УДК 004

Васильев В.И.
студент 2 курса
факультет «Информационные системы и технологии»
Северный Арктический федеральный университет
Высшая школа информационных технологий и автоматизированных систем
Россия, г. Архангельск

РАЗРАБОТКА КЛИЕНТ-СЕРВЕРНОГО ПРИЛОЖЕНИЯ ДЛЯ УЧЁТА ПРОДУКЦИИ МАГАЗИНА

Аннотация: Статья посвящена моделированию информационной системы предприятия. А также работе с языком UML. Кроме того, в данной работе рассказывается создание базы данных. А также в статья рассматривается пример создания приложения на языке программирования Java.

Ключевые слова: Java, UML, архитектура информационных систем, MySQL, база данных, информационная система.

Vasilyev V.I.
student, 2 year, faculty "Information Systems and Technology"
Northern Arctic Federal University, Graduate School of Information Technology and Automated Systems
Russia, Arkhangelsk

DEVELOPMENT OF A CUSTOMER-SERVER APPLICATION FOR ACCOUNTING STORE PRODUCTS

Annotation: The article is devoted to modeling the enterprise information system. As well as working with the UML language. In addition, this paper describes the creation of a database. Also, the article discusses an example of creating an application in the Java programming language.

Keywords: Java, UML, information systems architecture, MySQL, database, information system.

Введение

В настоящее время информационные технологии играют важную роль в жизнедеятельности предприятий. Информатизация стала неотъемлемой частью нашей жизни и работы, ускоряя решение задач, принятие решений и, самое главное, передачу информации. Рост информационных технологий благоприятствовал необходимости увеличения штата сотрудников информационно-технических отделов, а также усложнению архитектуры вычислительных сетей.

В прошлом информация, преимущественно располагалась на локальном компьютере, либо компьютере-сервере, что позволяло быстрый доступ к данным. Одними из главных угроз потерь документов являлась неисправность компьютера, но со временем развитие технологий увеличило

объем угроз от различного рода программ-вирусов, которые либо шифруют информацию на компьютере жертвы, для получения выгоды интернет-мошенников, либо попросту уничтожают ее.

Так же увеличенный ритм жизни людей с частыми разъездами все больше ставит задачи резервного архивирования и синхронизации данных между устройствами.

1 Анализ предметной области

Рассмотрим работу магазина покупатель совершает, покупки в магазине, выбирая различные товары.

Товары, поступающие в магазин, подлежат учёту и записываются в список прибывших товаров. Эти товары разделяются на различные категории. Товар имеет определённую цену, а также обладает свойствами, указанными в его описании. Кассир работает на кассе и при покупке клиентом магазина товара заносит его покупку в книгу покупок для учета купленной продукции.

Так же обычно при оплате товара клиент имеет карту клиента, при помощи которой может получить бонусные баллы или скидку для покупки. Разные клиенты имеют разные карты с уникальным идентификатором и разными фиксированными размерами скидок.

Темой курсового проекта является разработка автоматизированной системы лояльности магазина, а также автоматизированной системы учёта продукции.

Занимаясь анализом предметной области, необходимо отразить ее основные составляющие, чтобы получить максимально эффективный результат.

Основными составляющими предметной области являются:

– исполнители;
– действия;
– объекты.

Исполнители – это лица, оперирующие с предметной областью.

Необходимо автоматизировать действия, которые они выполняют. Таким образом, все действия предметной области станут функциями информационной системы.

Объекты – это то, на что направлены действия и с чем работают исполнители предметной области. Впоследствии объекты станут таблицами баз данных.

Были определены следующие группы исполнителей:

– оператор;
– кассир;
– незарегистрированный клиент;
– зарегистрированный клиент.

Действия, происходящие в предметной области, были определены следующие:

– регистрация клиента;

– авторизация клиента;
– работа с кассовым аппаратом;
– управление базой данных.

Объекты, с которыми взаимодействуют исполнители предметной области:

– товар;
– карта лояльности.

В качестве выводов на основании проделанного анализа предметной области был сформирован перечень первичных требований к проекту:

– наличие базы данных карт лояльности клиентов, продавцов, заказов и продуктов компании;
– возможность заполнения личных данных карт лояльности клиентов;
– возможность редактирования личных данных карт лояльности клиентов;
– возможность удаления личных данных карт лояльности клиентов
– возможность заполнения данных о продавцах;
– возможность редактирования данных о продавцах;
– возможность удаления данных о продавцах;
– возможность заполнения данных о заказах;
– возможность редактирования данных о заказах;
– возможность удаления данных о заказах;
– возможность заполнения данных о продуктах;
– возможность редактирования данных о продуктах;
– возможность удаления данных о продуктах.

2 Проектирование архитектуры

2.1 Определение архитектуры

Архитектура приложения — совокупность важнейших решений об организации программной системы. В основе работы приложения лежит так называемая модель взаимодействия клиент-сервер, которая позволяет разделять функционал и вычислительную нагрузку между клиентскими приложениями и серверными приложениями. Ниже на рисунке 1, представлена схема клиент-серверной архитектуры

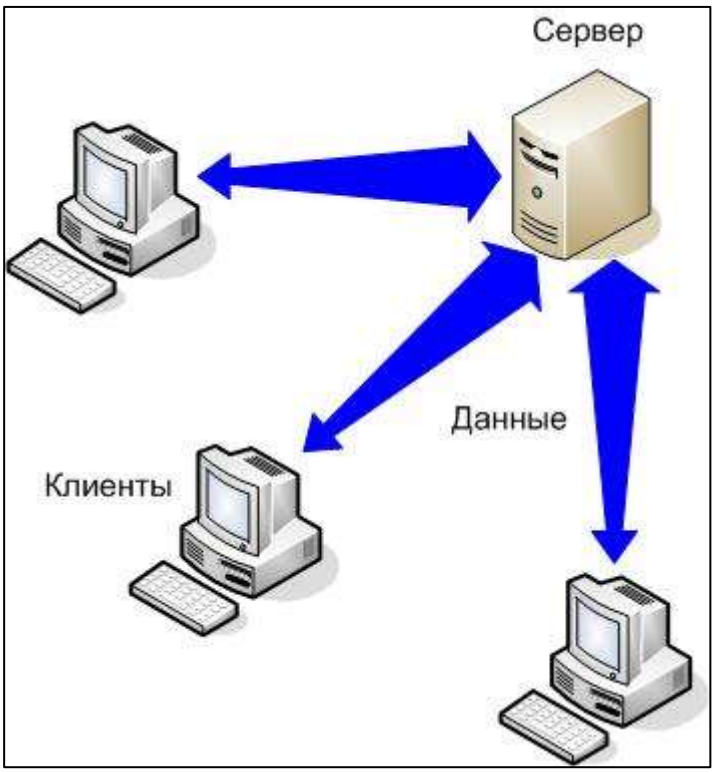

Рисунок 1 Модель архитектуры приложения

Клиент и сервер взаимодействую друг с другом в сети Интернет или в любой другой компьютерной сети при помощи различных сетевых протоколов, например, IP протокол, HTTP протокол, FTP и другие. Протоколов на самом деле очень много и каждый протокол позволяет оказывать ту или иную услугу. Также стоит заметить, что в основе взаимодействия клиент-сервер лежит принцип того, что такое взаимодействие начинает клиент, сервер лишь отвечает клиенту и сообщает о том, может ли он предоставить услугу клиенту и если может, то на каких условиях.

2.2 Проектирование диаграммы вариантов использования

Вариант использования представляет собой последовательность действий (транзакций), выполняемых системой в ответ на событие, инициируемое некоторым внешним объектом (действующим лицом).

Вариант использования описывает типичное взаимодействие между пользователем и системой. В простейшем случае вариант использования определяется в процессе обсуждения с пользователем тех функций, которые он хотел бы реализовать.

Для системы лояльности магазина была разработана диаграмма вариантов использования, представленная на рисунке 2.

