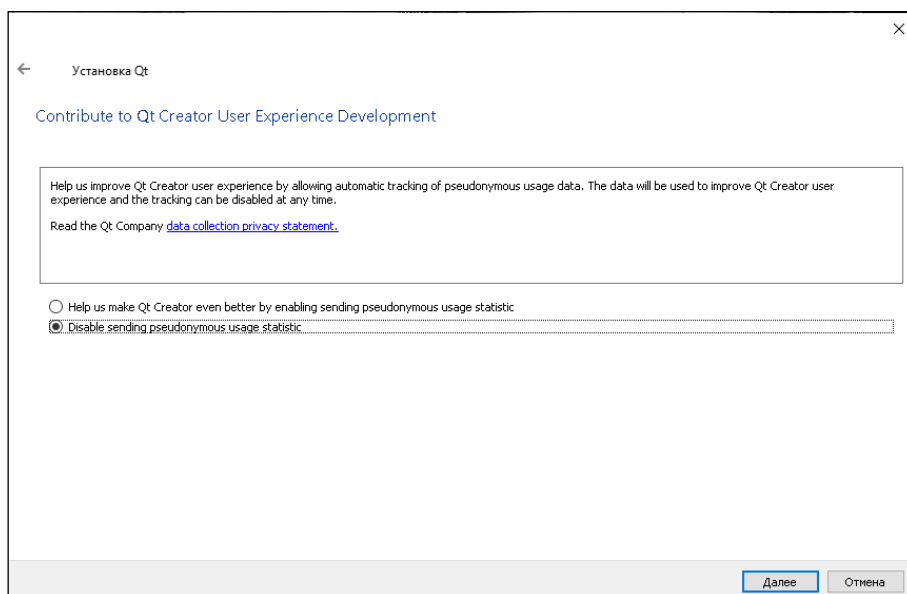


Рисунок А.6 — О предоставлении статистики использования продукта разработчикам

6. Далее необходимо указать путь для установки Qt Creator (Рисунок А.7) и расположение в меню «Пуск». Следует обратить внимание, что не следует использовать пути, содержащие символы национальных алфавитов (кириллица, иероглифы и другие), в указании путей для

установки необходимо использовать только буквы латинского алфавита, символ подчеркивания и арабские цифры.

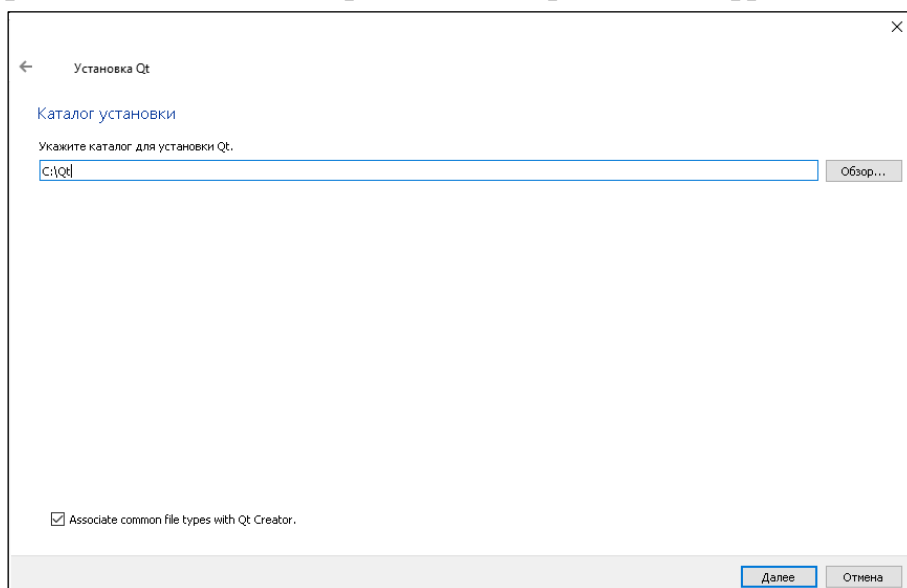


Рисунок А.7 — Указание пути установки Qt Creator

7. Для того чтобы выполнить предложенные задания необходимо установить определённый набор компонентов (Рисунок А.8). Минимально необходимый набор, для выполнения описанных выше работ, следующий⁷:

- Qt Creator 4.11.1 CDB Debugger Support.
- Debugging Tools for Windows
- MinGW 7.3.0 64-bit

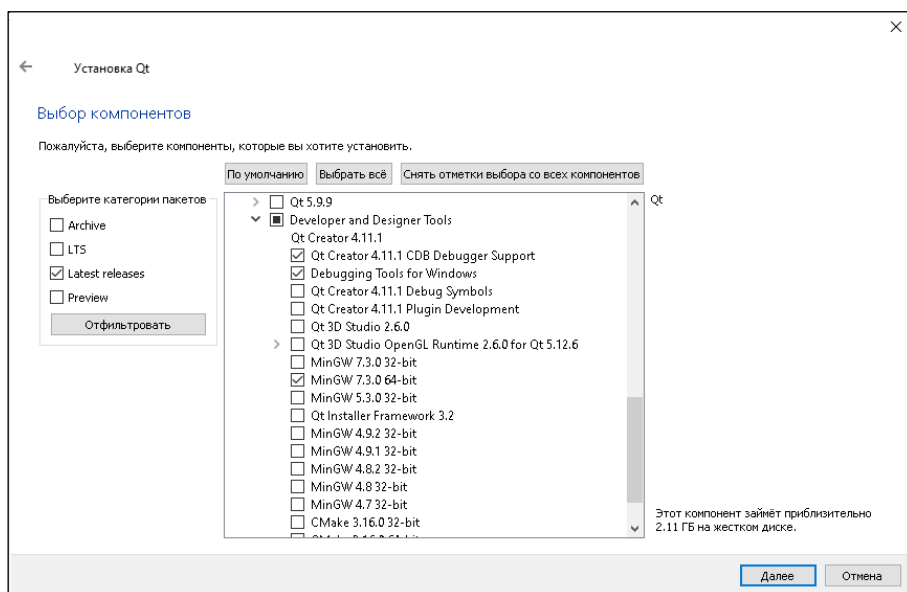


Рисунок А.8 — Выбор компонентов Qt Creator

⁷ Выбор 32-х разрядной или 64-х разрядной версии осуществляется исходя из версии Вашей операционной системы. Версии Qt Creator и компонентов могут отличаться от предложенных здесь

8. И в этом же окне находим нужную версию Qt (обычно выбирается самая свежая) и выбираем для нее советуемый выбранному ранее MinGW (Рисунок А.9):

