



МИНОБРНАУКИ РОССИИ
Федеральное государственное бюджетное образовательное учреждение
высшего образования
«МИРЭА – Российский технологический университет»
РТУ МИРЭА

Институт кибербезопасности и цифровых технологий
Кафедра КБ-9 «Предметно-ориентированные информационные системы»

Лабораторная работа

по дисциплине

«Управление данными»

наименование дисциплины

Тема лабораторной работы «Подключение приложения к источнику данных»

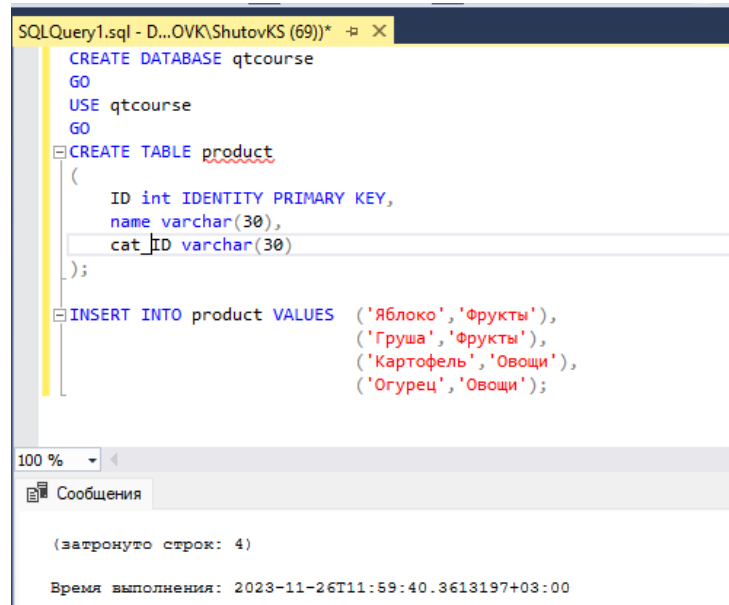
Студент группы БСБО-08-22
(учебная группа)

Шутов К.С.
Фамилия И.О.

Москва, 2023г.

Подключение разрабатываемого приложения к БД

В ходе лабораторной работы была создана база данных, к которой будет подключаться разрабатываемое приложение, и таблицу внутри этой базы, с которой приложение будет работать (см. рис 1).



```
SQLQuery1.sql - D:\OVK\ShutovKS (69))* -> X
CREATE DATABASE qtcourse
GO
USE qtcourse
GO
CREATE TABLE product
(
    ID int IDENTITY PRIMARY KEY,
    name varchar(30),
    cat_ID varchar(30)
);
INSERT INTO product VALUES ('Яблоко', 'Фрукты'),
('Груша', 'Фрукты'),
('Картофель', 'Овощи'),
('Огурец', 'Овощи');
```

100 %

Сообщения

(затронуто строк: 4)

Время выполнения: 2023-11-26T11:59:40.3613197+03:00

Рисунок 1. Создание базы данных

Далее в среде разработки Qt Creator был создан проект QtAppllication. Был добавлен новый компонент Класс формы Qt Designer. В окне Дизайн был создан пользовательский интерфейс, содержащий четыре поля для ввода и кнопку. К кнопке было привязано событие

Было создано диалоговое окно для ввода строки подключения (см. рис 2):

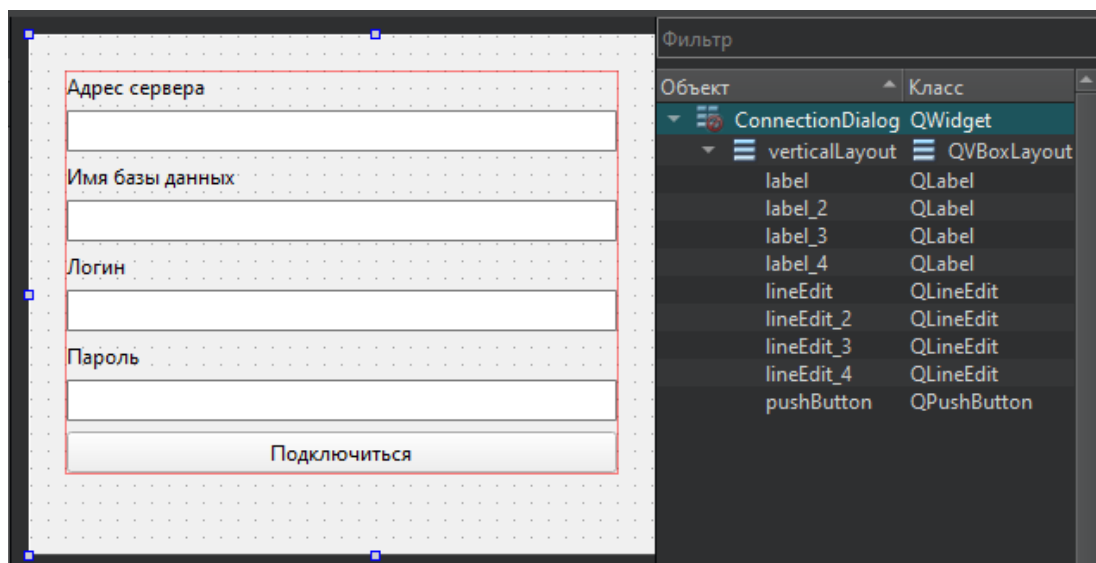


Рисунок 2. Диалоговое окно connectiondialog.ui

Написан метод для подключения к базе данных из параметров, полученных из диалогового окна (см. Листинг 1).

```
void ConnectionDialog::on_pushButton_clicked()
{
    db = QSqlDatabase::addDatabase("QODBC");
    db.setDatabaseName("DRIVER={SQL Server};SERVER=" +
        ui->lineEdit->text() +
        ";DATABASE=" +
        ui->lineEdit_2->text() +
        ";");
    db.setPassword(ui->lineEdit_3->text());
    db.setPassword(ui->lineEdit_4->text());
    msg = new QMessageBox();
    if(db.open())
    {
        msg->setText("Соединение установлено");
    }
    else
    {
        msg->setText("Соединение не установлено");
    }
    msg->show();
}
```

Листинг 1. Метод для подключения к бд

Тестовое подключение к базе данных прошло успешно (см. Рисунок 3).