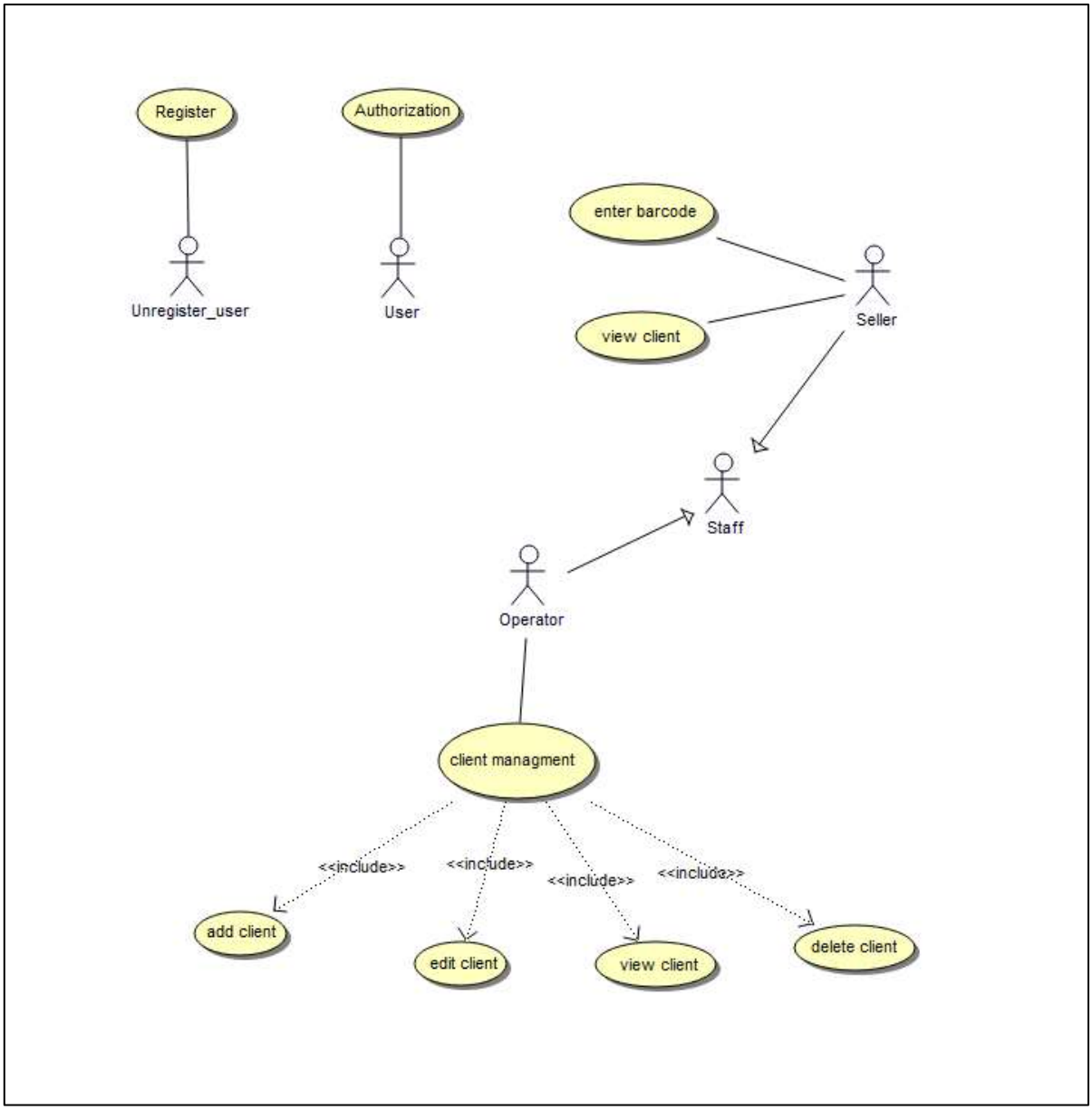

Рисунок 2 – Модель вариантов использования

2.3 Разработка потока событий

Целью потока событий является документирование процесса обработки данных, реализуемого в рамках варианта использования. Этот документ подробно описывает, что будут делать пользователи системы, и что - сама система.

Система имеет несколько вариантов использования:

– регистрация пользователя;
– авторизация пользователя;
– ввод штрих кода;
– просмотр клиента;
– изменение клиента.

Основной поток включает в себя:

– вариант использования начинается, когда пользователь регистрируется в системе;

– пользователь при завершении регистрации добавляет себя в базу данных;

– затем при совершении покупок пользователь показывает свой идентификатор (штрих код) продавцу;

– продавец заносит транзакцию в базу данных и начисляет пользователю бонусы.

Альтернативный поток А1 включает в себя:

– пользователь не имеет мобильного приложения;

– оператор вносит данные пользователя в базу данных;

– оператор выдает пользователю физическую карту;

– затем при совершении покупок пользователь показывает свой идентификатор (штрих код) продавцу;

– продавец заносит транзакцию в базу данных и начисляет пользователю бонусы.

Альтернативный поток А2 включает в себя:

– пользователь регистрировал аккаунт;

– пользователь запрашивает пароль на адрес электронной почты;

– пользователь восстанавливает учётную запись.

Альтернативный поток А3 включает в себя:

– пользователь неправильно ввёл данные;

– оператор редактирует данные пользователя.

2.4 Выводы

На этапе проектирования следующие задачи требовали реализации:

– определение архитектуры проекта;

– проектирование диаграммы вариантов использования;

– разработку потока событий.

Все поставленные задачи по проектированию были выполнены в полной мере, что позволяет перейти к этапу разработки информационной системы

3 Установка и администрирование. Развертывание удалённого сервера и настройка прав пользователей

Для развёртывания удалённого сервера был приобретён хост на сайте REG.RU после приобретения хостинга на посту пришли логины и пароли от доступа в панель управления хостингом, доступа к ftp и доступа к Mysql.

Перед началом работы с базой данных был выполнен вход в панель управления хостингом и разрешён удалённый доступ как показано на рисунке 3.

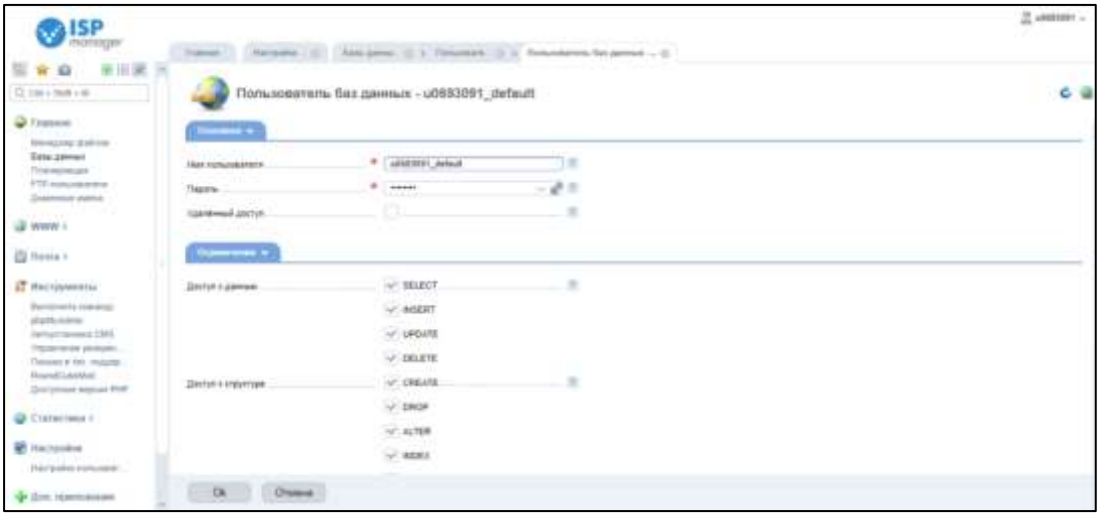

Рисунок 3– Модель вариантов использования

Для проверки работы базы данных была использована программа MySQLWorkbench. В программе было создано новое подключение, где был указан ip-адресс и порт сервера, а также название пользователя и пароль. Процесс настройки подключения показан на рисунке 4.

