МИНИСТЕРСТВО ОБРАЗОВАНИЯ НАУКИ И МОЛОДЕЖНОЙ ПОЛИТИКИ

НИЖЕГОРОДСКОЙ ОБЛАСТИ

Государственное бюджетное профессиональное образовательное учреждение

НИЖЕГОРОДСКИЙ РАДИОТЕХНИЧЕСКИЙ КОЛЛЕДЖ

Специальность 09.02.07 Информационные системы и программирование

Квалификация Программист

КУРСОВАЯ РАБОТА

МДК 11.01 Технология разработки и защиты баз данных

Тема: Разработка приложения и базы данных для предметной области «Университеты»

|  |  |  |
| --- | --- | --- |
| Выполнил обучающийся группы  2ИСиП-22-3с  Власова К.С. |  | Проверил преподаватель  Дворянинова М.С.  Проект защищен с оценкой \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_ Дата защиты\_\_\_\_\_\_\_\_\_\_\_\_ Подпись\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_ |

Нижний Новгород

2023 г.

Оглавление

[**Введение** 3](#_Toc151948001)

[**1. Анализ предметной области** 4](#_Toc151948002)

[**2. Проектирование** 6](#_Toc151948003)

[**2.1 ERD-модель** 6](#_Toc151948004)

[**2.2. Словарь данных** 6](#_Toc151948005)

[**3. Разработка** 10](#_Toc151948006)

[**3.1 Создание и заполнение таблиц** 10](#_Toc151948007)

[**3.2 Запросы, хранимые процедуры и триггеры.** 11](#_Toc151948008)

[**3.4 Разработка приложения.** 16](#_Toc151948009)

[**3.4.1 Разработка графического интерфейса** 16](#_Toc151948010)

[**3.4.2 Разработка логики приложения и подключение БД.** 18](#_Toc151948011)

[**4. Руководство администратора** 34](#_Toc151948012)

[**Заключение** 36](#_Toc151948013)

[**Список литературы** 37](#_Toc151948014)

# **Введение**

Современные технологии играют ключевую роль в совершенствовании бизнес-процессов и обеспечении удовлетворения потребностей клиентов. Один из таких сегментов, где информационные системы привносят значительные улучшения – сфера образования. Университеты постоянно стремятся улучшить и оптимизировать свои приложения для более удобной работы преподавателей и студентов, предоставляя преподавателям возможность вносить изменения, а студентам просматривать важную для них информацию.

Целью моей курсовой работы является создание приложения и базы данных для министерства образования, позволяющее преподавателям и системным администраторам авторизовываться в приложении, вносить изменения в базу данных, менять рейтинг университетов, для студентов будет предоставляться возможность просмотра информации о предметах. Это позволит министерству образования более тщательно следить за работой университетов и менять их рейтинг.

В соответствии с поставленной целью необходимо решить следующие задачи:

* Анализ предметной области;
* Использование методов проектирования баз данных;
* Использование методов разработки;
* Интеграция функциональных компонентов;
* Сделать вывод о проделанной работе, спроектировать базу данных и приложения, к которому она будет подключаться.

Создание такого приложения с использованием базы данных и графического интерфейса для министерства образования и университетов станет хорошим решением для повышения эффективности работы над этой темой. Также с помощью этого, министерство образование сможет проверить большее количество университетов и более эффективно отслеживать их деятельность.

# **1. Анализ предметной области**

Анализ предметной области "Университеты" позволяет рассмотреть различные аспекты, связанные с этой сферой деятельности. Университеты являются высшими учебными заведениями, специализирующимися в проведении образовательных программ на более высоком уровне, чем колледжи или техникумы. Ниже приведены некоторые ключевые аспекты, которые следует учесть:

1. Образовательные программы:

* Университеты предлагают широкий спектр образовательных программ. Они включают программы бакалавриата, магистратуры и докторантуры. Университеты также могут предлагать разнообразные курсы и специализации, чтобы удовлетворить потребности студентов из разных областей знаний.

1. Научная работа и исследования:

* Университеты активно занимаются научной деятельностью и исследованиями. Они поддерживают ученых и исследователей, проводят фундаментальные и прикладные исследования, организуют научные конференции и публикуют научные статьи. Научная работа является важной составляющей деятельности любого университета.

1. Учебный персонал:

* Университеты нанимают квалифицированный учебный персонал, включающий преподавателей, профессоров и научных сотрудников. Они играют важную роль в образовательном процессе и исследовательской работе. Качество и опытность преподавателей является одним из ключевых факторов привлечения студентов в университеты.

1. Студенты и обучение:

* Университеты принимают на обучение студентов из различных стран и предоставляют им возможность получить качественное образование. Важная роль также отводится студенческой жизни, включая академические клубы, спортивные команды, общежития и другие мероприятия, которые способствуют развитию личности студентов.

1. Аккредитация и заслуги:

* Университеты должны проходить процесс аккредитации, чтобы подтвердить качество своих программ и учебных стандартов. Высшее образование является важным фактором для профессионального и личного развития, поэтому выбор аккредитированных университетов имеет большое значение.

В целом, предметная область "Университеты" предлагает уникальные возможности для получения образования, научной деятельности и развития студентов. Университеты являются главными актёрами в образовательной системе многих стран и играют важную роль в социальном и экономическом развитии общества.

# **2. Проектирование**

## **2.1 ERD-модель**

Для того чтобы создать базу данных мы создали ERD-модель, на которой мы можем более подробно рассмотреть связи между таблицами и некоторые ключевые моменты (Рис.1).