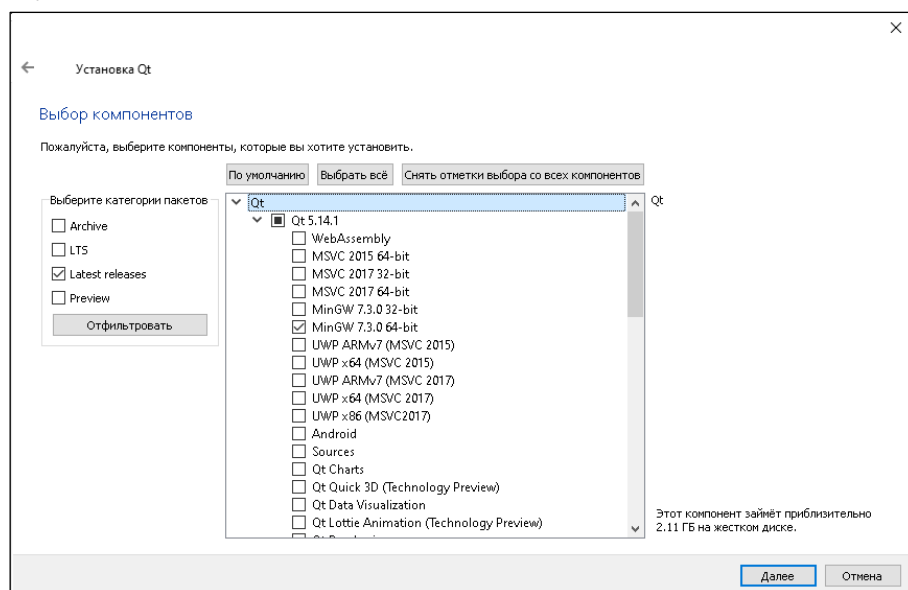


Рисунок А.9 — Минимально необходимые компоненты Qt

9. Принимаем условие лицензионного соглашения в нижней части окна (Рисунок А.10) и нажимаем «Далее» здесь и в последующих окнах (Рисунок А.11) до начала процесса установки (Рисунок А.12).

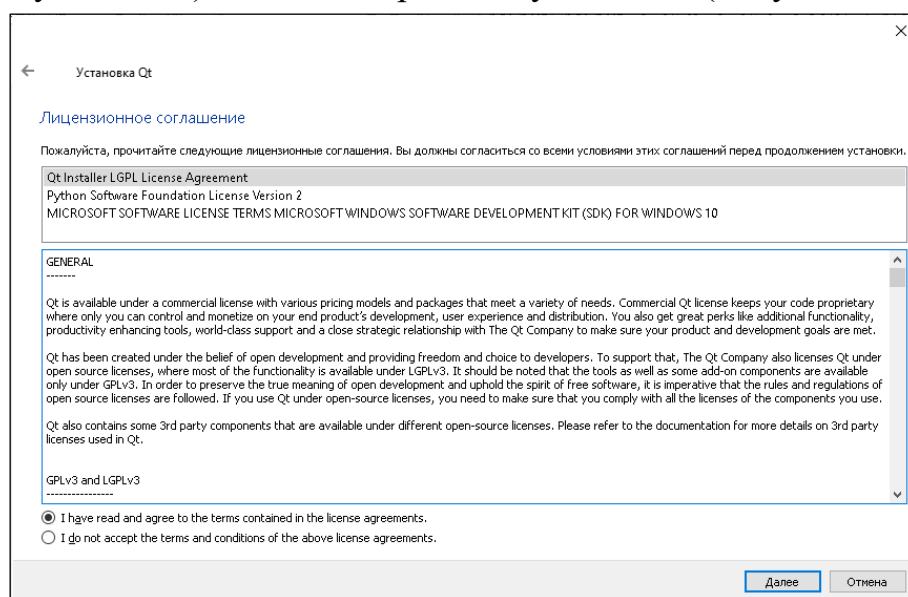


Рисунок А.10 — Условия лицензионного соглашения

Если вы не согласны с условиями лицензионного соглашения, то следует прервать установку приложения и отказаться от использования продукта.

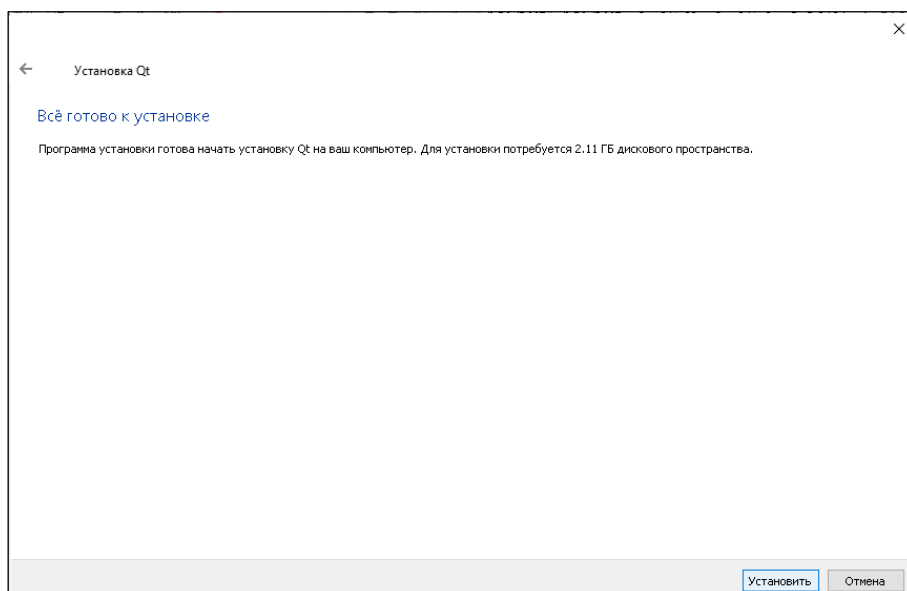


Рисунок A.11 — Информирование о готовности к установке

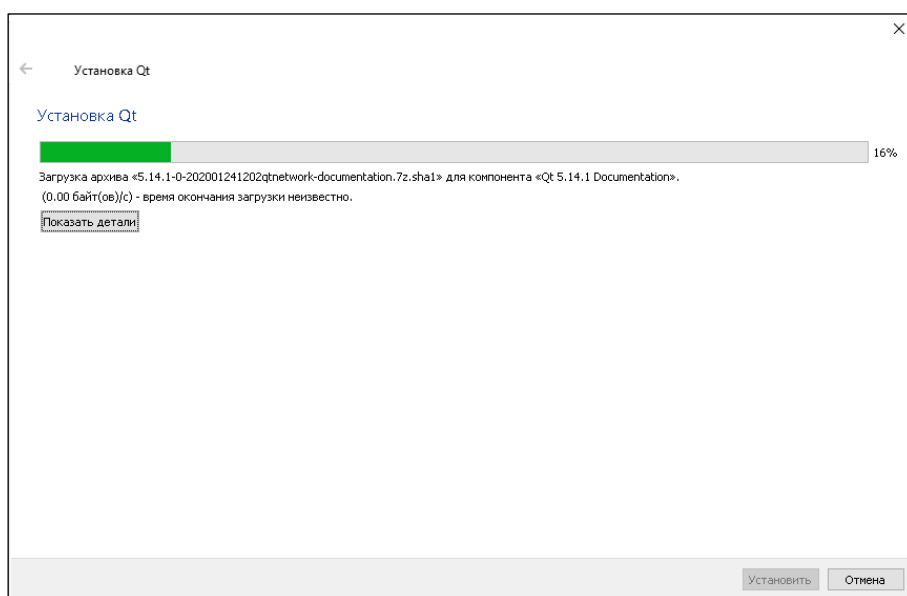


Рисунок A.15 — Процесс установки

После окончания процесса установки, нажмите кнопку «Готово». Для запуска приложения воспользуйтесь ярлыком на рабочем столе или найдите соответствующий ярлык в меню «Пуск».