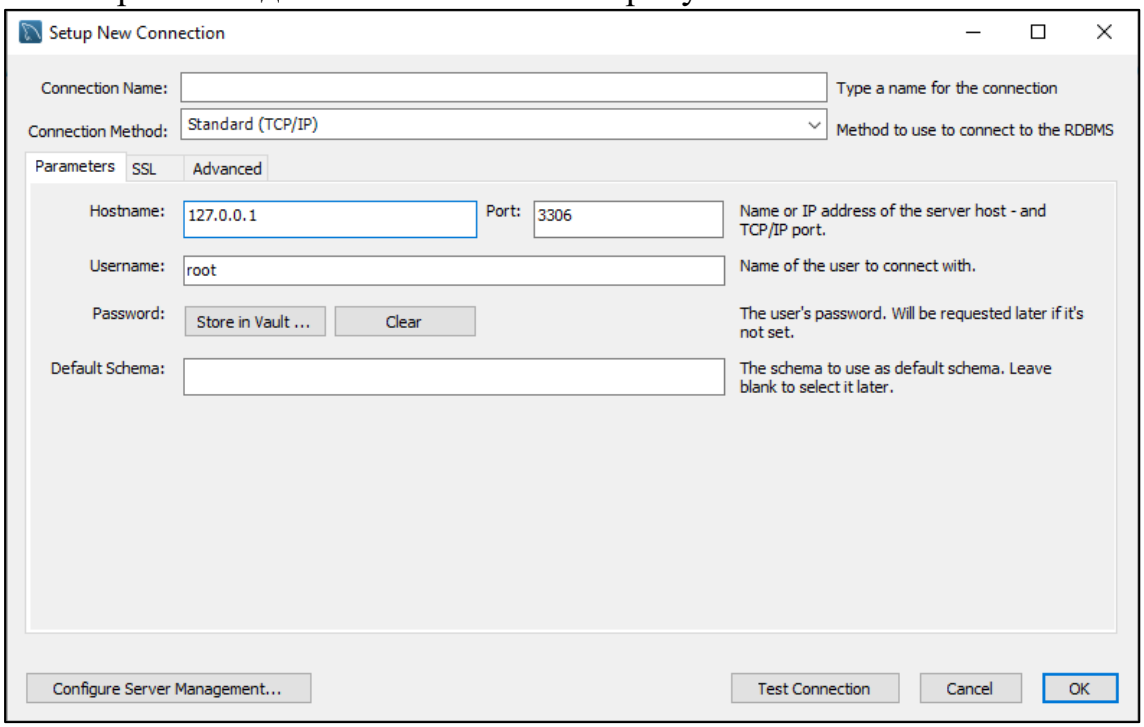

Рисунок 4 – Окно нового подключения в MySQLWorkBench

После успешного подключения для проверки корректности работы была создана таблица, как показано на рисунке 5.

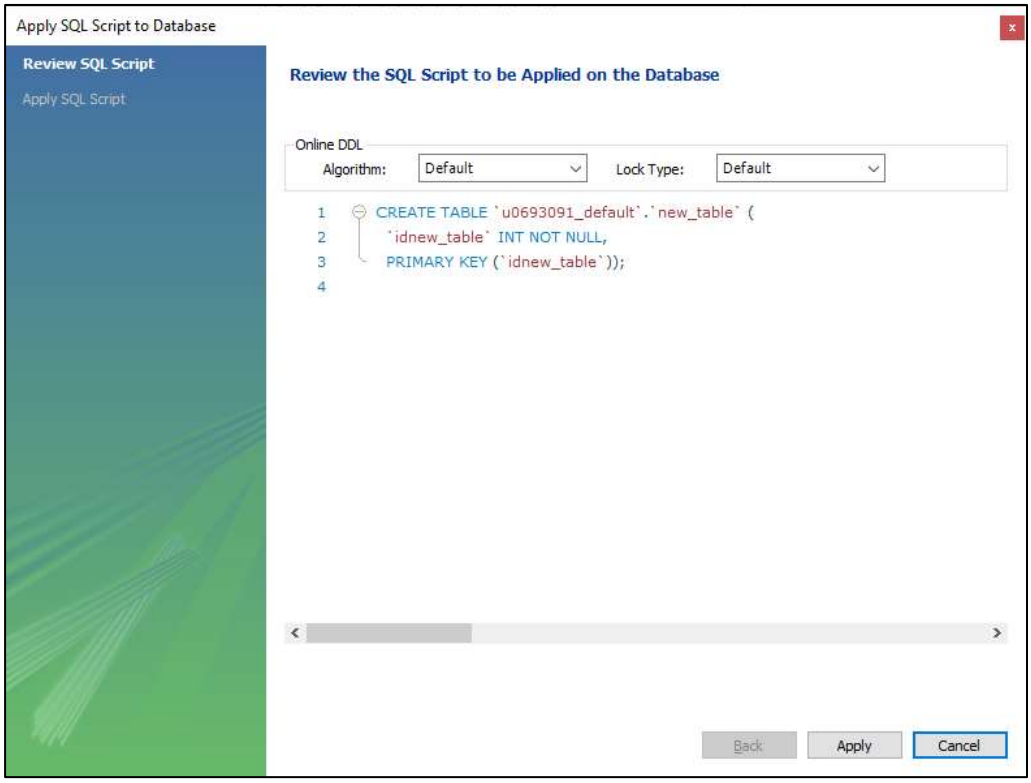

Рисунок 5– Окно запросов в MySQLWorkBench

4 Создание базы данных

4.1 Создание ER-диаграммы

Для создания базы данных была использована программа MySQLWorkbench, так как в этой программе имеется возможность создавать ER-диаграммы, с возможностью импорта в SQL.

Созданная SQL диаграмма показана на рисунке 6.

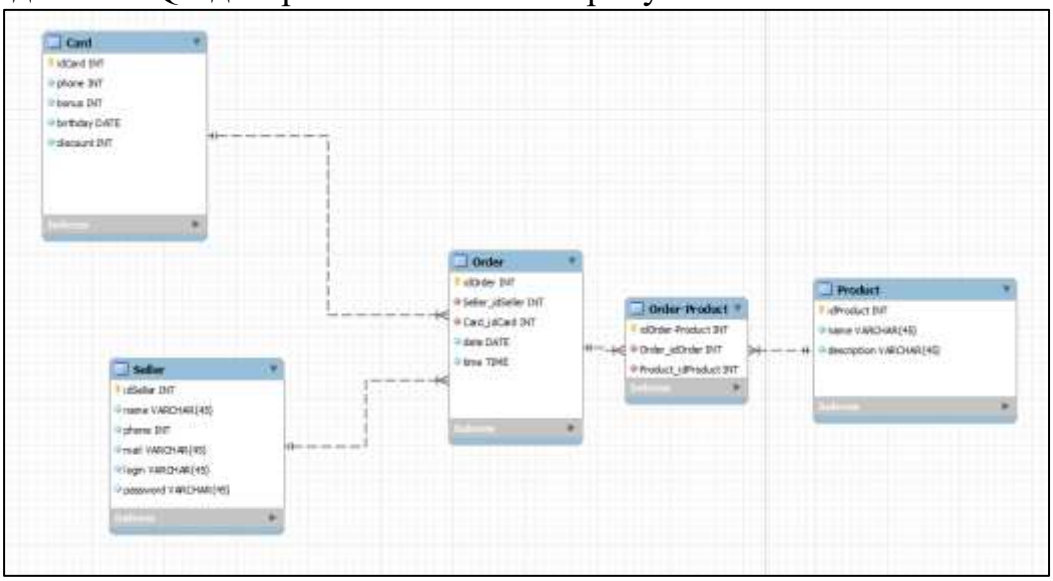

Рисунок 6 – ER-диаграмма

После этого ER диаграмма была преобразована в SQL файл.

4.2 Импорт базы данных на сервер

Для импорта базы данных был выполнен вход в PhpMyAdmin и после этого были импортированы готовые таблицы на основе ER-диаграммы из

MySQLWorkBench, для этого в PhpMyAdmin был выбран пункт Импорт, как показано на рисунке 7.

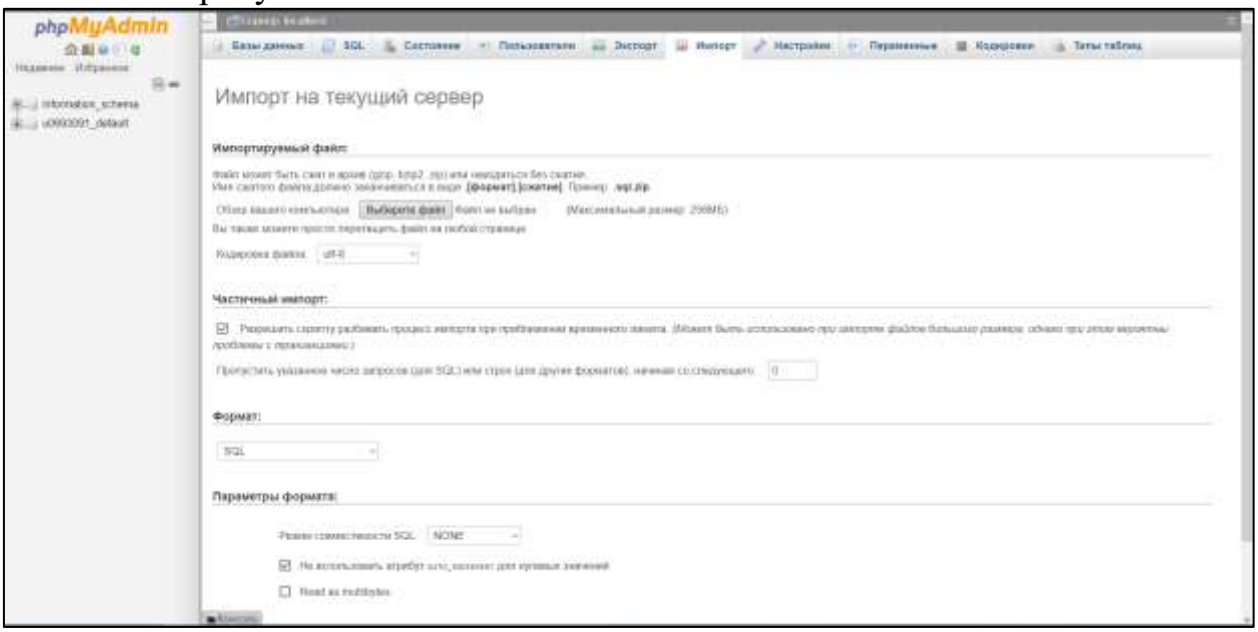

Рисунок 7 – Окно импорта в phpMyAdmin

После этого был выбран предварительно созданный файл в MySQLWorkBench, SQL файл. После импортирования файла все таблицы были созданы в базе данных автоматически, как показано на рисунке 8.