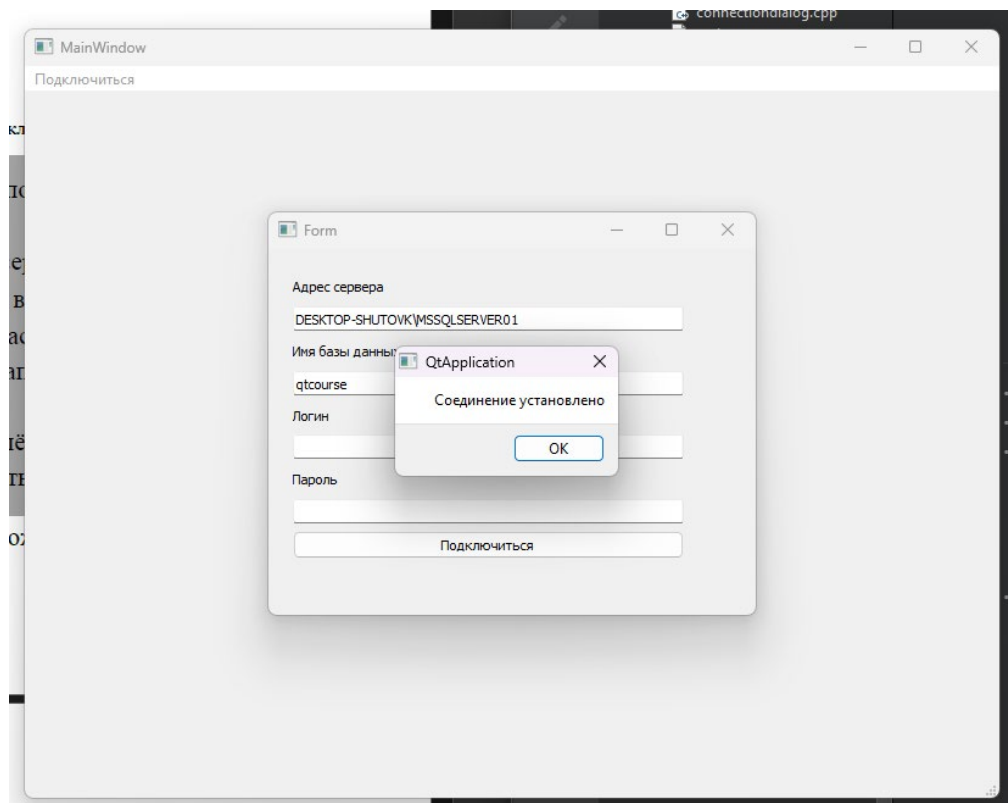


Рисунок 3. Успешное тестовое подключение

Контрольные вопросы

1. Для установления соединения с базой данных в Qt необходим класс QSqlDatabase. Этот класс предоставляет функции для создания и управления соединениями с базами данных. Вот пример использования этого класса:

```
QSqlDatabase db = QSqlDatabase::addDatabase("QSQLITE");  
db.setDatabaseName("mydatabase.db");  
if (!db.open()) {  
    // обработка ошибки  
}
```

2. Qt осуществляет взаимодействие с базой данных на трех уровнях (слоях):
 - Уровень абстракции: предоставляет классы для работы с базами данных, такие как QSqlDatabase, QSqlQuery, QSqlRecord и т.д.
 - Уровень драйвера: содержит специфические для каждой СУБД реализации этих классов.
 - Уровень подключения: управляет подключением к базе данных и выполнением SQL-запросов.
3. Назначение каждого уровня взаимодействия с базой данных:
 - Уровень абстракции: предоставляет унифицированный интерфейс для работы с различными базами данных.
 - Уровень драйвера: обрабатывает специфические для каждой СУБД детали.
 - Уровень подключения: управляет физическим подключением к базе данных и выполнением SQL-запросов.
4. ODBC (Open Database Connectivity) - это стандарт, который позволяет приложениям подключаться к различным источникам данных. Он поддерживает различные СУБД, включая MySQL, Oracle, SQL Server и другие. ODBC предоставляет API (Application Programming Interface), который позволяет приложениям подключаться к базам данных, выполнять SQL-запросы и обрабатывать результаты.
5. Для подключения к СУБД MySQL в Qt следует использовать плагин QMYSQL. Этот плагин предоставляет функции для работы с MySQL,

включая создание соединений, выполнение запросов и обработку результатов. Вот пример использования этого плагина:

```
QSqlDatabase db = QSqlDatabase::addDatabase("QMYSQL");
db.setHostName("localhost");
db.setDatabaseName("mydatabase");
db.setUserName("username");
db.setPassword("password");
if (!db.open()) {
    // обработка ошибки
}
```

Дополнительные задания

1. Добавлена кнопка в верхней части макета для отключения базы данных (см. Рисунок 4).

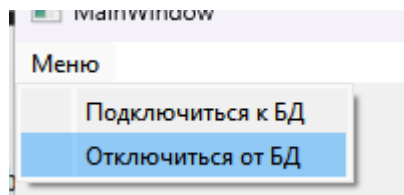


Рисунок 4. Кнопка для отключения от БД

Реализована функция разрыва текущего соединения по нажатию кнопки (см. Листинг 2)

```
void MainWindow::on_disconnect_to_database_triggered()
{
    msg = new QMessageBox();
    if(db.isOpen())
    {
        db.close();
        msg->setText(«Соединение с базой данных разорвано»);
    }
    else
    {
        msg->setText(«Нет установленного соединения с базой данных»);
    }
    msg->show();
}
```

Листинг 2. Реализация функции разрыва текущего соединения

2. Доработан метод `on_pushButton_clicked()` таким образом, чтобы в случае ошибки подключения пользователь получал информацию о том, какая именно ошибка произошла (см. Листинг 3).

```
void ConnectionDialog::on_pushButton_clicked()
{
    db->setDatabaseName("DRIVER={SQL Server};SERVER=" +
        ui->lineEdit->text() +
        ";DATABASE=" +
        ui->lineEdit_2->text() +
        ";");
}
```

```

db->setPassword(ui->lineEdit_3->text());
db->setPassword(ui->lineEdit_4->text());
msg = new QMessageBox();
if(db->open())
{
    msg->setText("Соединение установлено");
}
else
{
    QSqlError lastError = db->lastError();
    QString errorText = "Соединение НЕ установлено. Ошибка: " + lastError.text();
    msg->setText(errorText);
}
msg->show();
}

```

Листинг 3. Доработан метод on_pushButton_clicked()

3. Не удалось подключиться к удалённой базе данных (см. Рисунок 5).