В Установка Microsoft SQL Server

Перед началом установки необходимо скачать установочный пакет нужной версии с официального сайта Microsoft. Для этого используя любую поисковую систему нужно ввести запрос похожий на предложенный на рисунке ниже (Рисунок Б.1):

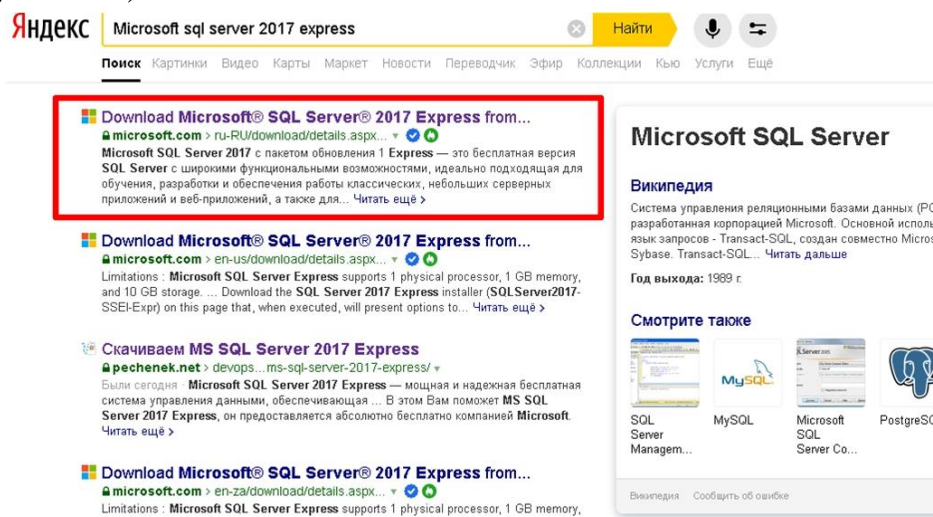


Рисунок Б.1 — Поисковый запрос

Перейдя по ссылке, нужно выбрать нужный языковой пакет и нажать кнопку «Скачать» (Рисунок Б.2):

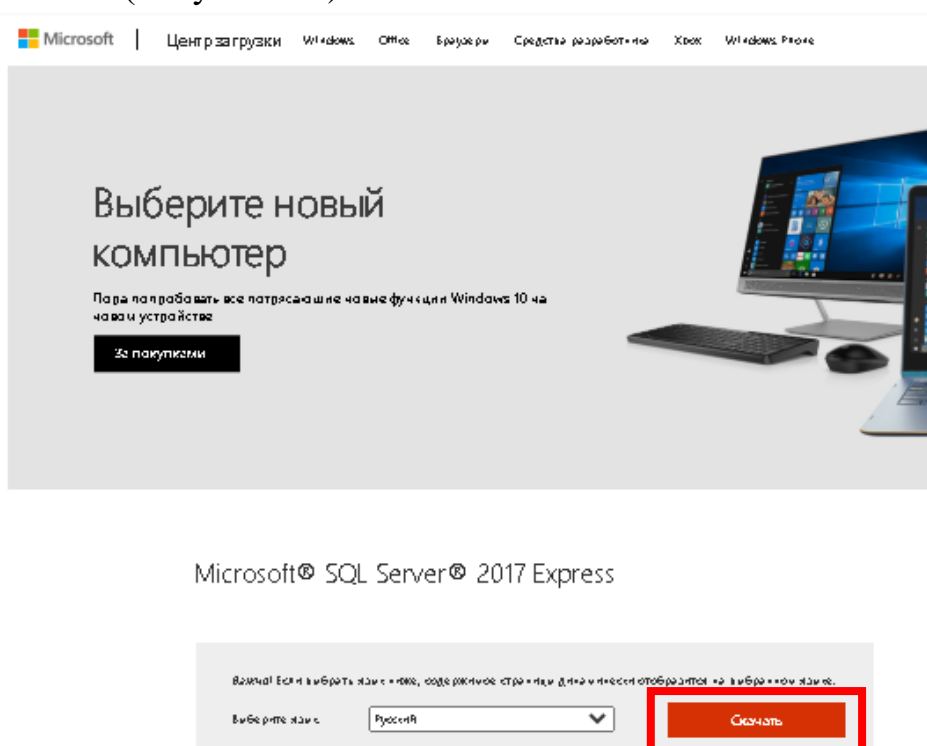


Рисунок Б.2 — Скачивание пакета установки с официального сайта

Далее, запустите загруженный файл и выберете тип установки⁸.

⁸ В большинстве случаев достаточно базовой установки, но в рамках данной инструкции будет рассмотрена установка, конфигурируемая пользователем.