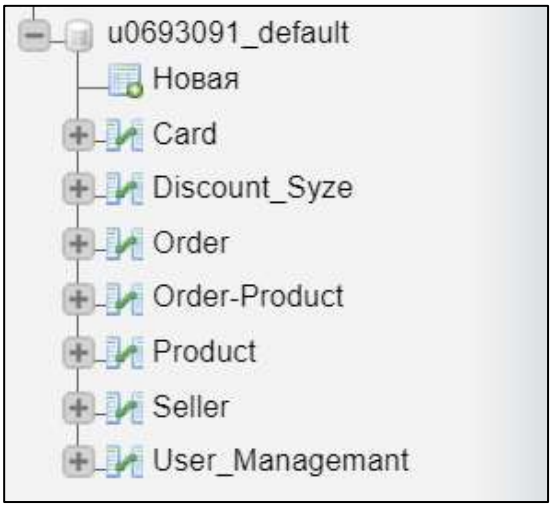

Рисунок 8 – Импортируемые таблицы

4.3 Выводы

В качестве выводов на основании созданной базы данных были реализована ER-модель базы данных, а также успешно был выполнен импорт созданной ER-модели на удалённый сервер.

5 Проектирование приложения

5.1 Дизайн приложения

Проектирование дизайна приложения происходило в программе SceneBuilder, так как эта программа позволяет сразу же использовать интерфейс с функциональностью приложения.

JavaFX Scene Builder - это инструмент визуального макета, который позволяет пользователям быстро создавать пользовательские интерфейсы

приложений JavaFX без программирования. Пользователи могут перетаскивать компоненты пользовательского интерфейса в рабочую область, изменять их свойства, применять таблицы стилей, а код FXML для создаваемого ими макета автоматически создается в фоновом режиме. В результате получается файл FXML, который затем можно объединить с проектом Java, связав пользовательский интерфейс с логикой приложения. [1]

JavaFX - платформа на основе Java для создания приложений с насыщенным графическим интерфейсом. Может использоваться как для создания настольных приложений, запускаемых непосредственно из-под операционных систем, так и для интернет-приложений, работающих в браузерах, и для приложений на мобильных устройствах. [2]

На рисунке 9 изображен макет входа в систему. В центре располагаются два текстовых поля для ввода логина и пароля, где пользователь будет вводить данные своей учетной записи. Ниже располагается кнопка входа при нажатии на которую пользователь либо попадает в систему управления базой данной, либо ему будет высвечиваться окно, сообщающее об ошибке входа, как показано на рисунке 10.

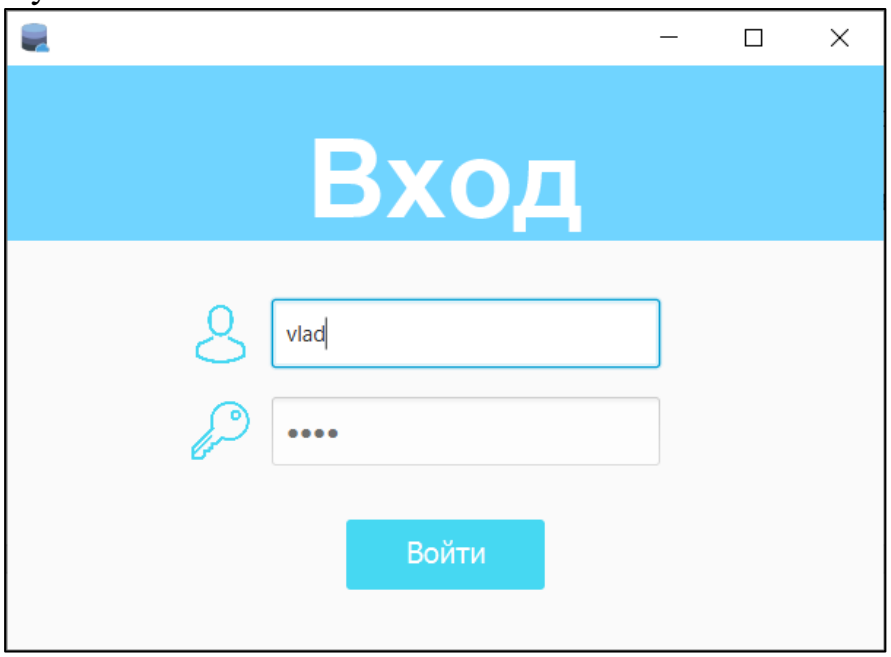

Рисунок 9 – Окно авторизации пользователя

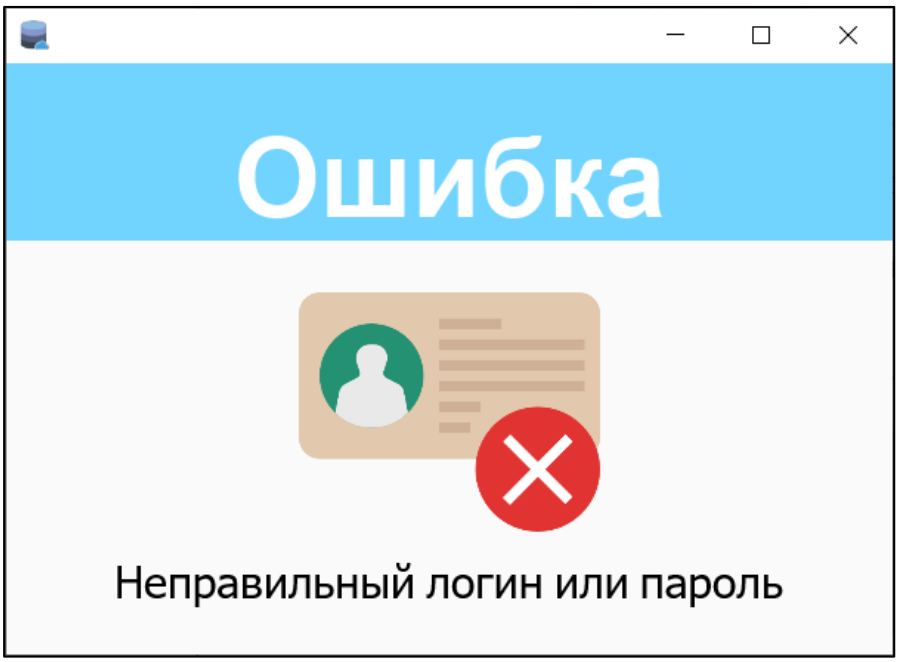

Рисунок 10 – Окно ошибки авторизации

На рисунке 11 изображено окно с выбором редактирования таблиц базы данных, где пользователь может выбрать любую базу данных для изменения, путем нажатия на кнопки с названиями таблиц базы данных.