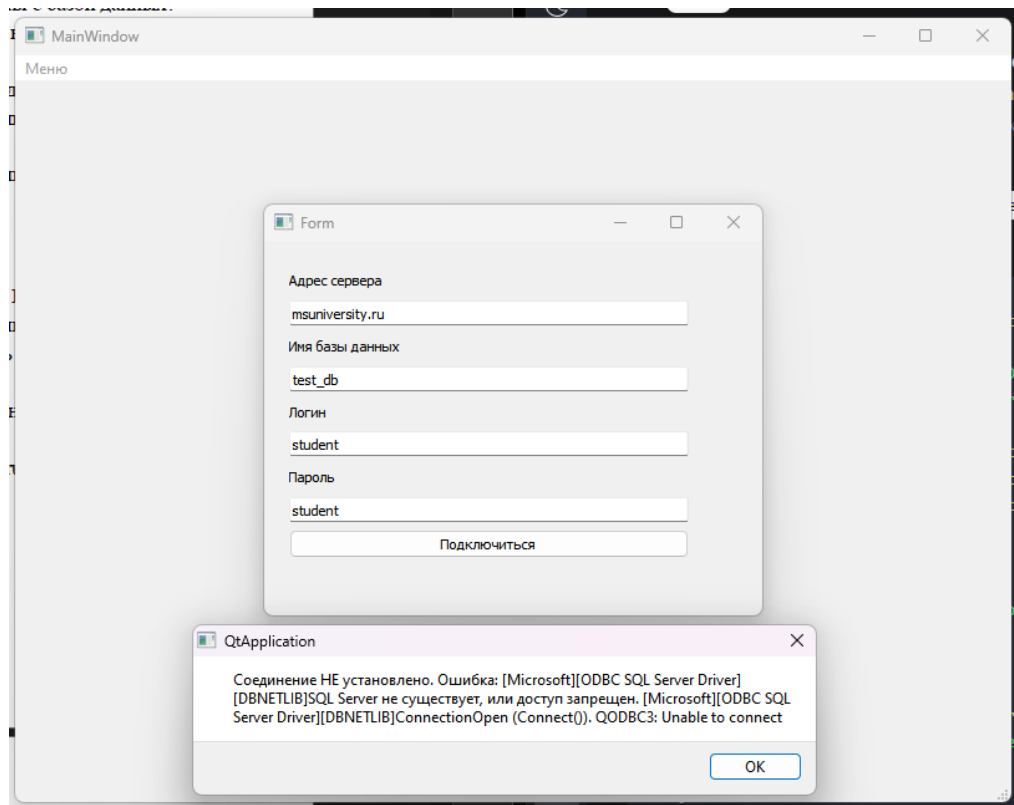


Рисунок 4. Попытка подключиться к удалённой базе данных

Вывод данных на экран

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 (см. Рисунок 5).

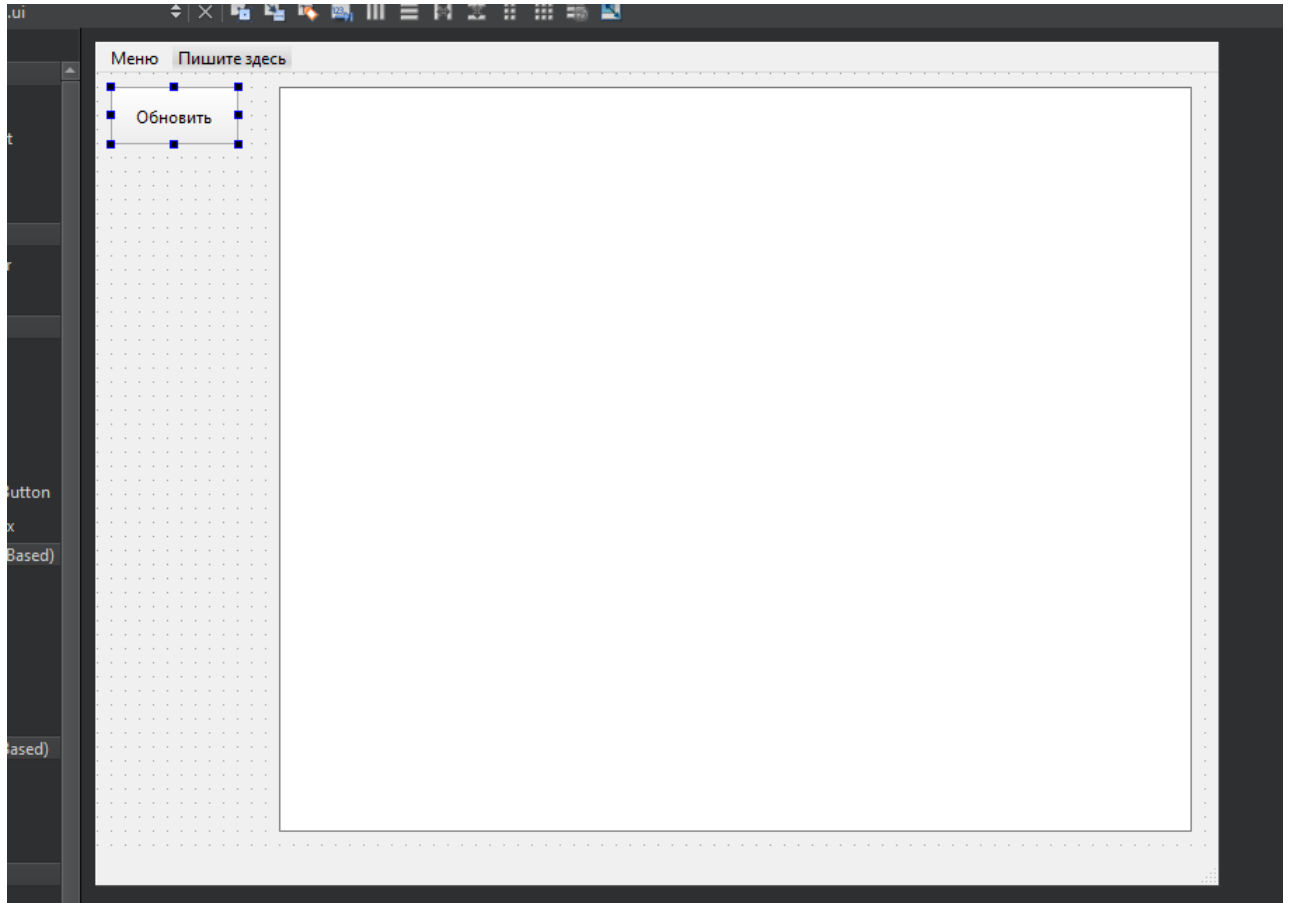


Рисунок 5. Изменено главное окно

QSqlQueryModel используется, для представления данных пользователю путём получения их из подключенной базы данных и последующим выводом в представления данных пользователю (см. Листинг 4).

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

Листинг 4. Представление данных пользователю

После успешного запуска приложения и установления подключения к БД, необходимо нажать на кнопку «Обновить», в tableView отображено содержимое таблицы product (см. Рисунок 6).