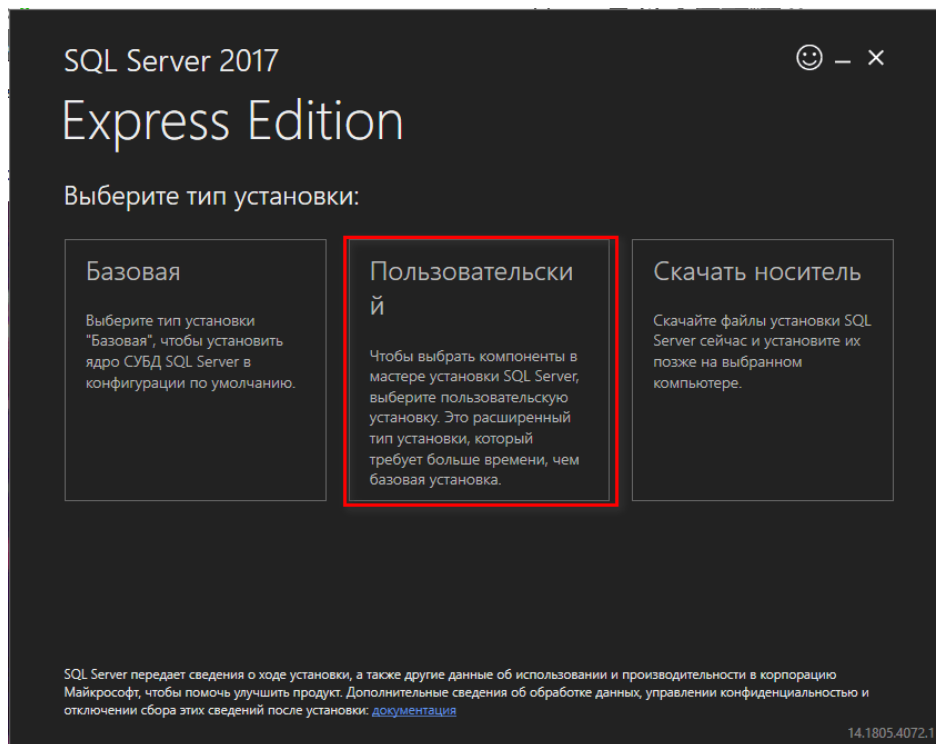


Рисунок Б.3 — Выбор типа установки

Далее необходимо указать путь, куда будут помещены файлы временные файлы, которые необходимы для установки СУБД (**Ошибка! Источник ссылки не найден.**)

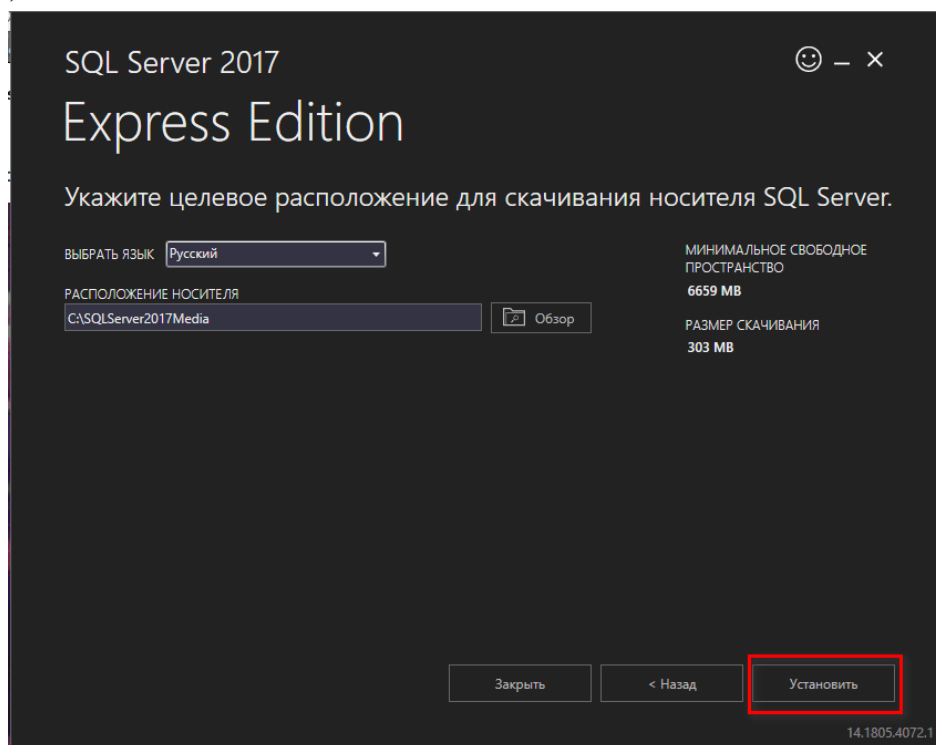


Рисунок Б.4 — Скачивание файлов установки

После нажатие на кнопку «Установить», начнется процесс загрузки установочных файлов, по окончании которой, запустится центр установки MS SQL Server (**Ошибка! Источник ссылки не найден.**)

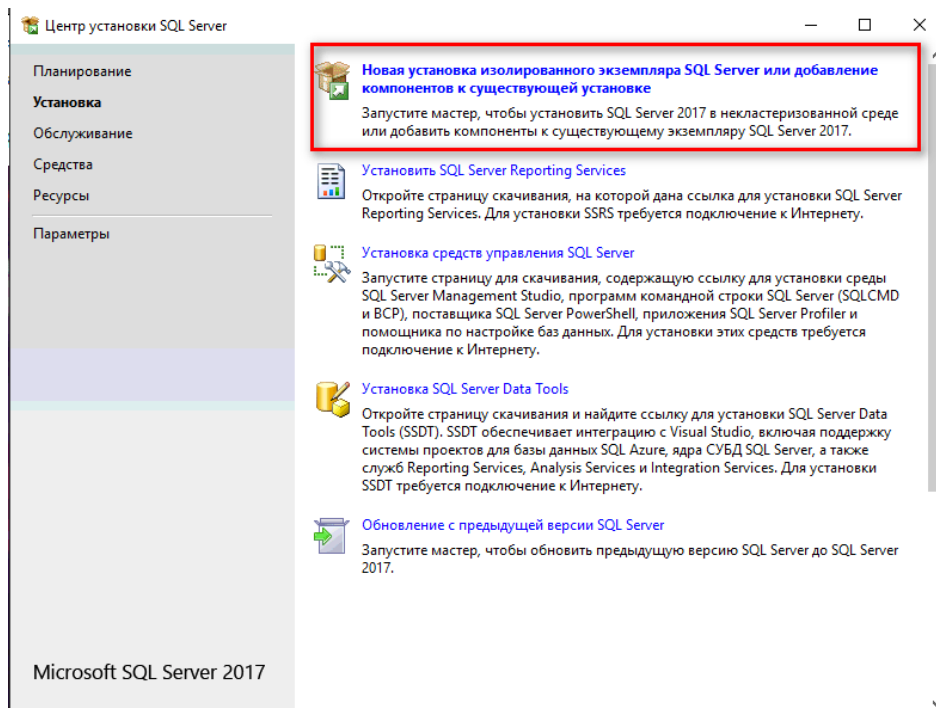


Рисунок Б.5 — Начало процесса установки

После запуска установки, через несколько шагов, установщик проверит готовность системы к установке (Рисунок Б.6)