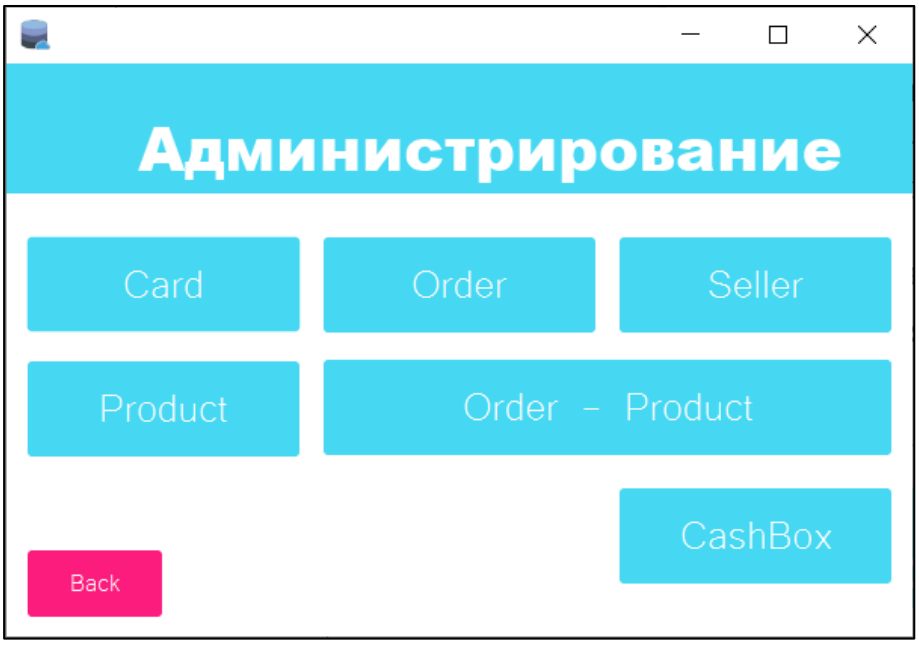

Рисунок 11 – Окно с выбором редактирования таблиц базы данных

Окна с редактированием таблиц баз данных функционально и визуально похожи друг на друга, поэтому для демонстрации дизайна и функциональности, рассмотрим окно редактирования карт клиентов. На рисунке 12 показано окно для изменения параметров карт у клиентов магазина. В левой части окна располагается таблица с содержимым таблицы базы данных «Card», в правой части окна располагаются текстовые поля для изменения записей таблицы базы данных «Card», с кнопками для выбора действий.

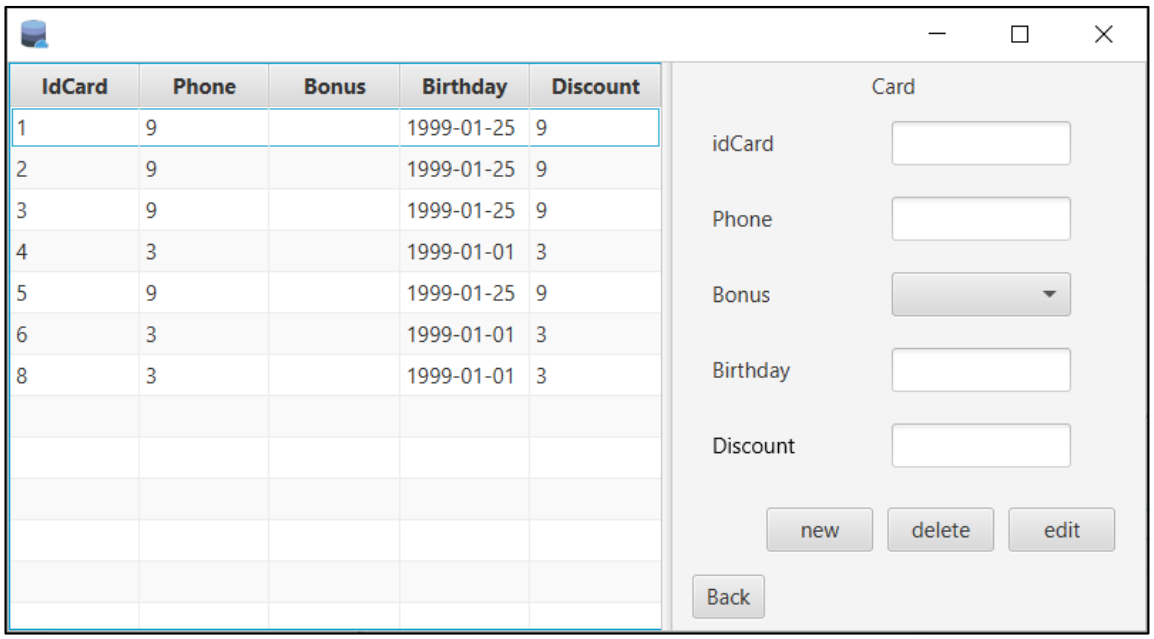

Рисунок 12 - Окна с редактированием таблиц баз данных

5.2 Описание методов

Вся система состоит из множества классов и контроллеров, управляющих активными окнами, а также из главного класса «Main» и класса «Database», отвечающего за взаимодействие с базой данных. Все файлы проекта представлены на рисунке 13.

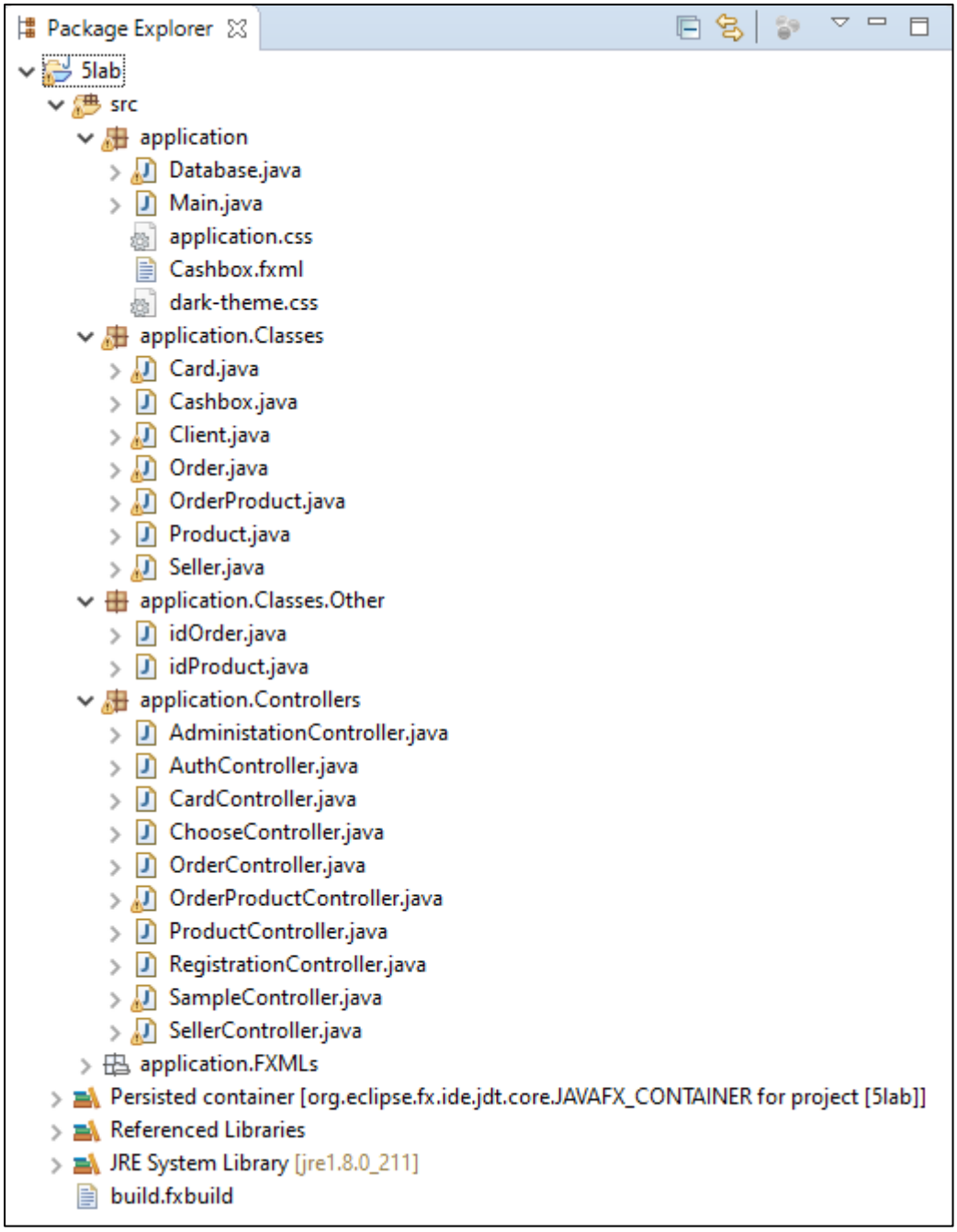

Рисунок 13 – Все классы в программе

Рассмотрим контроллер на примере окна с редактирования карт клиентов – «OrderController». Контроллер содержит метод «initialize», показанный на рисунке 14.

```
    @FXML
    private void initialize() {
        tcIdOrder.setCellValueFactory(new PropertyValueFactory<Order, Integer>("IdOrder"));
        tcSeller_idSeller.setCellValueFactory(new PropertyValueFactory<Order, Integer>("Seller_idSeller"));
        tcCard_idCard.setCellValueFactory(new PropertyValueFactory<Order, Integer>("Card_idCard"));
        tcDate.setCellValueFactory(new PropertyValueFactory<Order, Date>("Date"));
        tcTime.setCellValueFactory(new PropertyValueFactory<Order, Time>("Time"));
        tvOrder.setItems(FXCollections.observableArrayList(db.getAllOrder()));


    showOrderDetails(null);
    tvOrder.getSelectionModel().selectedItemProperty()
            .addListener((observable, oldValue, newValue) -> showOrderDetails(newValue));
}
```

Рисунок 14 – Метод «initialize»

Метод «initialize», необходим для запуска других функций, а также записи начальных значений из базы данных в программу. В контроллере «OrderController» в методе «initialize» реализовано заполнение таблицы данными и базы данных, привязка текстовых полей к переменным и заполнения их начальными значениями, а также методом «showOrderDetails», который берёт данные из таблицы.