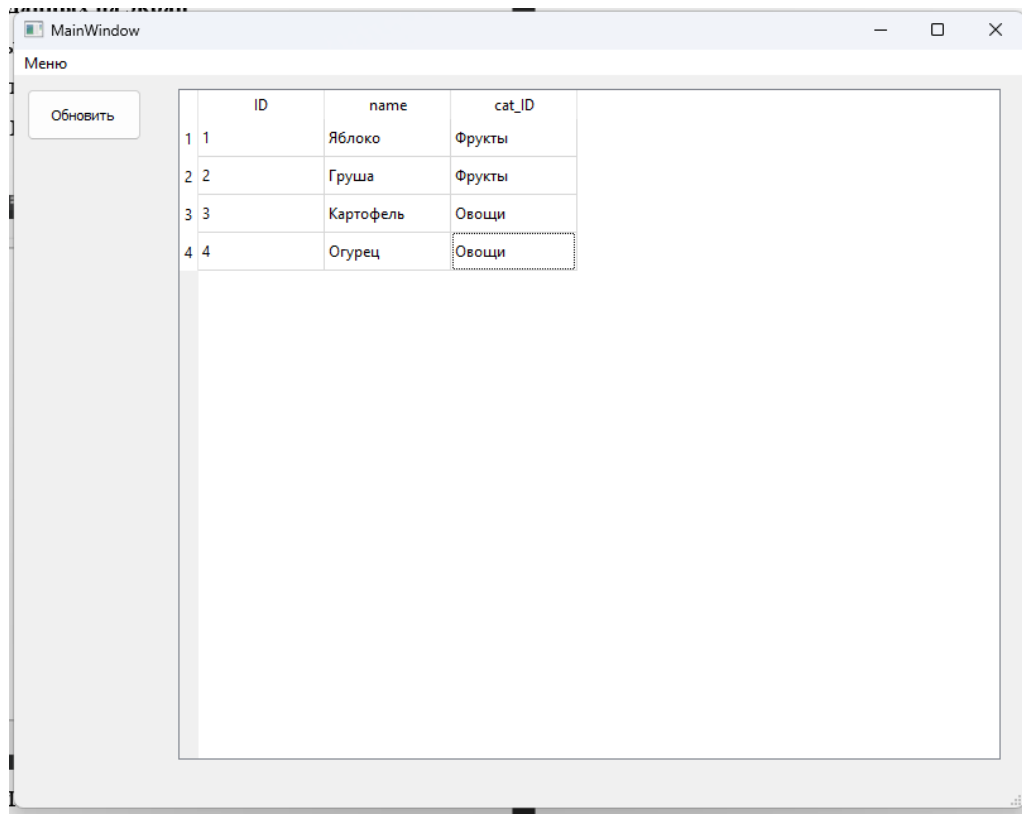


Рисунок 6. Тестирование получение данных из БД

Контрольные вопросы

1. Какой шаблон проектирования лежит в основе архитектуры модель/представление?

Архитектура модель/представление (Model/View) основана на шаблоне проектирования модель/представление/контроллер (Model/View/Controller, MVC). Однако, в отличие от MVC, в архитектуре модель/представление контроллер обычно не отделяется от представления, что упрощает структуру и увеличивает гибкость.

2. Из каких объектов она состоит?

Архитектура модель/представление состоит из трех основных компонентов:

- Модель (Model): отвечает за управление данными. Модель может быть связана с базой данных, API или любым другим источником данных.
- Представление (View): отвечает за отображение данных пользователю. Представление может быть любым интерфейсом, который отображает данные модели.

- Контроллер (Controller): в архитектуре модель/представление, контроллер обычно объединяется с представлением и отвечает за обработку пользовательских взаимодействий и обновление модели и представления в соответствии с этими взаимодействиями.

3. В чем отличие архитектуры модель/представление от шаблона MVC?

Основное отличие архитектуры модель/представление от шаблона MVC заключается в том, что в архитектуре модель/представление контроллер обычно не отделяется от представления. Это упрощает структуру и увеличивает гибкость, так как позволяет изменять поведение представления без необходимости изменять модель. В MVC же, контроллер и представление обычно тесно связаны, что может усложнить изменение поведения представления.

4. Найти в документации Qt способ создания модели для выборки данных из нескольких таблиц с возможностью редактирования?

Для создания модели, которая может выбирать данные из нескольких таблиц, можно использовать класс `QAbstractTableModel` или `QAbstractItemModel` в Qt. Эти классы предоставляют методы для работы с табличными данными и поддерживают возможность редактирования данных. Для каждой таблицы, из которой нужно выбрать данные, вы можете создать отдельный экземпляр модели. Затем вы можете использовать `QSortFilterProxyModel` для объединения данных из нескольких моделей в одну модель, которую затем можно использовать в представлении.

5. Какие ещё классы моделей поддерживаются Qt?

В Qt поддерживаются различные классы моделей, включая:

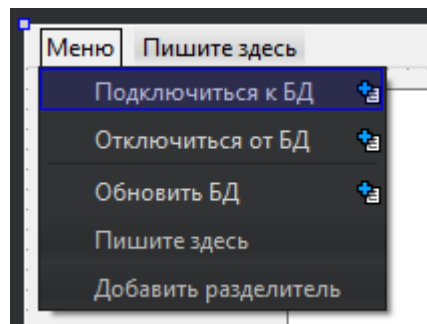
- `QAbstractItemModel`: Базовый класс для моделей, которые представляют данные в виде иерархической структуры.
- `QAbstractTableModel`: Базовый класс для моделей, которые представляют данные в виде таблицы.
- `QAbstractListModel`: Базовый класс для моделей, которые представляют данные в виде списка.

- QStringListModel: Модель, которая представляет данные в виде списка строк.
- QStandardItemModel: Модель, которая представляет данные в виде списка стандартных элементов.
- ListModel: Модель, которая представляет данные в виде списка элементов в QML.

Дополнительное задание

1. Реализовать вызов функции обновления таблицы через главное меню приложения.

Добавлена кнопка обновить в главное меню.



Написан метод для кнопки.

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

2. Внести изменения в свойства объекта tableView которые спрячут от пользователя ключевой столбец таблицы product (при этом, его нельзя исключать его из запроса).

Добавлена код при выводе информации из базы данных который скрывает 0 колонку.

```
ui->tableView->setColumnHidden(0, true);
```

	name	cat_ID
1	Яблоко	Фрукты
2	Груша	Фрукты
3	Картофель	Овощи
4	Огурец	Овощи

3. Добавить возможность фильтрации строк таблицы по одному признаку.

Признак студент выбирает самостоятельно. Результат работы фильтра проверить на таблице, содержащей не менее 15 записей.

```
void MainWindow::on_pushButton_filter_clicked()
{
    QSortFilterProxyModel *proxyModel = new QSortFilterProxyModel(this);
    proxyModel->setSourceModel(qmodel);
    ui->tableView->setModel(proxyModel);

    proxyModel->setFilterKeyColumn(2);
    proxyModel->setFilterRegExp(ui->lineEdit_filter->text());
}
```

Функции добавления, изменения и удаления записей

Создан новый экран для добавления новой строки в таблицу БД (см. Рисунок 7).

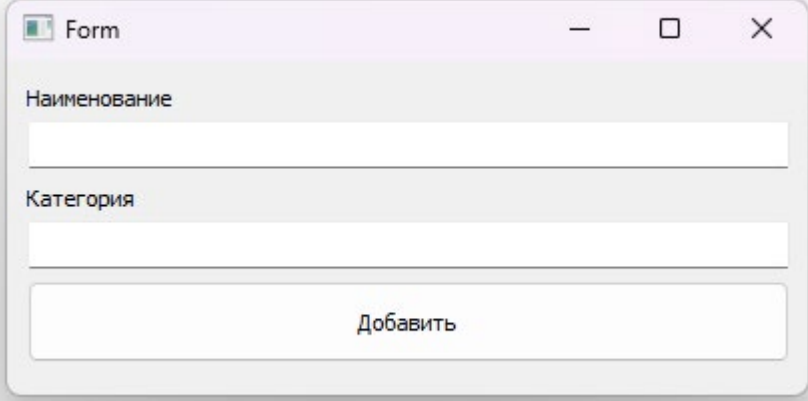


Рисунок 7. Экран добавления новой строки в таблицу БД

Написан метод для добавления значений в базу данных (см. Листинг 5).