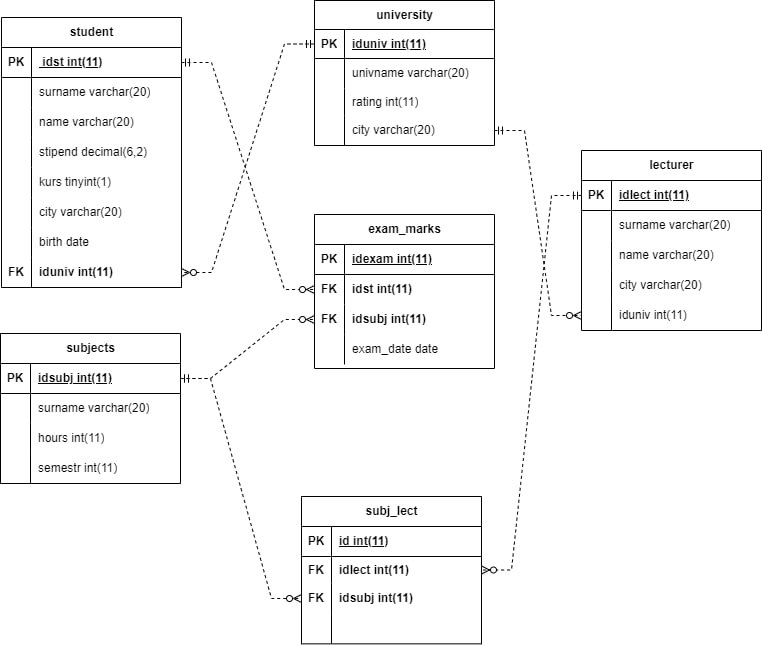


Рис.1 — ERD-модель

Для разработки базы данных мы создаем такую модель, в которой существует 6 сущностей, опишем их в словаре данных.

## **2.2. Словарь данных**

1. Студенты (student):
   * Код студента (idst);
   * Фамилия (surname);
   * Имя (name);
   * Стипендия (stipend);
   * Курс обучения (kurs);
   * Город (city);
   * Дата рождения (birth);
   * Код университета (iduniv FK).
2. Предметы (subjects):
   * Код предмета (idsubj);
   * Название предмета (surname);
   * Часы (hours);
   * Семестр (semestr).
3. Университеты (university):
   * Код университета(iduniv);
   * Название университета (univname);
   * Рейтинг (rating);
   * Город (city).
4. Оценки за экзамены (exam\_marks):
   * Код экзамена (idexam);
   * Дата экзамена (exam\_date);
   * Код студента (idst FK);
   * Код предмета (idsubj FK).
5. Преподаватели (lecturer):
   * Код преподавателя (idlect);
   * Фамилия (surname);
   * Имя (name);
   * Город (city);
   * Код университета (iduniv FK).
6. Таблица, объединяющая преподавателей с учебными предметами (subj\_lect):
   * Код (id);
   * Код преподавателя (idlect FK);
   * Код предмета (idsubj FK).

В нижеследующих таблицах описаны поля всех таблиц базы данных, используемых в разработанной информационной системе.

Таблица student

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Key | Field Name | Data Type | Required? | Примечания |
| PK | idst | INT | Y |  |
|  | surname | VARCHAR(20) | Y |  |
|  | name | VARCHAR(20) | Y |  |
|  | stipend | DECIMAL(6,2) | Y |  |
|  | kurs | TINYINT(1) | Y |  |
|  | city | VARCHAR(20) | Y |  |
|  | birth | DATE | Y |  |
| FK | iduniv | INT | Y | Ссылка на таблицу |

Таблица subjects

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Key | Field Name | Data Type | Required? | Примечания |
| PK | idsubj | INT | Y |  |
|  | surname | VARCHAR(20) | Y |  |
|  | hours | INT | Y |  |
|  | semestr | INT | Y |  |

Таблица university

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Key | Field Name | Data Type | Required? | Примечания |
| PK | iduniv | INT | Y |  |
|  | univname | VARCHAR(20) | Y |  |
|  | rating | INT | Y |  |
|  | city | VARCHAR(20) | Y |  |

Таблица exam\_marks

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Key | Field Name | Data Type | Required? | Примечания |
| PK | idexam | INT | Y |  |
|  | exam\_date | VARCHAR(20) | Y |  |
| FK | idst | INT | Y | Ссылка на таблицу |
| FK | idsubj | INT | Y | Ссылка на таблицу |

Таблица lecturer

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Key | Field Name | Data Type | Required? | Примечания |
| PK | idlect | INT | Y |  |
|  | surname | VARCHAR(20) | Y |  |
|  | name | VARCHAR(20) | Y |  |
|  | city | VARCHAR(20) | Y |  |
| FK | iduniv | INT | Y | Ссылка на таблицу |

Таблица subj\_lect

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Key | Field Name | Data Type | Required? | Примечания |
| PK | id | INT | Y |  |
| FK | idlect | INT | Y | Ссылка на таблицу |
| FK | idsubj | INT | Y | Ссылка на таблицу |

# **3. Разработка**

Существует достаточно много систем управления базами данных такие как PostgreSQL, Firebird, Interbase, IBM DB2, Informix, MS SQL Server и т.п.

База данных будет разрабатывается с помощью системы управления базами данных SQL Server Management Studio Management Studio 2019 – это очень удобный и простой инструмент для разработки баз данных и комфортной работы с ними. Подходит для малых и средних приложений.