Контроллер содержит метод «showOrderDetails», показанный на рисунке 15.

```
private void showOrderDetails(Order c1) {
    if (c1 != null) {
        tfIdOrder.setText(Integer.toString(c1.getIdOrder()));
        tfSeller_idSeller.setText(Integer.toString(c1.getSeller_idSeller()));
        tfCard_idCard.setText(Integer.toString(c1.getIdOrder()));
        tfDate.setText(df.format(c1.getDate()));
        tfTime.setText((c1.getTime().toString()));

    } else {
        tfIdOrder.setText("");
        tfSeller_idSeller.setText("");
        tfCard_idCard.setText("");
        tfDate.setText("");
        tfTime.setText("");
    }
}
```

Рисунок 15– Метод «showOrderDetails»

Методом «showOrderDetails» необходим для отображения данных в текстовых полях при нажатии на строку в таблице, для их дальнейшего редактирования. В метод реализовано заполнение полей значениями из таблицы, которая берёт свои значения из базы данных, если же этого не происходит, то записываются нулевые значения.

Так же контроллер обладает кнопками, для функциональности которых, прописаны отдельные методы. Для кнопок «new», «edit», «delete» прописаны собственные методы, которые показаны на рисунке 16.

```
@FXML
private void handleNew() throws SQLException, NumberFormatException, ParseException  {
    db.newOrder(Integer.parseInt(tfIdOrder.getText()),  Integer.parseInt(tfSeller_idSeller.getText()),
            Integer.parseInt(tfCard_idCard.getText()),new Date(df.parse(tfDate.getText()).getTime()),tfTime.getText());
    tvOrder.setItems(FXCollections.observableArrayList(db.getAllOrder()));
}
@FXML
private void handleEdit() throws SQLException, NumberFormatException, ParseException  {
    db.editOrder(Integer.parseInt(tfIdOrder.getText()),  Integer.parseInt(tfSeller_idSeller.getText()),
            Integer.parseInt(tfCard_idCard.getText()),new Date(df.parse(tfDate.getText()).getTime()),tfTime.getText());
    tvOrder.setItems(FXCollections.observableArrayList(db.getAllOrder()));
}
@FXML
private void handleDelete() throws SQLException, NumberFormatException, ParseException  {
    db.deleteOrder(Integer.parseInt(tfIdOrder.getText()));
    tvOrder.setItems(FXCollections.observableArrayList(db.getAllOrder()));
}
```

Рисунок 16 – Методы «handleNew», «handleEdit», «handleDelete»

Метод «handleNew» отвечает за создание новой записи в базе данных, который берёт изменённые значения из текстовых полей.

Метод «handleEdit» отвечает за изменение записи в базе данных, который берёт изменённые значения из текстовых полей.

Метод «handleDelete» отвечает за удаление записи в базе данных по идентификатору, который берёт изменённые значения из текстовых полей.

Ещё контроллер «OrderController» обладает методом для кнопки «Back», который продемонстрирован на рисунке 17

```
@FXML
void BackButton() {
Stage stage0 = (Stage) bBack.getScene().getWindow();
stage0.close();

FXMLLoader loader = new FXMLLoader();
loader.setLocation(getClass().getResource("/application/FXMLs/Administration.fxml"));

try{
    loader.load();
} catch(Exception e) {
e.printStackTrace();
    }

Parent root = loader.getRoot();
Stage stage = new Stage();
stage.setScene(new Scene(root));
stage.show();
}
```

Рисунок 17– Метод «BackButton»

Этот метод находит текущую сцену у кнопки «Back», после чего закрывает её и открывает новую заранее указанную сцену

Рассмотрим методы класса, на примере класса «Order». Класс «Order» имеет метод установки начальных значений, который показан на рисунке 18

```
public Order() {
    this.IdOrder = new SimpleIntegerProperty(0);
    this.Seller_idSeller = new SimpleIntegerProperty(0);
    this.Card_idCard = new SimpleIntegerProperty(0);
    this.Date1 = new SimpleObjectProperty<Date>(Date.valueOf("2018-01-01"));
    this.Time1 = new SimpleObjectProperty<Time>(Time.valueOf("22:00:00"));;
}
```

Рисунок 18 – Метод «Order»

В этом методе указываются значения, которые не были указаны в сеттерах.

Так же класс «Order» имеет сеттеры и геттеры, которые показаны на рисунке 19.

```
public Order(Integer IdOrder, Integer Seller_idSeller, Integer Card_idCard, Date Date1, Time Time1) {
    this.IdOrder = new SimpleIntegerProperty(IdOrder);
    this.Seller_idSeller = new SimpleIntegerProperty(Seller_idSeller);
    this.Card_idCard = new SimpleIntegerProperty(Card_idCard);
    this.Date1 = new SimpleObjectProperty<Date>(Date1);
    this.Time1 = new SimpleObjectProperty<Time>(Time1);
}

public Integer getIdOrder() {
    return this.IdOrder.get();
}

public Integer getSeller_idSeller() {
    return this.Seller_idSeller.get();
}

public Integer getCard_idCard() {
    return this.Card_idCard.get();
}

public Date getDate() {
    return this.Date1.get();
}

public Time getTime() {
    return this.Time1.get();
}
```

Рисунок 19 – Геттеры и сеттеры класса «Order»

Так же программа имеет класс Databae, который служит для связи с базой данных. Для подключения к базе данных был использован JDBC driver, который был предварительно скачан и подключён как показано на рисунке

В классе «Database» есть метод отвечающий за подключение к базе данных при помощи JDBC driver, как показано на рисунке 20.

```
private boolean openConnection() {
    try {

        Class.forName("com.mysql.jdbc.Driver").newInstance();
        this.conn = DriverManager.getConnection("jdbc:mysql://37.140.192.114:3306?user=u0693091_default&password=xH3cI91_");
    } catch (InstantiationException | IllegalAccessException | ClassNotFoundException | SQLException e) {
        // TODO Auto-generated catch block
        e.printStackTrace();
        return false;
    }
    return true;
}
```

Рисунок 20 – Метод «openConnection»

Этот метод создает подключение к указанному ip-адресу используя заранее прописанный логин и пароль, если же подключение не удается то он выводит сообщение об ошибке в консоль.

В классе «Database» есть метод отвечающий за закрытие соединения как показано на рисунке 21.

```
public void closeConnection() {
    try {
        if (this.conn != null)
            this.conn.close();
    } catch (SQLException e) {
        e.printStackTrace();
    }
    this.conn = null;
}
```

Рисунок 21 – Метод «closeConnection»

В классе «Database» есть метод отвечающий за авторизацию продавца в системе, как показано на рисунке 22.