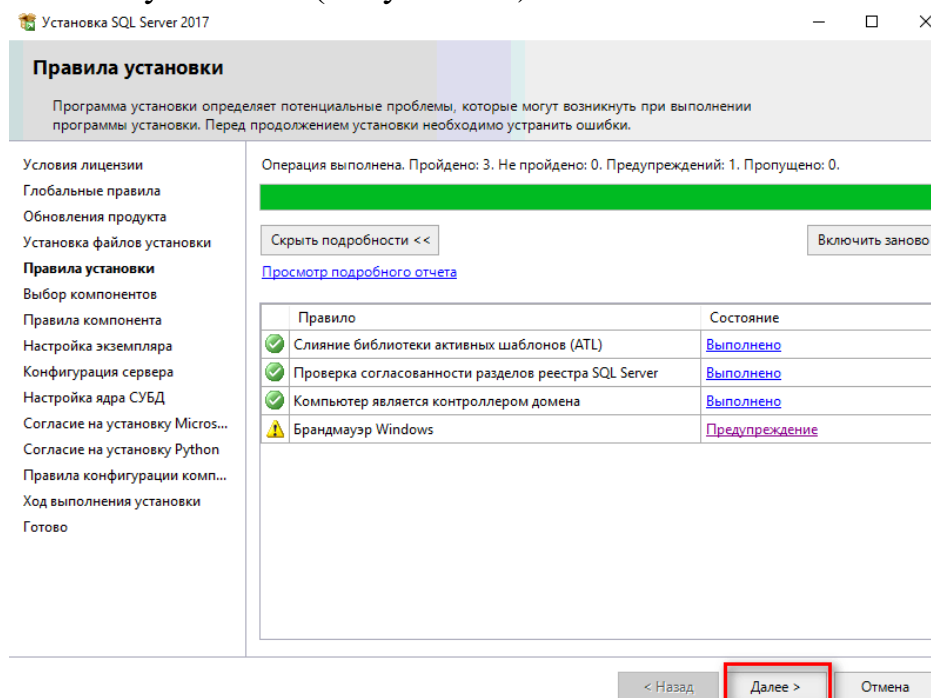


Рисунок Б.6 - Автопроверка на соответствие правилам

Далее необходимо выбрать или удалить компоненты в список компонентов для установки. Для упрощения принятия решения в правой части формы содержится подсказка (Рисунок Б.7)

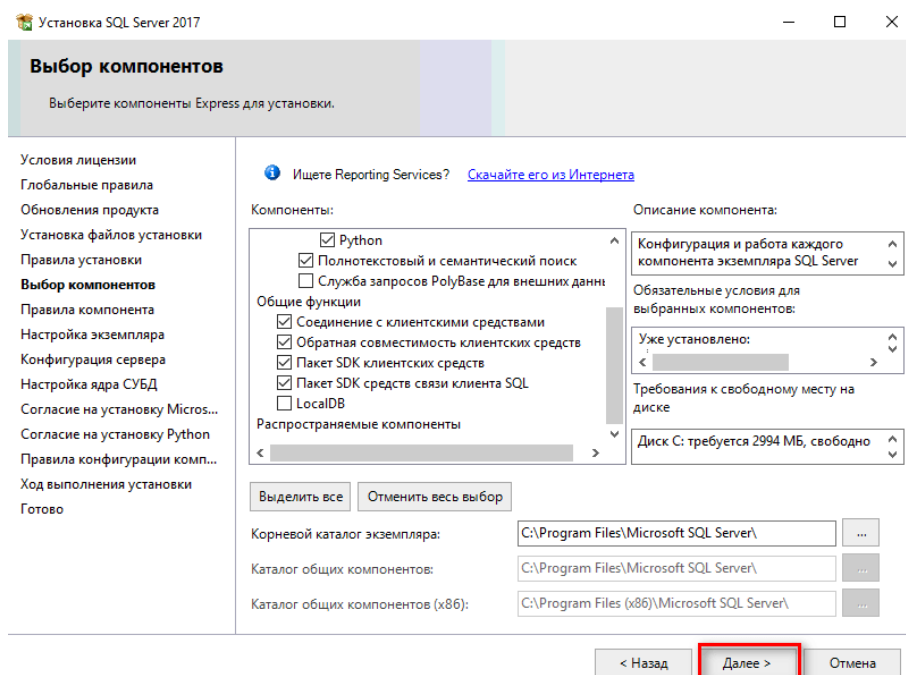


Рисунок Б.7 — Выбор компонентов для установки

После выбора необходимых компонентов необходимо настроить экземпляр сервера. В этом окне указывается Имя и идентификатор сервера (Рисунок Б.8)

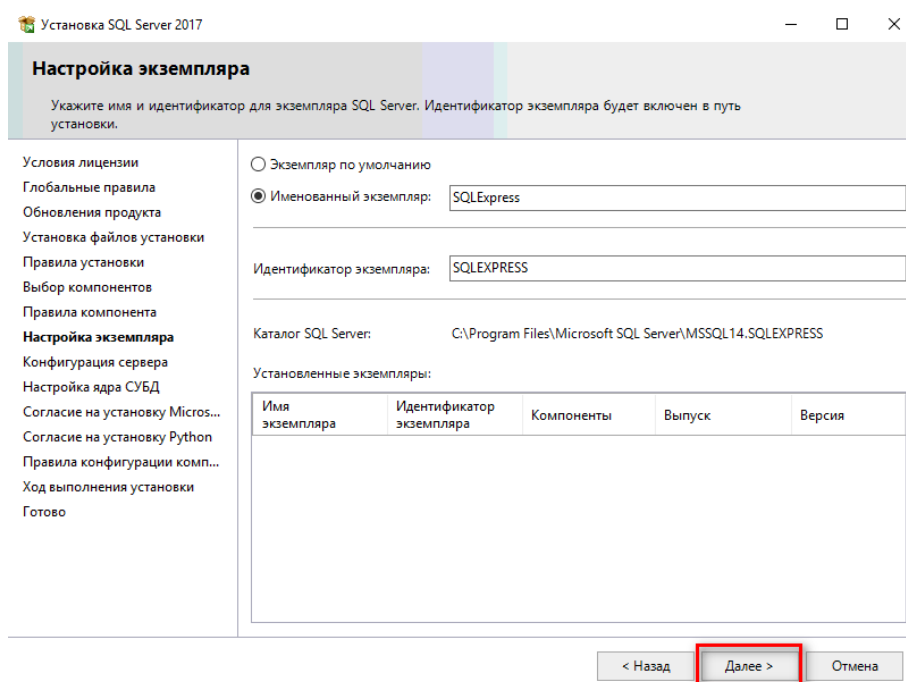


Рисунок Б.8 – Настройка экземпляра сервера

На следующем экране (**Ошибка! Источник ссылки не найден.**) можно указать сервисные аккаунты, отличные от стандартных, и предоставить право на выполнение задач обслуживания тома службе ядра СУБД SQL Server, что повысит скорость инициализации файлов, но СУБД может получить доступ к удаленному контенту. На вкладке «Параметры сортировки» можно изменить

параметры сортировки движка базы данных. На указанном примере мы предоставим привилегии, оставим по умолчанию параметры сортировки и нажмем «Далее»:

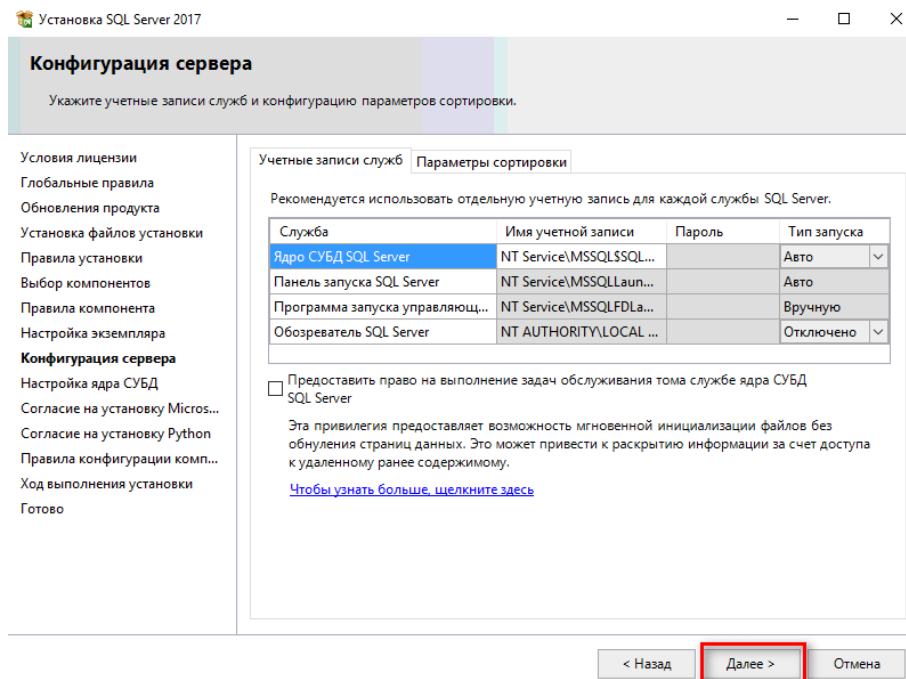


Рисунок Б.9- Настройка конфигурации сервера

Далее следует настройка ядра СУБД (Рисунок Б.10). Компания Microsoft не рекомендует менять режим проверки подлинности, однако в процессе освоения программы может потребоваться использование смешанного режима, в любом случае выбор можно изменить позже.

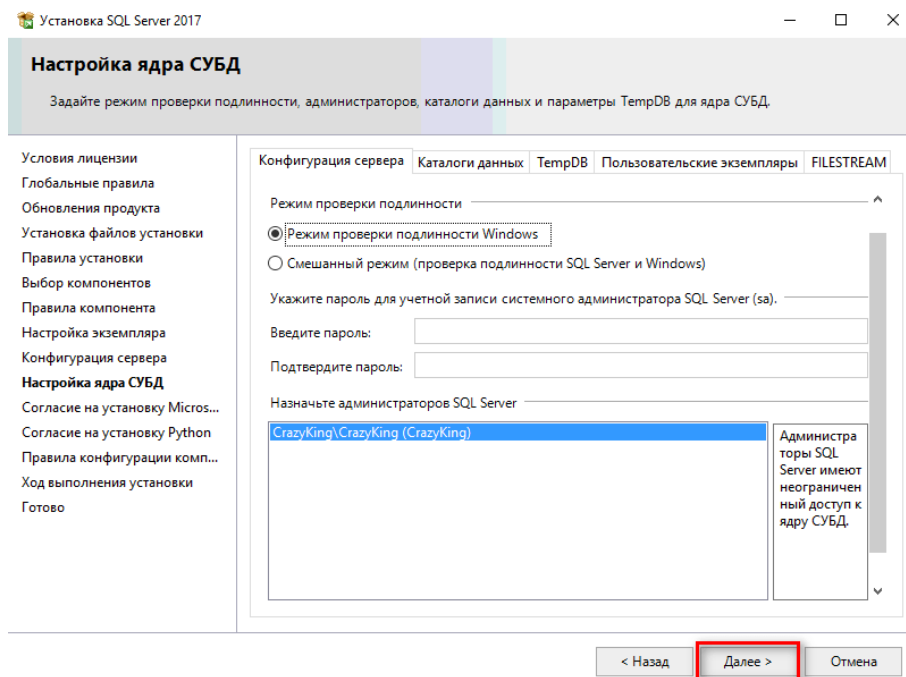


Рисунок Б.10 — Настройка ядра СУБД

С Установка Microsoft SQL Server Management Studio

Установка Microsoft SQL Server Management Studio не является обязательной, но её использование существенно упростит работу с СУБД за счёт наличия графического интерфейса, поддержки различных визуальных конструкторов.

По информации из официальной документации, SQL Server Management Studio (SSMS) — это интегрированная среда для управления любой инфраструктурой SQL, от SQL Server до баз данных SQL Azure. SSMS предоставляет средства для настройки, наблюдения и администрирования экземпляров SQL Server и баз данных. С помощью SSMS можно развертывать, отслеживать и обновлять компоненты уровня данных, используемые приложениями, а также создавать запросы и скрипты.

Для установки SSMS необходимо загрузить установочный файл с официального сайта, который можно найти с помощью любой поисковой системы по запросу приведённому на рисунке ниже или похожему запросу (Рисунок С.1).

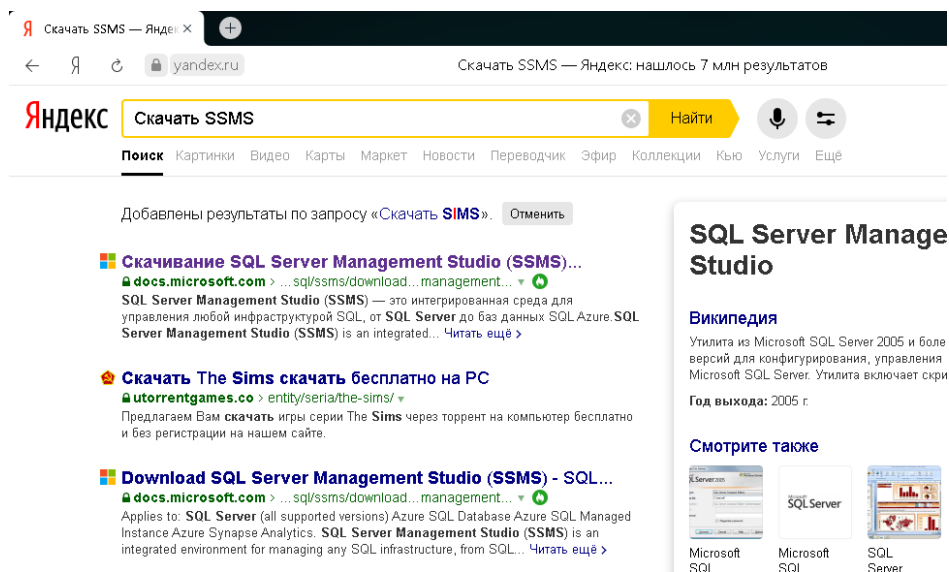


Рисунок С.1 — Поисковый запрос

На рисунке выше приведены первые три результата запроса «Скачать SSMS» и с большой долей вероятности на нужный сайт поисковая машина выдаст несколько ссылок. В приведённом примере это первый и третий результат.