**3.1 Создание и заполнение таблиц**

Создание и заполнение таблиц будем производит с помощью запросов. Как пример создания возьмем листинг нескольких таблиц.

|  |
| --- |
| create table student(  idst int constraint cd PRIMARY KEY,  surname varchar(50),  name varchar(50),  stipend int,  kurs tinyint,  city varchar(50),  birth date,  iduniv int); |

Листинг 1 – создание таблицы student.

|  |
| --- |
| create table university(  iduniv int constraint cd PRIMARY KEY,  univname varchar(50),  rating int(50),  city int varchar(50)  ); |

Листинг 2 – создание таблицы university.

|  |
| --- |
| create table subjects(  idsubj int constraint cd PRIMARY KEY,  surname varchar(50),  hours int(50),  semestr int varchar(50)  ); |

Листинг 3 – создание таблицы subjects.

Также для примера заполнения таблиц возьмем как пример некоторые запросы для заполнения.

|  |
| --- |
| insert into student (idst, surname, name, stipend, kurs, city, birth, iduniv)  values (1, 'Potapova', 'Anastasia', 1300, 2, 'Norilsk', '2004-03-25',13),  (2,'Stepanova', 'Nadezhda', 1300, 2, 'Karaganda', '2003-10-18',4),  (3, 'Vlasova', 'Ksenia', 1900, 2, 'Yevpatoria', '2004-04-15',9),  (4, 'Agutin', 'Oleg', 533, 1, 'Orel', '2006-12-01',8),  (5, 'Cupkin', 'Konstantin', 787, 4, 'Karaganda', '2000-07-16',4),  (6, 'Frolov', 'Nikita', 0, 3, 'Moscow', '1994-09-29',10),  (7, 'Hololo', 'Artur', 2335, 1, 'Norilsk', '2006-04-28',1),  (8, 'Lord', 'Kirill', 1200, 2, 'Nizhniy Novgorod', '2006-05-17',6),  (9, 'Yakovlev', 'Yaroslav', 3001, 4, 'Sevastopol', '2001-11-03',11),  (10, 'Tutova', 'Galina', 2700, 3, 'Norilsk', '2003-10-11',1),  (11, 'Kapitoshka', 'Oleg', 6000, 2, 'Tula', '2005-08-09',3),  (12, 'Tamova', 'Tamara', 4500, 3, 'Ufa', '1998-01-14',2),  (13, 'Zdeceva', 'Milana', 0, 1, 'Tula', '1990-02-27',3),  (14, 'Odincova','Galina', 500, 4, 'Astrahan', '2007-07-29',7),  (15, 'Durakov', 'Oleg', 0, 5, 'Rostov', '2003-05-17',12),  (16, 'Horosho', 'Michael', 199, 3, 'Ufa', '2005-06-11',2); |

Листинг 4 – заполнение таблицы student.

|  |
| --- |
| insert into lecturer (idlect, surname, name, city, iduniv)  values (1, 'Potapov', 'Sergey', 'Ufa',2),  (2, 'Genadiev', 'Genadiy', 'Karaganda',4),  (3, 'Galahova', 'Anna', 'Rostov',5),  (4, 'Kozlov', 'Kirill', 'Nizhniy Novgorod',6),  (5, 'Semchenko', 'Valeriya', 'Yevpatoria',9),  (6, 'Baum', 'Mark', 'Tula',3),  (7, 'Solovey', 'Daniil', 'Norilsk',13),  (8, 'Solona', 'Ksenia', 'Sevastopol',11),  (9, 'Smerdnova', 'Sofia', 'Astrahan',7),  (10, 'Borozdina', 'Maria', 'Moscow',10),  (11, 'Vasileva', 'Kristina', 'Norilsk',1),  (12, 'Musienko', 'Maksim', 'Moscow',10),  (13, 'Grechihin', 'Evgeniy', 'Ufa',2),  (14, 'Fedorov', 'Yuriy', 'Tula',3); |

Листинг 5 – заполнение таблицы lecturer.

## **3.2 Запросы, хранимые процедуры и триггеры.**

Теперь создадим несколько запросов на выборку, одну хранимую процедуру и два триггера.

Первый запрос выводит список университетов с рейтингом выше 300, совместно с эти выводя максимальную стипендию в данном университете.

|  |
| --- |
| select university.iduniv, university.univname, university.rating,  university.city, Max (student.stipend) AS stip  FROM university INNER JOIN student ON university.iduniv = student.iduniv  where ((university.rating)>300)  group by university.iduniv, university.univname, university.rating,  university.city; |

Листинг 6 – запрос 1.

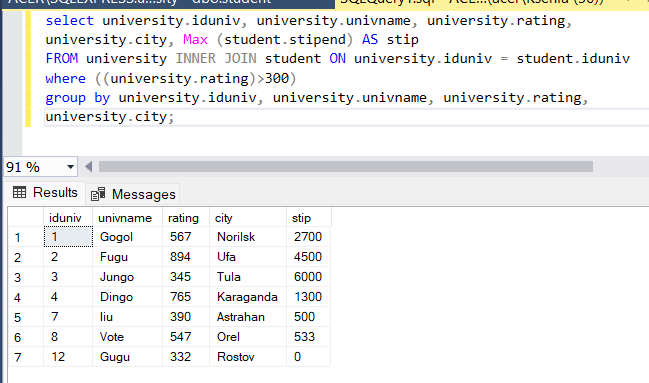


Рис.2 – вывод запроса из листинга 6.