Этот метод принимает логин и пароль из окна авторизации, а за тем при активном подключении делает выборку в таблице «Seller» и при нахождении совпадающих результатов он прибавляет к счётчику единицу и отправляет этот результат вызванному методу из класса «AuthController»

В классе «Database» есть типовые методы отвечающий за получение данных из таблиц В качестве примера рассмотрим метод связанный с таблицей базы данных «Order».

```
//вход в систему обращаемся к базе данных проверяем записи
public int getSellers(String login, String password) {
    Statement st = null;
    ResultSet rs = null;
    int counter = 0;
    List<Seller> lSellers = new ArrayList<Seller>();
    if (openConnection()) {
        try {
            st = conn.createStatement();
            rs = st.executeQuery("select * from u0693091_default.Seller WHERE login ='" + login +
                    "' AND password = '" + password+"';");
            while (rs.next()) {
                counter++;
            }
        } catch (SQLException e) {
            System.out.println("SQl exception" + e.getMessage());
            e.printStackTrace();
            return 0;
        } finally {
            try {
                if (st != null)
                    st.close();
                closeConnection();
            } catch (SQLException e) {
                e.printStackTrace();
            }
            st = null;
        }
    }

    return counter;
}
```

Рисунок 22– Метод «getSellers»

Метод «getAllOrder» получает данные из таблицы базы данных «Order» и предаёт их в «OrderController». Метод «getAllOrder» представлен на рисунке 23.

```
public List<Order> getAllOrder() {
    Statement st = null;
    ResultSet rs = null;
    List<Order> lOrder = new ArrayList<Order>();
    if (openConnection()) {
        try {
            st = conn.createStatement();
            rs = st.executeQuery("select * from u0693091_default.Order");
            while (rs.next()) {
                lOrder.add(new Order(rs.getInt("idOrder"), rs.getInt("Seller_idSeller"), rs.getInt("idCard"),rs.getDate("date"),rs.getTime("time")));
            }
        } catch (SQLException e) {
            System.out.println("SQl exception" + e.getMessage());
            e.printStackTrace();
            return null;
        } finally {
            try {
                if (st != null)
                    st.close();
                closeConnection();
            } catch (SQLException e) {
                e.printStackTrace();
            }
            st = null;
        }
    }
    return lOrder;
}
```

Рисунок 23–– Метод «getAllOrder»

Так же в классе «Database» есть типовые методы, отвечающие за добавление, редактирование и удаление данных. В качестве примера рассмотрим методы связанные с таблицей базы данных «Order».

Метод «newOrder» отвечает за создание новой записи в таблице базы данных «Order». Метод «newOrder» представлен на рисунке 24.

```
public void newOrder(int idOrder, int Seller_idSeller, int Card_idCard, Date Date, String Time) {
    if (openConnection()) {
        Statement st = null;
        try {
            st = conn.createStatement();
            st.executeUpdate("insert into u0693091_default.Order (idOrder, Seller_idSeller, idCard, Date, Time) values('" +
            idOrder + "','" + Seller_idSeller + "','" + Card_idCard + "','" + Date + "','" + Time +"');");
        } catch (SQLException e) {
            System.out.println("SQl exception" + e.getMessage());
            e.printStackTrace();
        } finally {
            try {
                if (st != null)
                    st.close();
                closeConnection();
            } catch (SQLException e) {
                e.printStackTrace();
            }
            st = null;
        }
    }
}
```

Рисунок 24-– Метод «newOrder»

В этом методе реализован запрос на вставку полученных данных в таблицу базы данных «Order».

Метод «editOrder» отвечает за редактирование записи в таблице базы данных «Order». Метод «editOrder» представлен на рисунке 25.

```
public void editOrder(int idOrder, int Seller_idSeller, int Card_idCard, Date Date, String Time) {
    if (openConnection()) {
        Statement st = null;
        try {
            st = conn.createStatement();
            st.executeUpdate("update u0693091_default.Card set Seller_idSeller = " +
            Seller_idSeller + "', idCard = '" + Card_idCard + "', Date = '" + Date + "', Time = '" + Time +"');");
        } catch (SQLException e) {
            System.out.println("SQl exception" + e.getMessage());
            e.printStackTrace();
        } finally {
            try {
                if (st != null)
                    st.close();
                closeConnection();
            } catch (SQLException e) {
                e.printStackTrace();
            }
            st = null;
        }
    }
}
```

Рисунок 25-– Метод «editOrder»

В этом методе реализован запрос на обновление данных в таблице базы данных «Order».

Метод «deleteOrder» отвечает за удаление записи в таблице базы данных «Order». Метод «deleteOrder» представлен на рисунке 26.

```
public void deleteOrder(int idOrder) throws SQLException {
    if (openConnection()) {
        Statement st = null;
        try {
            st = (Statement) conn.createStatement();
            st.executeUpdate("delete from u0693091_default.Order WHERE idOrder =" + idOrder);
        } catch (SQLException e) {
            System.out.println("SQl exception" + e.getMessage());
            e.printStackTrace();
        } finally {
            try {
                if (st != null)
                    st.close();
                closeConnection();
            } catch (SQLException e) {
                e.printStackTrace();
            }
            st = null;
        }
    }
}
```

Рисунок 26-– Метод «deleteOrder»

В этом методе реализован запрос на удаление данных в таблице базы данных «Order».

5.3 Выводы

В ходе проектирования приложения был разработан удобный и интуитивно понятный интерфейс приложения в программе SceneBuilder. А также добавлен функционал приложение путём написания программного кода на языке программирования Java в среде разработке Eclipse.

6 Тестирование приложения

Проверка на неправильный вход, представлена на рисунке 27. Вводим неправильный логин и пароль. А результате появляется окно с ошибкой. Проверка выполнена.

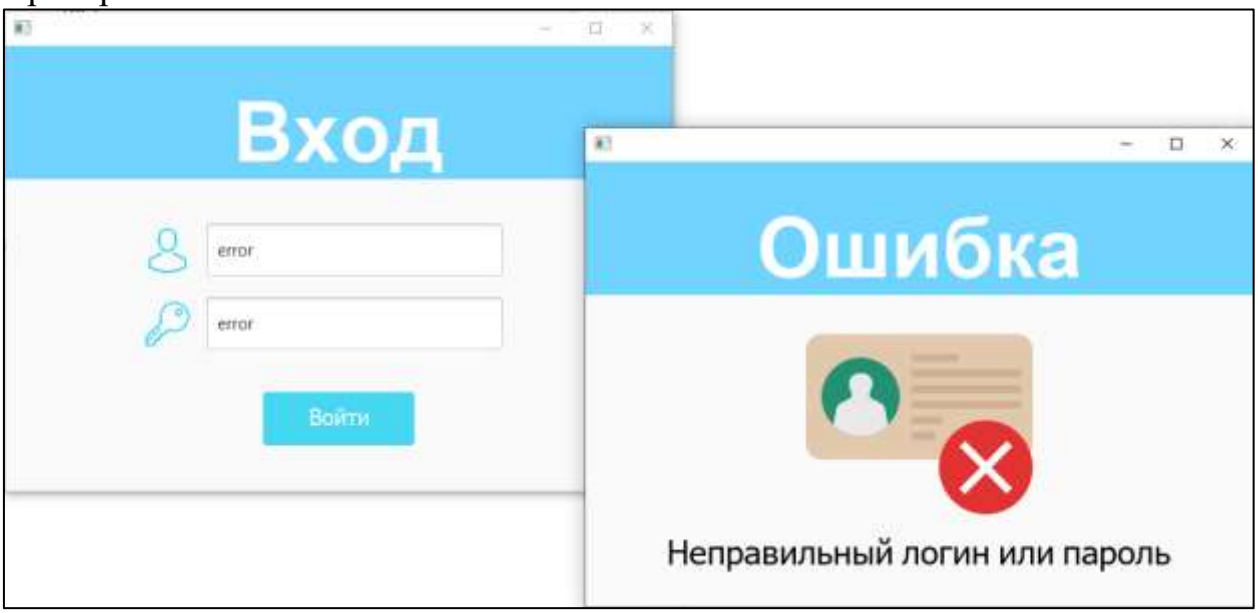

Рисунок 27 – Проверка на неправильный вход

Проверка на правильный вход, представлена на рисунке 28. Вводим неправильный логин и пароль. А результате появляется окно с выбором таблиц. Проверка выполнена.