Перейдя по ссылке, нужно найти раздел «Скачать SSMS», и нажать на гиперссылку ниже (Рисунок С.2). Через некоторое время начнётся загрузка установочного пакета.

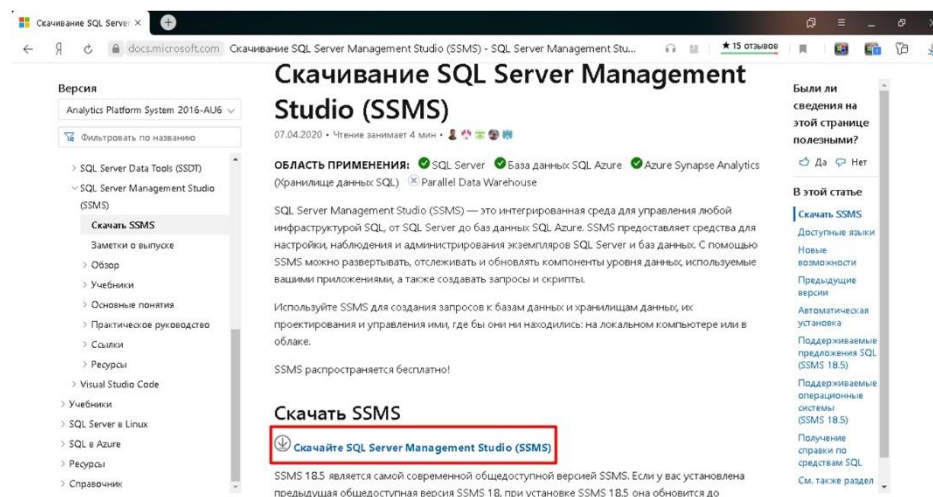


Рисунок С.2 — Загрузка SSMS с официального сайта

После окончания загрузки нужно запустить установочный файл, через некоторое время на экране появится экран приветствия и инструкции по установке среды (Рисунок С.3). Проверьте путь установки и нажмите «Установить» (“Install” в случае нелокализованной версии)

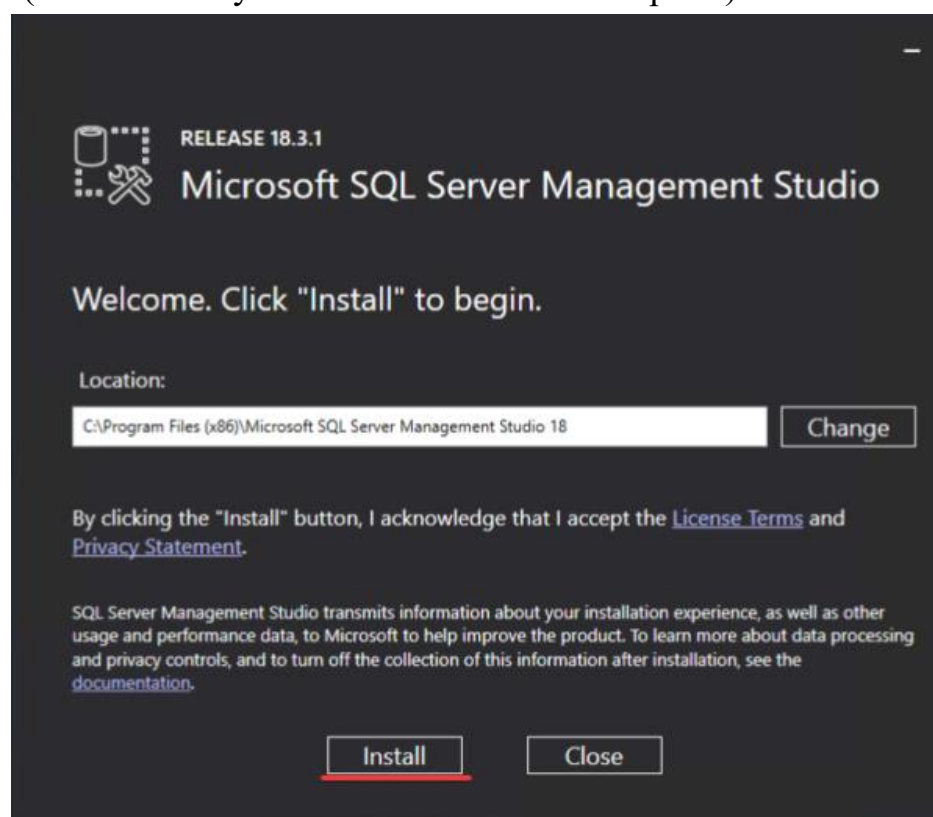


Рисунок С.3 — Начало установки

После нажатия на кнопку начнётся процесс установки, необходимо дождаться окончания его окончания. Подключение к сети Интернет при установке не требуется. Информация о ходе процесса отображается на экране (Рисунок С.4)