Второй запрос использует SELECT, использующий связанные подзапросы и выполняет вывод имен и идентификаторы студентов с максимальным значением стипендии для города, в котором живет студент.

|  |
| --- |
| select idst, name, stipend  from student  where stipend = (select Max(st.stipend) from student as st  group by st.city having st.city = student.city) |

Листинг 7 – запрос 2.

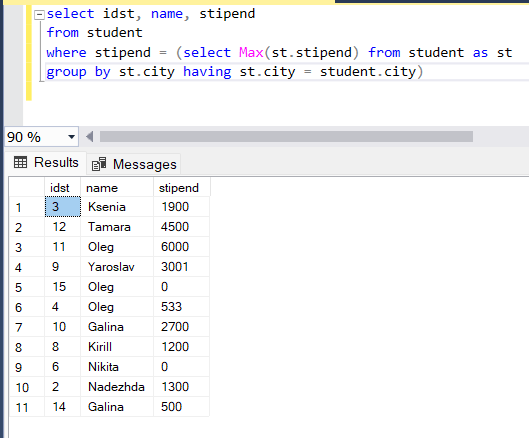


Рис.3 – вывод запроса из листинга 7.

Третий запрос содержит EXIST, выбирающий сведения обо всех студентах, для которых в том же городе, где живет студент, существуют университеты, в которых он не учится.

|  |
| --- |
| select \* from student  where exists (select \* from university  where student.iduniv <> university.iduniv  and student.city = university.city) |

Листинг 8 – запрос 3.

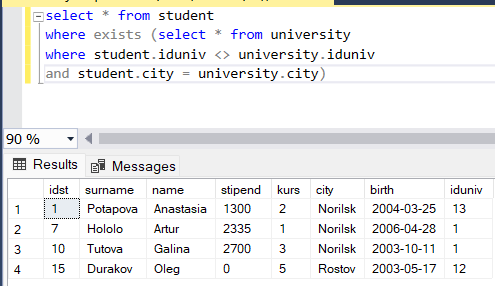


Рис.4 – вывод запроса из листинга 8.

Четвертый запрос, выбирающий из таблицы EXAM\_MARKS данные о названиях предметов обучения, для которых значение полученных на экзамене оценок (поле MARK) превышает любое значение оценки для предмета, имеющего идентификатор, равный 10

|  |
| --- |
| select distinct surname from exam\_marks  inner join subjects on exam\_marks.idsubj = subjects.idsubj  where mark > any (select mark from exam\_marks where idsubj = 10) |

Листинг 9 – запрос 4.

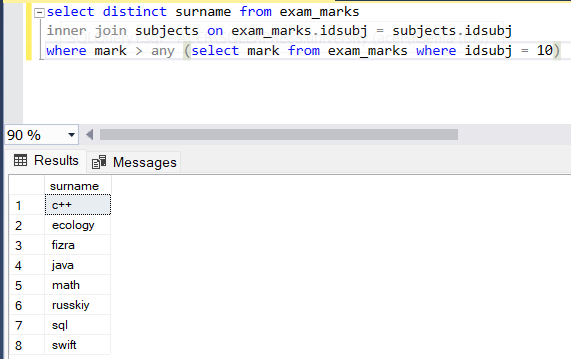


Рис.5 – вывод запроса из листинга 9.

Пятый запрос, который позволяет вывести имена и идентификаторы всех студентов, для которых точно известно, что они проживают в городе, где нет ни одного университета.

|  |
| --- |
| select idst, name  from student  where city not in (select distinct university.city from university) |

Листинг 10 – запрос 5.

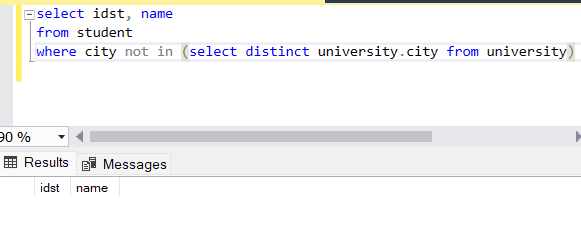


Рис.6 – вывод запроса из листинга 10.

Хранимая процедура выводит фамилию, имя, курс и оценку студента с помощью JOIN и ORDER BY.

|  |
| --- |
| CREATE PROCEDURE GetStudentMarks  AS  BEGIN  SELECT  student.surname,  student.name,  student.kurs,  exam\_marks.mark  FROM  student  JOIN  exam\_marks ON student.idst = exam\_marks.idst  ORDER BY  student.surname,  student.name;  END |

Листинг 11 – хранимая процедура.

|  |
| --- |
| EXEC GetStudentMarks; |

Листинг 12 – вывод хранимой процедуры.

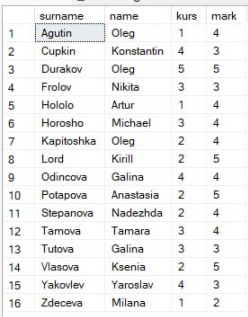


Рис.7 – вывод хранимой процедуры.

Первый триггер написан для обновления оценки по предмету.

|  |
| --- |
| CREATE TRIGGER UpdateSubjectGrade  ON exam\_marks  AFTER UPDATE  AS  BEGIN  IF UPDATE(Grade)  BEGIN  UPDATE exam\_marks  SET Grade = CASE  WHEN Grade < 0 THEN 0  WHEN Grade > 100 THEN 100  ELSE Grade  END  WHERE ExamID IN (SELECT ExamID FROM inserted)  END  END |

Листинг 13 – первый триггер.