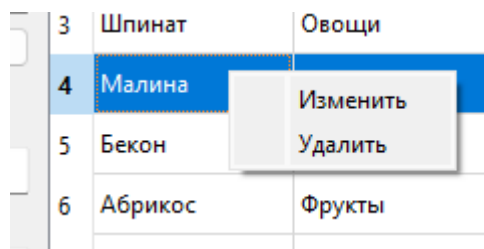
```
void AddDialog::on_pushButton_add_clicked()
{
    QSqlQuery *query = new QSqlQuery();
    query->prepare("INSERT INTO product VALUES"
        "(:name,:category)");
    query->bindValue(":name", ui->lineEdit_name->text());
    query->bindValue(":category", ui->lineEdit_category->text());

    QMessageBox*mess= new QMessageBox();
    if(!query->exec())
    {
        mess->setText("Запрос составлен неверно!");
        mess->show();
    }

    close();
}
```

Листинг 5. Добавление новой строчки в БД

Добавлены кнопки контекстного меню для таблицы просмотра, для быстрого взаимодействия (см. Рисунок 8).



3	Шпинат	Овощи
4	Малина	
5	Бекон	
6	Абрикос	Фрукты

Рисунок 8. Кнопки контекстного меню

Добавлен экран для формы редактирования ячейки базы данных при вызове через контекстное меню (см. Рисунок 9).

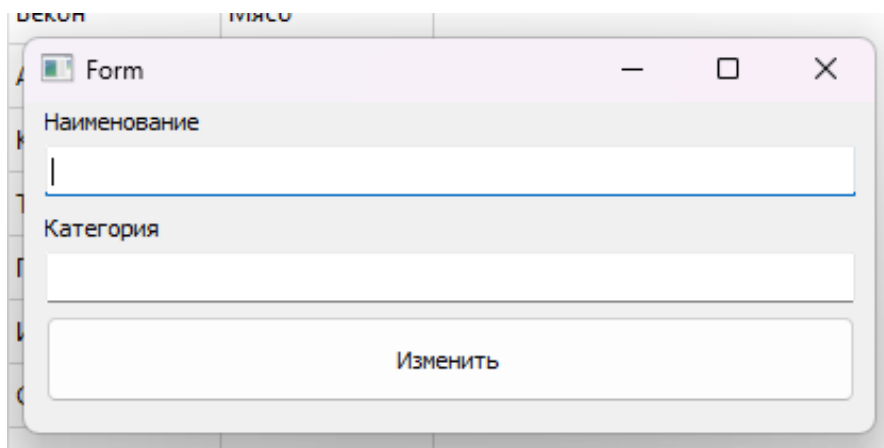
A screenshot of a Windows-style dialog box titled 'Form'. It has a light gray background and a title bar with standard minimize, maximize, and close buttons. Inside the dialog, there are two text input fields. The first is labeled 'Наименование' (Name) and the second is labeled 'Категория' (Category). Below these fields is a single button labeled 'Изменить' (Edit).

Рисунок 9. Экран редактирования

Добавлены поля для редактирования значений базы данных (см. Рисунок 10).

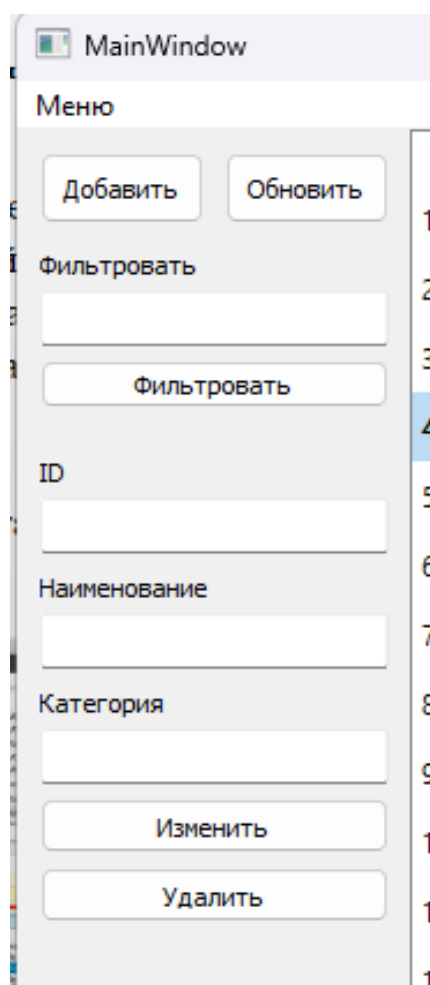
A screenshot of the 'MainWindow' application interface. The window has a title bar with the text 'MainWindow'. Below the title bar, there is a section labeled 'Меню' (Menu) containing two buttons: 'Добавить' (Add) and 'Обновить' (Update). Below this is a section labeled 'Фильтровать' (Filter) with a text input field and a 'Фильтровать' button. Further down, there are three input fields labeled 'ID', 'Наименование', and 'Категория'. At the bottom of the main content area are two buttons: 'Изменить' (Edit) and 'Удалить' (Delete). To the right of the main content area, there is a vertical list of numbers: 1, 2, 3, 4, 5, 6, 7, 8, 9, 1, 1, 1. The number 4 is highlighted in blue.

Рисунок 10. Поля для редактирования значений БД

Контрольные вопросы

1. Основное отличие QSqlQuery и QSqlQueryModel

QSqlQuery и QSqlQueryModel оба являются частью модуля Qt SQL и используются для взаимодействия с базами данных, они служат разным целям.

QSqlQuery используется для выполнения SQL-запросов к базе данных. Этот класс предоставляет интерфейс для выполнения запросов и получения результатов. Вы можете использовать QSqlQuery для выполнения любых SQL-запросов, таких как SELECT, INSERT, UPDATE и DELETE.

QSqlQueryModel, с другой стороны, является моделью данных, которая используется для представления результатов SQL-запроса в виде модели данных, которую можно использовать в виджетах Qt, таких как QTableView и QListView. QSqlQueryModel автоматически обновляет свою модель данных при изменении результатов SQL-запроса.

2. Инструкции, которые можно выполнить с помощью QSqlQuery

С помощью QSqlQuery можно выполнять любые SQL-запросы, которые поддерживаются используемым драйвером базы данных. Это включает в себя следующие типы запросов:

- SELECT: используется для выбора данных из базы данных.
- INSERT: используется для вставки новых данных в базу данных.
- UPDATE: используется для обновления существующих данных в базе данных.
- DELETE: используется для удаления данных из базы данных.