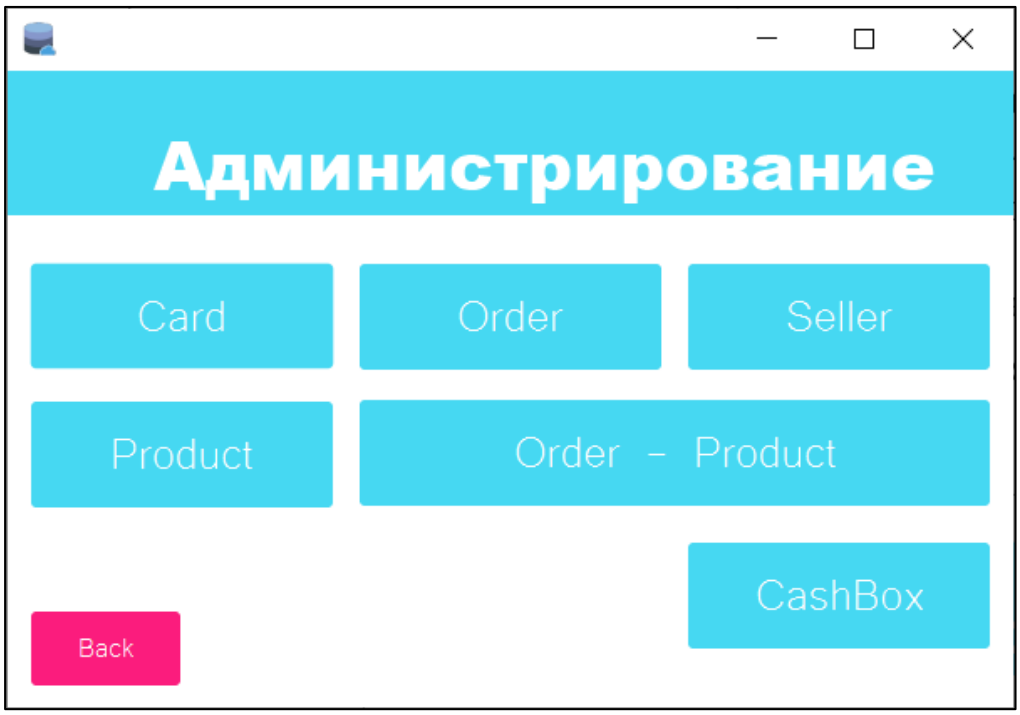

Рисунок 28– Проверка на правильный вход

Проверка на создание записей с одинаковым первичным ключом, представлена на рисунке 29. Вводим в поле с первичным ключом идентификатор, существующий в таблице базы данных, запись не создаётся. Проверка выполнена.

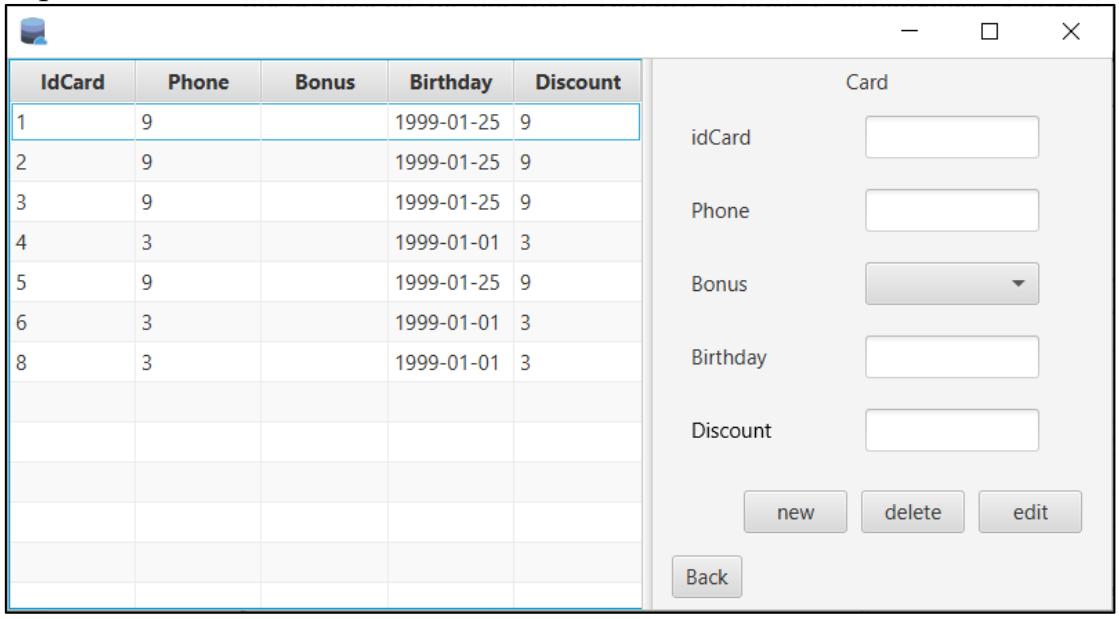

Рисунок 29 – Проверка на создание записей с одинаковым первичным ключом

Проверка на удаление взаимосвязанных элементов, представлена на рисунке 30. Пытаемся удалить элемент, который взаимосвязан с другими элементами, удаление не удается. Проверка выполнена.

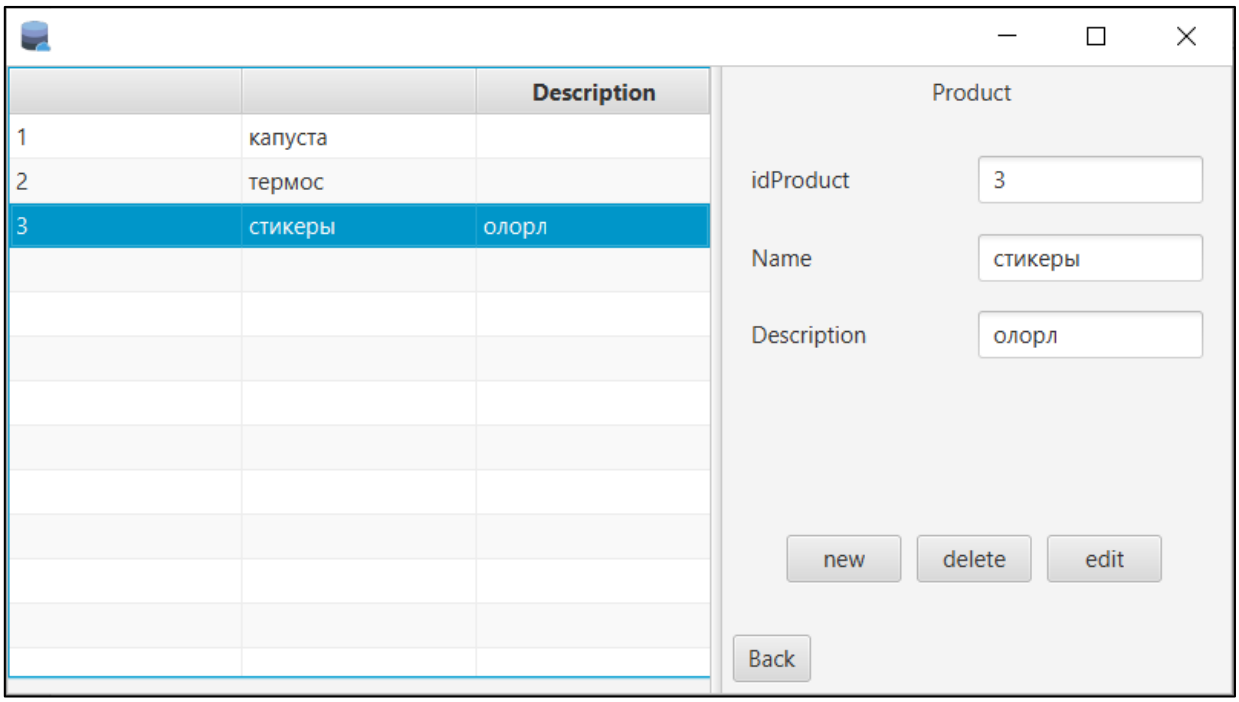

Рисунок 30 – Проверка на удаление взаимосвязанных элементов

В ходе проведения тестирования были проверены основные средства защиты от некорректного ввода данных пользователем системы.

Заключение

Целью данного проекта была разработка системы лояльности магазина, позволяющая автоматизировать процесс учёта товаров, а также предоставить клиентам магазина доступ к просмотру информации о картах лояльности. Разработанная система обладает функциями добавления, удаления и редактирования записей всех таблиц базы данных, так же система имеет в своей функциональности окна регистрации и авторизации пользователей.

Так же система имеет ряд защит от введения некорректных данных пользователем системы.

В ходе разработки были учтены все поставленные требования, а информационная система включила в себя все необходимые функции.

В конце было произведено тестирование всех частей системы, на основании которого можно сказать, что система работает корректно и удовлетворяет всем поставленным задачам.

Информационная система интуитивно понятна и не вызывает сложности в работе.

Все поставленные задачи были выполнены.

Использованные источники:

1. Страница загрузки JavaFX Scene Builder [Электронный ресурс]. Режим доступа:https://www.oracle.com/technetwork/java/javase/downloads/javafxscene builder-info-2157684.html (дата обращения 24.06.2019).;
2. Википедия [Электронный ресурс]. Режим доступа https://ru.wikipedia.org/wiki/JavaFX (дата обращения 24.06.2019).;