Второй триггер написан для вывода имени и фамилии только с большой буквы, даже если в базу их запишут с маленькой буквы.

|  |
| --- |
| CREATE TRIGGER FormatStudentNames  ON student  FOR INSERT, UPDATE  AS  BEGIN  SET NOCOUNT ON;  UPDATE s  SET s.surname = UPPER(LEFT(s.surname, 1)) + LOWER(SUBSTRING(s.surname, 2, LEN(s.surname))),  s.name = UPPER(LEFT(s.name, 1)) + LOWER(SUBSTRING(s.name, 2, LEN(s.name)))  FROM inserted i  INNER JOIN student s ON i.idst = s.idst;  END |

Листинг 14 – второй триггер.

## **3.4 Разработка приложения.**

### **3.4.1 Разработка графического интерфейса**

Для начала разработки приложения нам нужно проанализировать какие окна у нас будут задействованы и активны. Так как мы разрабатываем приложение для пиццерии нам необходимо понимать, что для приложения необходимы такие окна как:

1. Окно авторизации,
2. Окно с переходом на другие окна,
3. Окна, содержащие информацию в виде таблиц.

После того как мы проанализировали это – можем приступить к разработке.

Первым мы создали окно авторизации, на котором расположены элементы для ввода логина и пароля и кнопка для входа в систему. При нажатии на кнопку мы переходим на окно с переходами (Рис.8).

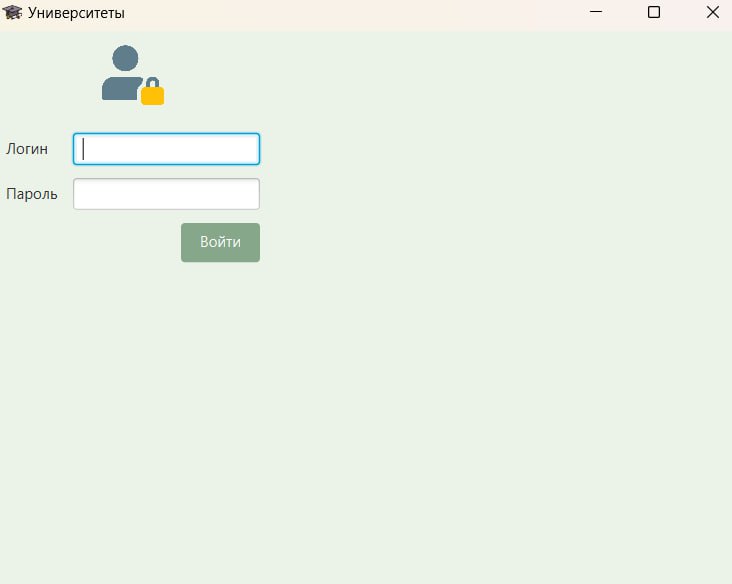
****

Рис.8 – окно авторизации.

Следующим создается окно регистрации, с помощью которого мы переходим на другие окна с таблицами. (Рис.9).

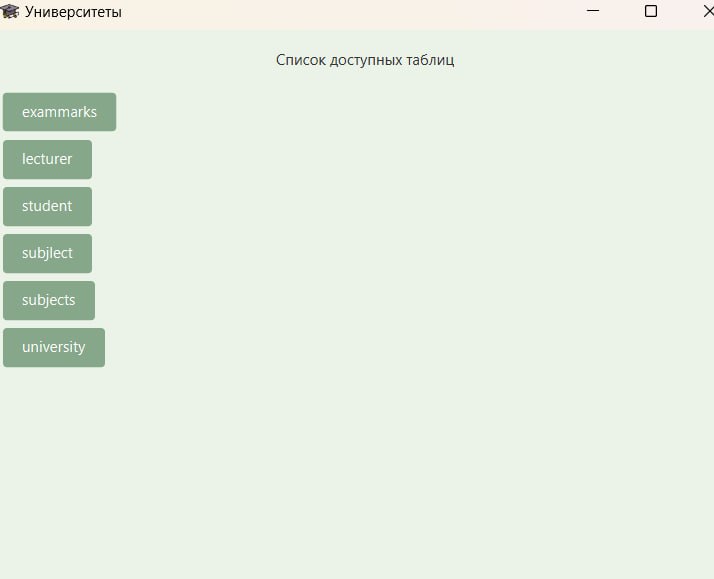


Рис.9 – переходное окно.

После этого мы занимаемся разработкой тех самых окно с таблицами, на которых можно будет изменить, сохранить, удалить или добавить записи в таблицы. Рассмотрим, как это выглядит на примере одного окна (Рис.10).

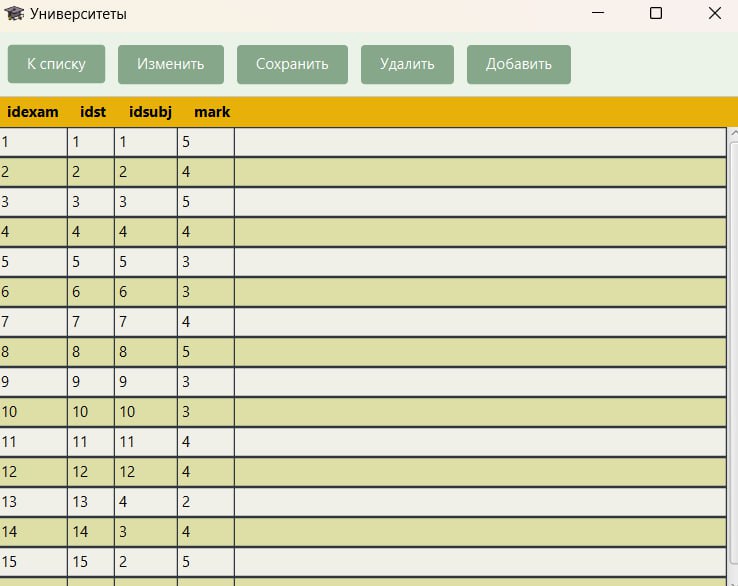


Рис.10 – окно с таблицей.