3. Метод для чтения данных из результирующей выборки

Для чтения данных из результирующей выборки, количество записей в которой заранее не известно, можно использовать следующий метод:

```
void readData(QSqlQuery &query) {
    while (query.next()) {
        // Читаем данные из каждой строки
        QString name = query.value("name").toString();
        int age = query.value("age").toInt();
        // ...
    }
}
```

В этом методе используется метод `next()` класса `QSqlQuery`, который перемещает курсор к следующей строке в результирующей выборке. Если все строки были прочитаны, `next()` возвращает `false`, и цикл завершается.

4. Сценарии работы с `QSqlQuery`

Сценарии работы с `QSqlQuery` могут включать следующее:

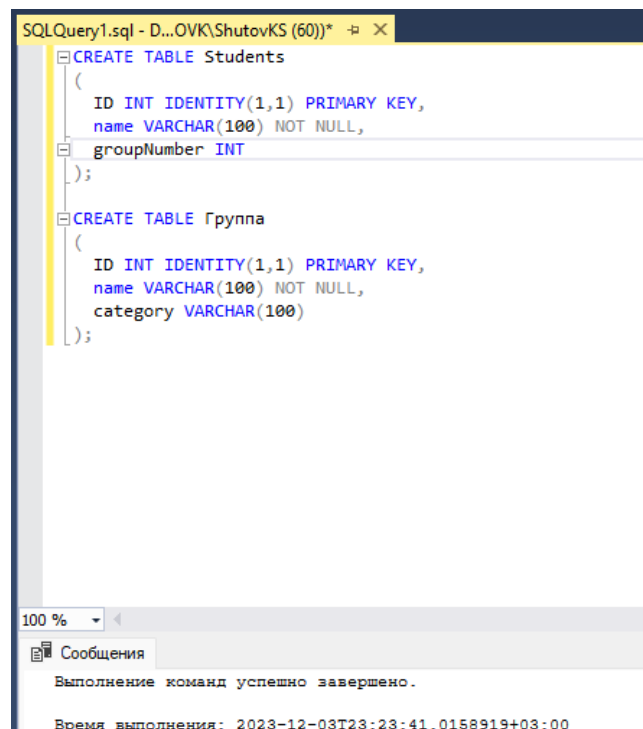
- Выполнение SQL-запроса: Это можно сделать с помощью метода `exec()`.
- Получение результатов запроса: Это можно сделать с помощью метода `next()`.
- Обработка ошибок: Это можно сделать с помощью метода `lastError()`.
- Работа с параметрами запроса: Это можно сделать с помощью методов `bindValue()` и `addBindValue()`.

Все эти операции могут быть выполнены в любом порядке и любое количество раз, в зависимости от требований вашего приложения.

Дополнительное задание

2. Создать таблицы

Созданы новые таблицы в базе данных.



```
SQLQuery1.sql - D:\OVK\ShutovKS (60)) *  X
CREATE TABLE Students
(
    ID INT IDENTITY(1,1) PRIMARY KEY,
    name VARCHAR(100) NOT NULL,
    groupNumber INT
);

CREATE TABLE Группа
(
    ID INT IDENTITY(1,1) PRIMARY KEY,
    name VARCHAR(100) NOT NULL,
    category VARCHAR(100)
);

Сообщения
Выполнение команд успешно завершено.
Время выполнения: 2023-12-03T23:23:41.0158919+03:00
```

Вывод данных из приложения

Добавлены две кнопки для экспорта таблицы базы данных в файл pdf или word (см. Рисунок 11).

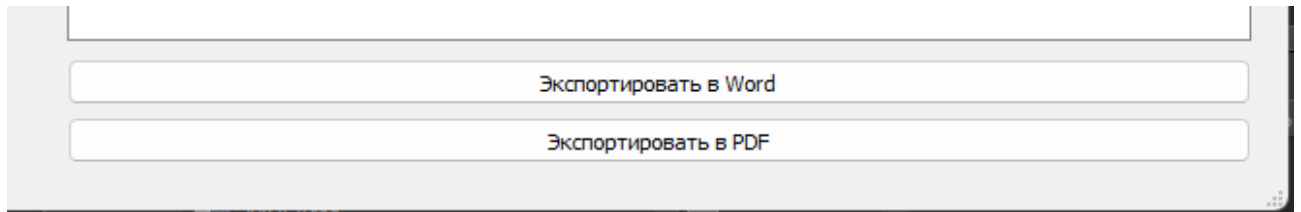


Рисунок 11. Кнопки экспорта

Для экспорта в word добавлен ещё один экран, где происходит выбор пути к файлу, и кнопка сформировать (см. Рисунок 12).

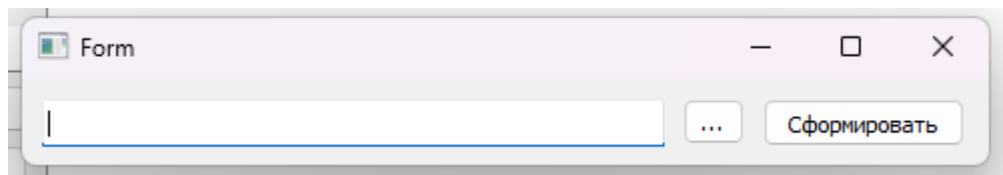


Рисунок 12. Экран для экспорта в word

Пример экспорта таблицы из базы данных в pdf представлен на рисунке 13.