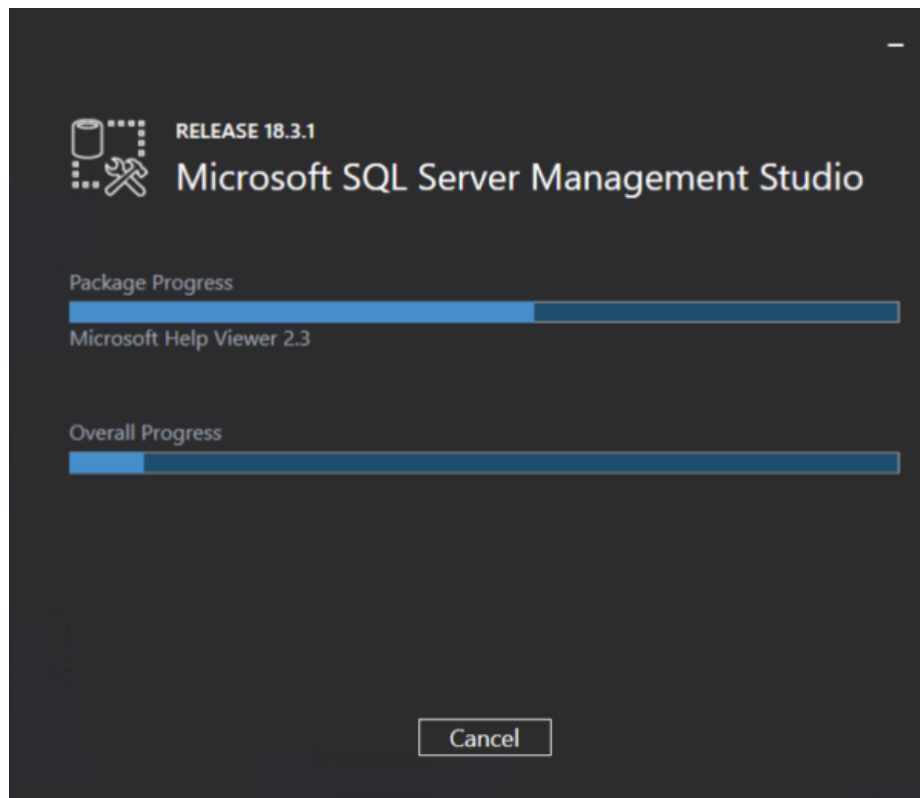


Рисунок С.4 — Установка SSMS

После окончания установки, приложение попросит перезапустить компьютер, чтобы все необходимые изменения вступили в силу. Желательно выполнить перезагрузку до первого запуска SSMS (Рисунок С.5)

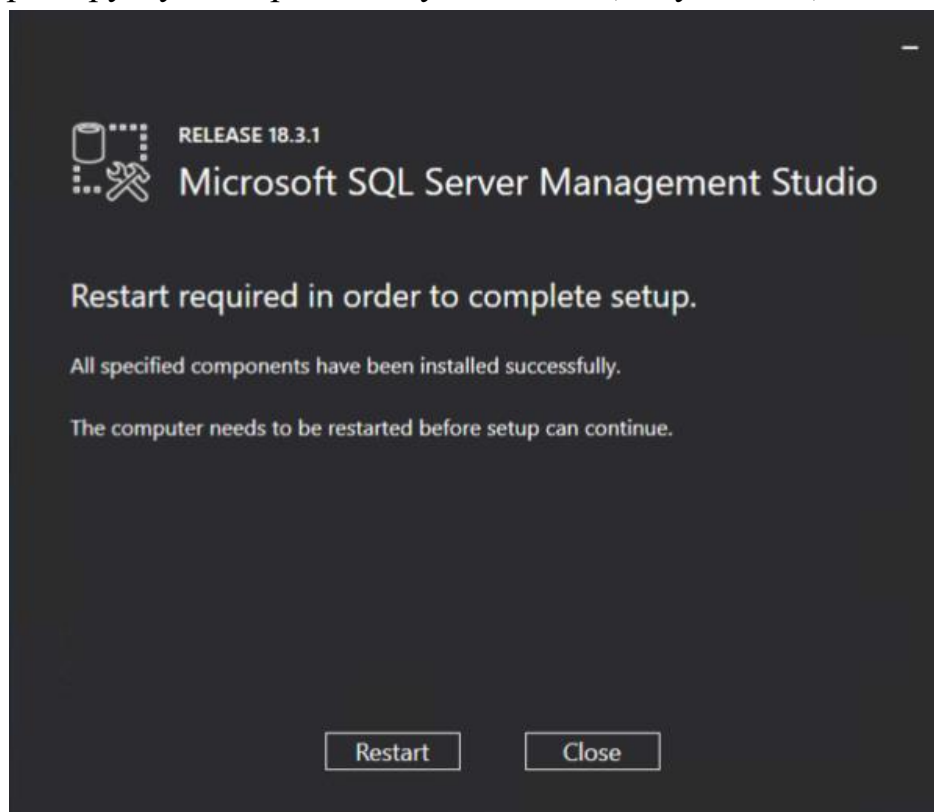


Рисунок С.5 — Просьба перезагрузить на компьютера

После перезагрузки, запустите приложение. Найти его можно в меню «Пуск» в папке Microsoft SQL Server Tools (Рисунок С.6)

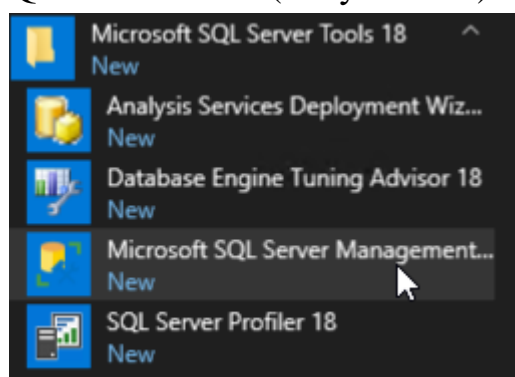


Рисунок С.6 — Расположение установленной SSMS в меню "Пуск"

SSMS является самостоятельным программным продуктом, для работы ему не требуется обязательной установки СУБД MS SQL Server. В случае удалённого подключения пользователю необходимо знать адрес сервера и имя, логин и пароль.

Независимо от того к какому серверу будет осуществляться подключение, при запуске SSMS откроется диалоговое окно подключения (Рисунок С.7).

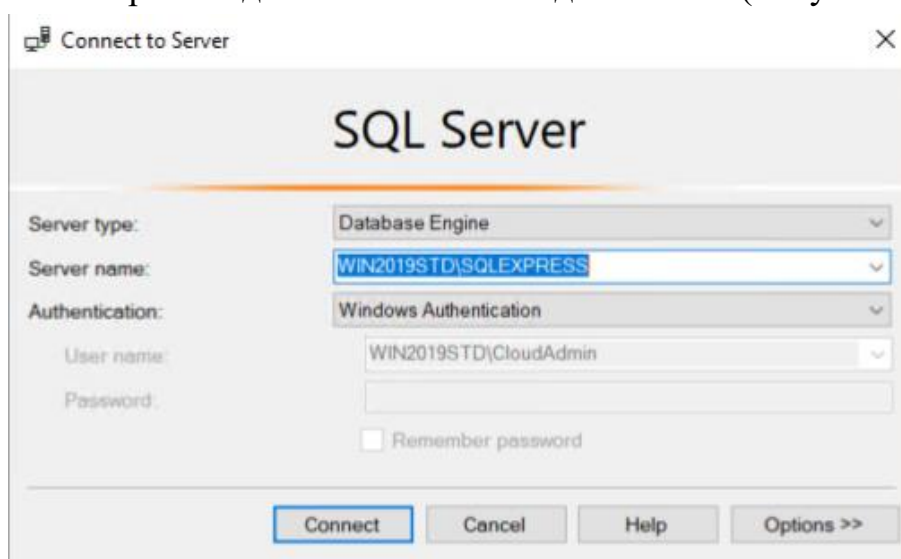


Рисунок С.7 — Окно подключения к серверу

В этом диалоговом окне необходимо указать имя экземпляра сервера (в случае локального подключения) или его адрес (в случае удалённого подключения), выбрать способ аутентификации и, при необходимости, логин и пароль.