### **3.4.2 Разработка логики приложения и подключение БД.**

С помощью класса Main запускается приложение, созданное при помощи JavaFX.

|  |
| --- |
| package com.example.university;  import javafx.application.Application;  import javafx.stage.Stage;  public class Launcher extends Application {  public static void main(String[] args) {  launch(args);  }  @Override  public void start(Stage mainStage) {  UniversityApp universityApp = new UniversityApp();  universityApp.start(mainStage);  }  } |

Листинг 15 – класс Main.

DataBaseUtils - класс, отвечающий за взаимодействие с базой данных. Например, получение всех названий таблиц из схемы dbo, получение строк из таблицы через "SELECT \* FROM ...", получение названий колонок из таблицы, обновления данных в таблице и т.д.

|  |
| --- |
| package com.example.university;  import java.sql.Connection;  import java.sql.ResultSet;  import java.sql.SQLException;  import java.sql.Statement;  import java.text.ParseException;  import java.text.SimpleDateFormat;  import java.util.ArrayList;  import java.util.Date;  import java.util.List;  public class DataBaseUtils {  public static List<String> getTableNames(Connection connection) {  List<String> tableNames = new ArrayList<>();  try (ResultSet resultSet = connection.getMetaData().getTables(null, "dbo", "%", new String[]{"TABLE"})) {  while (resultSet.next()) {  tableNames.add(resultSet.getString("TABLE\_NAME"));  }  } catch (SQLException e) {  handleSQLException(e);  }  return tableNames;  }  public static List<List<String>> getTableData(Connection connection, String tableName) {  List<List<String>> tableData = new ArrayList<>();  try (Statement statement = connection.createStatement();  ResultSet resultSet = statement.executeQuery("SELECT \* FROM " + tableName)) {  int columnCount = resultSet.getMetaData().getColumnCount();  while (resultSet.next()) {  List<String> rowData = new ArrayList<>();  for (int i = 1; i <= columnCount; i++) {  rowData.add(resultSet.getString(i));  }  tableData.add(rowData);  }  } catch (SQLException e) {  handleSQLException(e);  }  return tableData;  }  public static String getFirstColumnName(Connection connection, String tableName) {  try (ResultSet resultSet = connection.getMetaData().getColumns(null, null, tableName, null)) {  if (resultSet.next()) {  return resultSet.getString("COLUMN\_NAME");  }  } catch (SQLException e) {  handleSQLException(e);  }  return null;  }  public static List<String> getColumns(Connection connection, String tableName) {  List<String> columnNames = new ArrayList<>();  try (ResultSet resultSet = connection.getMetaData().getColumns(null, null, tableName, null)) {  while (resultSet.next()) {  columnNames.add(resultSet.getString("COLUMN\_NAME"));  }  } catch (SQLException e) {  handleSQLException(e);  }  return columnNames;  }  public static void updateTableData(Connection connection, String tableName, List<List<String>> newData) {  try {  List<String> columnNames = getColumns(connection, tableName);  for (List<String> rowData : newData) {  String originalId = rowData.get(0);  String updateQuery = buildUpdateQuery(connection, tableName, columnNames, rowData, originalId);  executeUpdateQuery(connection, updateQuery);  }  } catch (SQLException e) {  handleSQLException(e);  }  }  private static void executeUpdateQuery(Connection connection, String query) throws SQLException {  try (Statement statement = connection.createStatement()) {  statement.executeUpdate(query);  }  }  private static String buildUpdateQuery(Connection connection, String tableName, List<String> columnNames, List<String> rowData, String originalId) {  List<String> columnValuePairs = new ArrayList<>();  for (int i = 1; i < columnNames.size(); i++) {  String columnName = columnNames.get(i);  String columnValue = rowData.get(i);  if (isDateTimeColumn(connection, tableName, columnName)) {  String[] dateFormats = {"yyyy-MM-dd HH:mm:ss", "yyyy-MM-dd"};  Date parsedDate = null;  for (String dateFormat : dateFormats) {  try {  SimpleDateFormat sdf = new SimpleDateFormat(dateFormat);  parsedDate = sdf.parse(columnValue);  if (parsedDate != null) {  break;  }  } catch (ParseException e) {  //  }  }  if (parsedDate != null) {  columnValue = new SimpleDateFormat("yyyyMMdd HH:mm:ss").format(parsedDate);  } else {  System.out.println("Error: " + columnValue);  }  }  columnValuePairs.add(columnName + " = N'" + columnValue + "'");  }  String idCol = getFirstColumnName(connection, tableName);  return "UPDATE " + tableName + " SET " +  String.join(", ", columnValuePairs) +  " WHERE " + idCol + " = '" + originalId + "'";  }  public static void removeRows(Connection connection, String tableName, List<List<String>> rowsToRemove) {  try {  for (List<String> rowData : rowsToRemove) {  String originalId = rowData.get(0);  String idCol = getFirstColumnName(connection, tableName);  String deleteQuery = "DELETE FROM " + tableName + " WHERE " + idCol + "= '" + originalId + "'";  executeUpdateQuery(connection, deleteQuery);  }  } catch (SQLException e) {  handleSQLException(e);  }  }  public static void insertRows(Connection connection, String tableName, List<List<String>> newData) {  try {  List<String> columnNames = getColumns(connection, tableName);  for (List<String> rowData : newData) {  String insertQuery = buildInsertQuery(tableName, columnNames, rowData);  executeUpdateQuery(connection, insertQuery);  }  } catch (SQLException e) {  handleSQLException(e);  }  }  private static String buildInsertQuery(String tableName, List<String> columnNames, List<String> rowData) {  String columns = String.join(", ", columnNames.subList(1, columnNames.size()));  String values = String.join(", ", rowData.subList(1, rowData.size()).stream()  .map(val -> "N'" + val + "'")  .toArray(String[]::new));  return "INSERT INTO " + tableName + " (" + columns + ") VALUES (" + values + ")";  }  private static boolean isDateTimeColumn(Connection connection, String tableName, String columnName) {  try (ResultSet resultSet = connection.getMetaData().getColumns(null, null, tableName, columnName)) {  if (resultSet.next()) {  int dataType = resultSet.getInt("DATA\_TYPE");  return dataType == java.sql.Types.TIMESTAMP || dataType == java.sql.Types.DATE;  }  } catch (SQLException e) {  handleSQLException(e);  }  return false;  }  private static void handleSQLException(SQLException e) {  e.printStackTrace();  }  } |