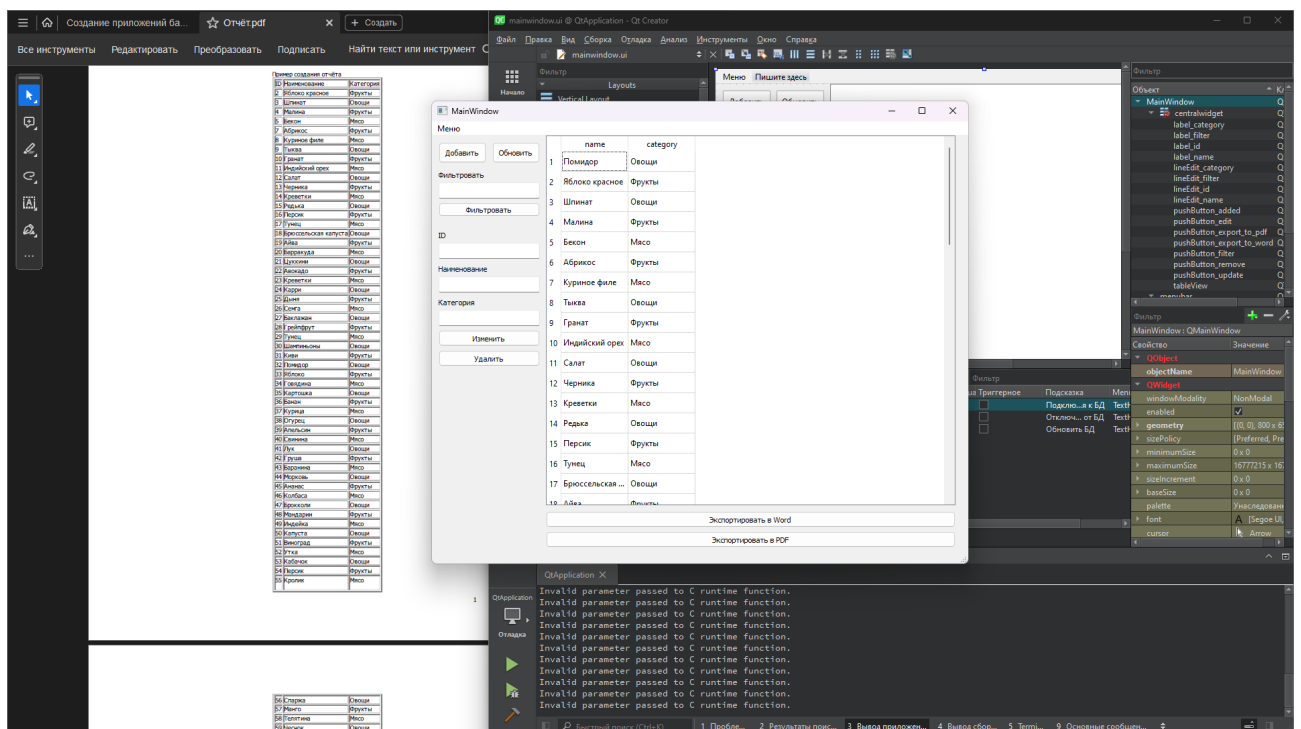


Рисунок 13. Пример экспорта

Контрольные вопросы

1. Простой отчёт:

Простой отчёт — это документ, предназначенный для передачи информации о конкретном событии, процессе или состоянии дел. Он обычно содержит фактическую информацию и может быть подготовлен в текстовой или табличной форме. Пример простого отчёта может быть отчёт о выполненной задаче, о результатах исследования или о статусе проекта.

2. Другие виды отчётов:

- Аналитический отчёт: Содержит анализ данных, их интерпретацию и рекомендации на основе полученных результатов.
- Финансовый отчёт: Сводная информация о финансовом состоянии компании или организации.
- Прогнозный отчёт: Предсказание будущих событий, трендов или результатов на основе текущих данных.
- Промежуточный отчёт: Предварительный отчёт об уже достигнутых результатах до завершения процесса или проекта.

3. Технология экспорта данных в MS Word:

Технология, обычно используемая для экспорта данных в MS Word, может включать в себя использование стандартных форматов, таких как RTF (Rich Text Format) или DOCX (формат документов Microsoft Word). Также, для автоматизации процесса экспорта, можно применять библиотеки или API, такие как Python-docx для языка программирования Python.

4. ActiveX в Qt:

В Qt фреймворк ActiveX обычно используется для взаимодействия с элементами управления ActiveX (OLE Control) внутри Qt-приложений. ActiveX — это технология, разработанная Microsoft, позволяющая встраивать объекты в документы или другие объекты. В контексте Qt, ActiveX может быть использован для встраивания элементов управления, созданных в других средах разработки (например, в Microsoft Visual Studio), в приложения Qt. Это облегчает интеграцию Qt-приложений с другими технологиями, особенно при разработке под Windows.

Дополнительные функции приложения

В таблицу продукт добавлен новый столбец, где будут храниться изображения (см. Рисунок 14).

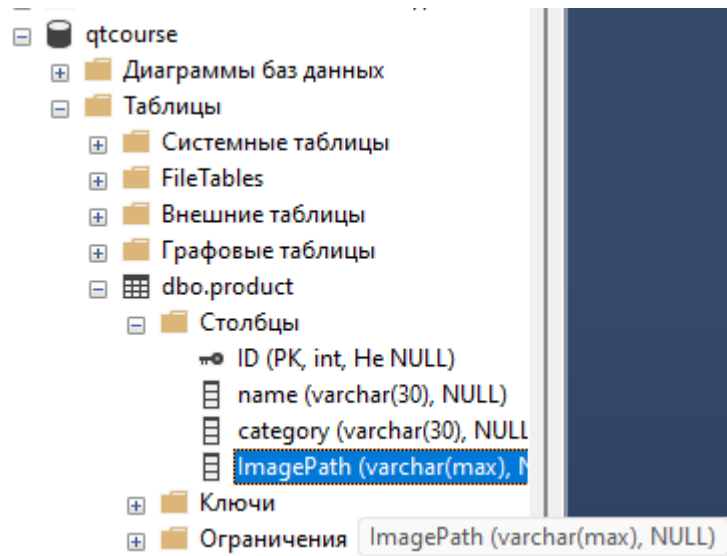


Рисунок 14. Добавление нового столбца

Добавлена новая таблица и в неё занесены случайные числа в заданном диапазоне (см. Рисунок 15).