Листинг 16 – DataBaseUtils.

DataBaseConnection - класс для установления соединения к базе данных.

|  |
| --- |
| package com.example.university;  import java.sql.Connection;  import java.sql.DriverManager;  import java.sql.SQLException;  public class DataBaseConnection {  static String username;  static String password;  public static Connection connect() {  String database = "university";  String SERVER\_IP = "jdbc:sqlserver://localhost:1433;databaseName=" + database + ";characterEncoding=UTF8;encrypt=true;" +  "trustServerCertificate=true;";  try {  return DriverManager.getConnection(SERVER\_IP, username, password);  } catch (SQLException ex) {  throw new RuntimeException("Не удалось установить соединение с БД", ex);  }  }  } |

Листинг 17 – DataBaseConnection.

UniversityApp - класс, содержащий основную логику приложения и выводящий первое окно для входа.

|  |
| --- |
| package com.example.university;  import javafx.geometry.HPos;  import javafx.geometry.Insets;  import javafx.geometry.Pos;  import javafx.scene.Scene;  import javafx.scene.control.Button;  import javafx.scene.control.Label;  import javafx.scene.control.PasswordField;  import javafx.scene.control.TextField;  import javafx.scene.image.Image;  import javafx.scene.image.ImageView;  import javafx.scene.layout.GridPane;  import javafx.scene.layout.VBox;  import javafx.stage.Stage;  import java.sql.Connection;  import java.sql.SQLException;  import java.util.ArrayList;  import java.util.List;  public class UniversityApp {  private TextField username;  private PasswordField password;  private Stage mainStage;  private Connection connection;  public void start(Stage mainStage) {  this.mainStage = mainStage;  GridPane grid = createGridPane();  UniversityAppUI.setupScene(mainStage, grid);  mainStage.show();  }  private GridPane createGridPane() {  Image image = new Image("file:img/login.png");  ImageView imageView = new ImageView(image);  imageView.setFitWidth(50);  imageView.setFitHeight(50);  GridPane.setConstraints(imageView, 0, 0, 2, 1);  GridPane.setHalignment(imageView, HPos.CENTER);  // Other components  Label usernameLabel = new Label("Логин");  GridPane.setConstraints(usernameLabel, 0, 2);  username = new TextField();  GridPane.setConstraints(username, 1, 2);  Label passwordLabel = new Label("Пароль");  GridPane.setConstraints(passwordLabel, 0, 3);  password = new PasswordField();  GridPane.setConstraints(password, 1, 3);  Button loginButton = new Button("Войти");  GridPane.setConstraints(loginButton, 1, 4);  GridPane.setHalignment(loginButton, javafx.geometry.HPos.RIGHT);  loginButton.setOnAction(e -> handleLogin());  GridPane grid = new GridPane();  grid.setPadding(new Insets(10, 10, 10, 10));  grid.setVgap(10);  grid.setHgap(12);  grid.getStyleClass().add("login");  grid.getChildren().addAll(imageView, usernameLabel, username, passwordLabel, password, loginButton);  return grid;  }  private void handleLogin() {  String username = this.username.getText();  String password = this.password.getText();  try {  DataBaseConnection.username = username;  DataBaseConnection.password = password;  connection = DataBaseConnection.connect();  if (connection.isValid(3)) {  System.out.println("Подключился к базе данных");  showTables();  } else {  System.out.println("Не смог подключиться к базе данных");  }  } catch (SQLException e) {  e.printStackTrace();  }  }  private List<Button> generateTableButtons(List<String> tableNames) {  List<Button> buttons = new ArrayList<>();  for (String tableName : tableNames) {  Button btn = new Button(tableName);  btn.setOnAction(e -> handleTableButtonClick(tableName));  buttons.add(btn);  }  return buttons;  }  private void handleTableButtonClick(String tableName) {  try (Connection connection = DataBaseConnection.connect()) {  List<String> columnNames = DataBaseUtils.getColumns(connection, tableName);  List<List<String>> tableData = DataBaseUtils.getTableData(connection, tableName);  GridPane tableScene = Table.createTable(tableName, columnNames, tableData, this::showTables);  UniversityAppUI.setupTableScene(mainStage, tableScene);  } catch (SQLException e) {  e.printStackTrace();  }  }  private void showTables() {  List<String> tableNames = DataBaseUtils.getTableNames(connection);  List<Button> tableButtons = generateTableButtons(tableNames);  Scene tableScene = createTableScene(tableButtons);  UniversityAppUI.changeCSS(tableScene);  mainStage.setScene(tableScene);  }  private Scene createTableScene(List<Button> tableButtons) {  GridPane grid = new GridPane();  grid.setPadding(new Insets(10, 10, 10, 10));  grid.setVgap(7);  grid.setHgap(9);  for (int i = 0; i < tableButtons.size(); i++) {  grid.add(tableButtons.get(i), 0, i);  }  Label label = new Label("Список доступных таблиц");  label.getStyleClass().add("scene-label");  label.setMaxHeight(30);  label.setPadding(new Insets(15, 0, 0, 0));  VBox layout = new VBox(7);  layout.getChildren().addAll(label, grid);  layout.setAlignment(Pos.TOP\_CENTER);  Scene tableScene = new Scene(layout, 600, 450);  UniversityAppUI.changeCSS(tableScene);  return tableScene;  }  } |