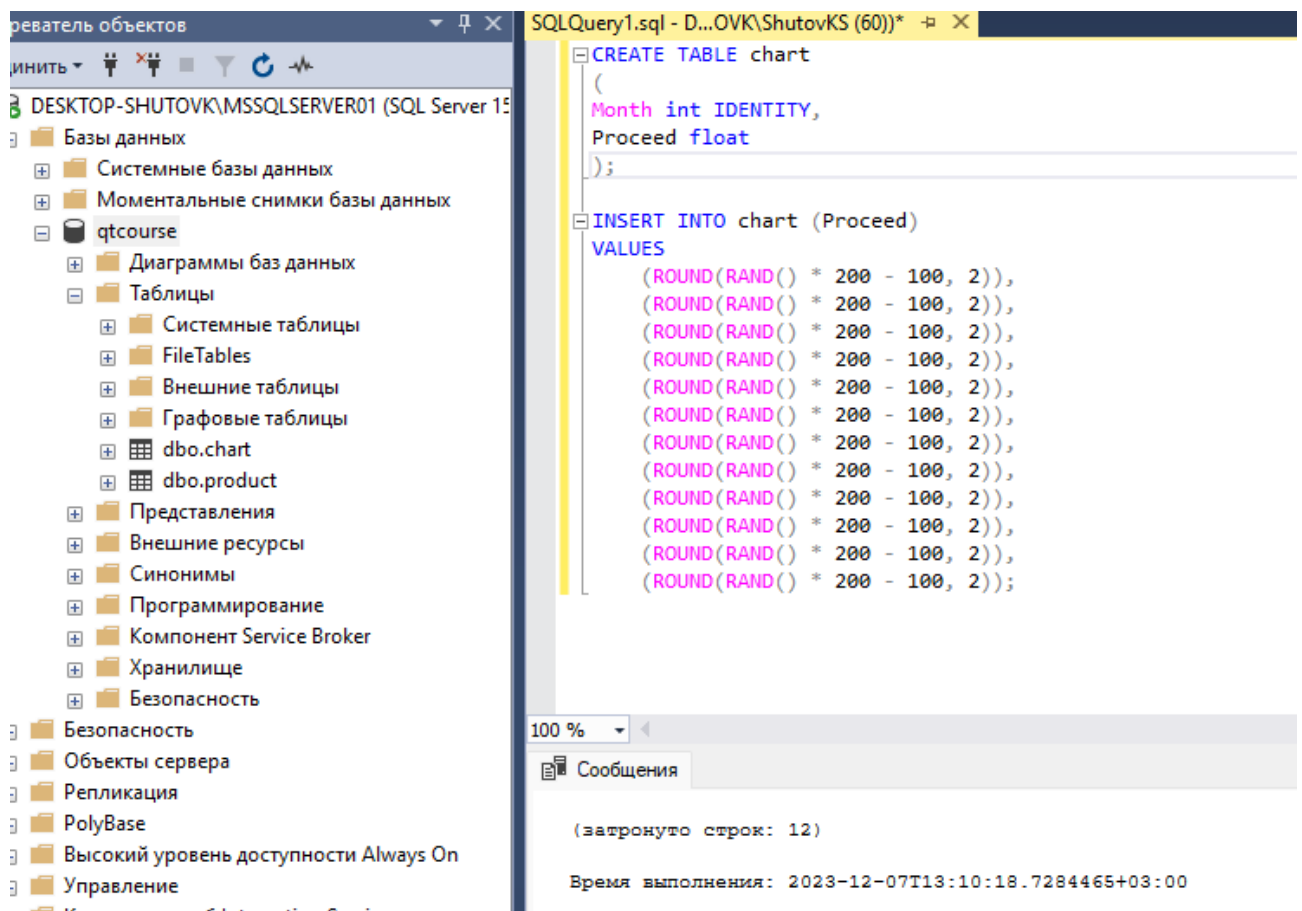


Рисунок 15. Создание новой таблицы

Добавлен экран с графиком (см. Рисунок 16).

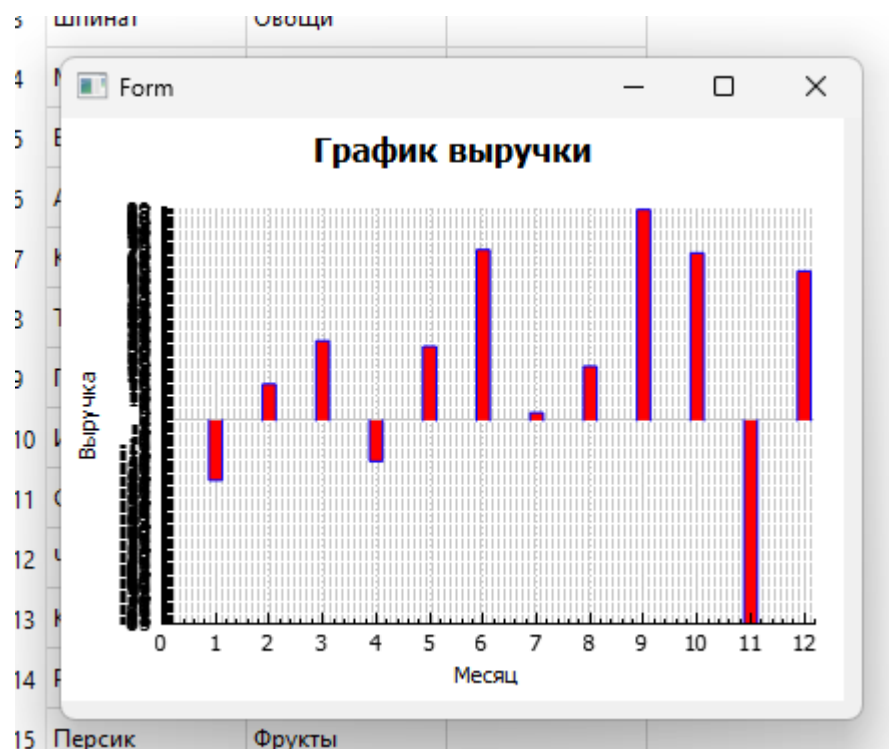


Рисунок 16. Экран графика

В таблицу продуктов добавлен новый столбец (см. Рисунок 17).

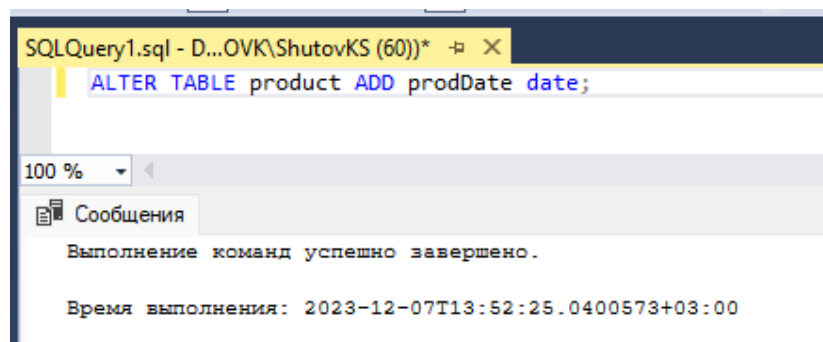


Рисунок 17. Добавление нового столбца

Добавлена дата поступления (см. Рисунок 18).

Наименование

Категория

Дата поступления

01.01.2000

Добавить

Рисунок 18. Дата поступления

Контрольные вопросы

1. Для разделения таблицы product на две таблицы, вероятно, использовались операторы для работы с таблицами в языке SQL, такие как CREATE TABLE, INSERT INTO, SELECT, и, возможно, ALTER TABLE. Например, можно было бы создать новую таблицу с определенными столбцами, скопировать необходимые данные из таблицы product в новую таблицу, а затем изменить структуру таблицы product с использованием оператора ALTER TABLE, чтобы соответствовать требованиям проекта.
2. Для хранения информации об изображении обычно используется тип данных BLOB (Binary Large Object) или его варианты, такие как BINARY или VARBINARY. Это позволяет хранить бинарные данные, включая изображения, в базе данных.
3. Да, изображение можно хранить непосредственно в базе данных, используя соответствующий тип данных (например, BLOB). Однако, это может вызвать увеличение размера базы данных, что может повлиять на производительность. В некоторых случаях более эффективным решением может быть хранение ссылок на файлы изображений, а сами изображения сохранять в файловой системе.
4. Для обеспечения доступности изображения всем пользователям информационной системы, обычно рекомендуется хранить изображения во внешнем хранилище, а не в базе данных. Это может быть файловая система сервера, облачное хранилище (например, Amazon S3, Google Cloud Storage) или специализированная система управления медиа-контентом. В базе данных можно хранить пути к изображениям или другую метаданную, которая позволит системе идентифицировать и загружать соответствующие изображения при необходимости.