Листинг 18 – UniversityApp.

UniversityAppUI - класс для хранения методов, отвечающих за интерфейс приложения. Например, там находитя метод changeCSS, который применяет css файл к сцене.

|  |
| --- |
| package com.example.university;  import javafx.scene.Scene;  import javafx.scene.image.Image;  import javafx.scene.layout.GridPane;  import javafx.stage.Stage;  import java.util.Objects;  public class UniversityAppUI {  public static void changeCSS(Scene scene) {  scene.getStylesheets().add(Objects.requireNonNull(UniversityAppUI.class.getResource("/style/style.css")).toExternalForm());  }  public static void changeCSS(GridPane gridPane) {  gridPane.getStylesheets().add(Objects.requireNonNull(UniversityAppUI.class.getResource("/style/style.css")).toExternalForm());  }  public static void setupScene(Stage mainStage, GridPane grid) {  mainStage.setTitle("Университеты");  mainStage.getIcons().add(new Image("file:img/logotip.png"));  Scene scene = new Scene(grid, 600, 450);  changeCSS(scene);  mainStage.setScene(scene);  }  public static void setupTableScene(Stage mainStage, GridPane tableScene) {  Scene scene = new Scene(tableScene, 600, 450);  changeCSS(scene);  mainStage.setScene(scene);  }  } |

Листинг 19 – UniversityAppUI.

Table - класс для работы с данными таблицы.

|  |
| --- |
| package com.example.university;  import javafx.beans.property.SimpleStringProperty;  import javafx.collections.FXCollections;  import javafx.collections.ObservableList;  import javafx.event.ActionEvent;  import javafx.event.EventHandler;  import javafx.geometry.Insets;  import javafx.scene.control.Button;  import javafx.scene.control.Control;  import javafx.scene.control.TableColumn;  import javafx.scene.control.TableView;  import javafx.scene.control.cell.TextFieldTableCell;  import javafx.scene.layout.ColumnConstraints;  import javafx.scene.layout.GridPane;  import javafx.scene.layout.Priority;  import javafx.scene.layout.RowConstraints;  import java.sql.Connection;  import java.sql.SQLException;  import java.util.ArrayList;  import java.util.List;  public class Table {  public static GridPane createTable(String tableName, List<String> columnNames, List<List<String>> tableData,  Runnable backButtonAction) {  TableView<ObservableList<String>> tableView = createTableView(columnNames);  configureTableColumns(tableView.getColumns());  ObservableList<ObservableList<String>> data = createTableData(tableData);  tableView.setItems(data);  GridPane buttonPane = createButtonPane(backButtonAction, tableName, tableView);  GridPane gridPane = createGridPane(tableView, buttonPane);  return gridPane;  }  private static TableView<ObservableList<String>> createTableView(List<String> columnNames) {  TableView<ObservableList<String>> tableView = new TableView<>();  tableView.getStyleClass().add("table");  for (int i = 0; i < columnNames.size(); i++) {  final int columnIndex = i;  TableColumn<ObservableList<String>, String> column = new TableColumn<>(columnNames.get(i));  column.setCellValueFactory(data -> new SimpleStringProperty(data.getValue().get(columnIndex)));  tableView.getColumns().add(column);  }  return tableView;  }  private static void configureTableColumns(List<TableColumn<ObservableList<String>, ?>> columns) {  for (TableColumn<ObservableList<String>, ?> column : columns) {  TableColumn<ObservableList<String>, String> stringColumn = (TableColumn<ObservableList<String>, String>) column;  stringColumn.setCellFactory(TextFieldTableCell.forTableColumn());  stringColumn.setOnEditCommit(t -> {  ObservableList<String> row = t.getTableView().getItems().get(t.getTablePosition().getRow());  row.set(t.getTablePosition().getColumn(), t.getNewValue());  });  }  }  private static ObservableList<ObservableList<String>> createTableData(List<List<String>> tableData) {  ObservableList<ObservableList<String>> data = FXCollections.observableArrayList();  for (List<String> rowData : tableData) {  data.add(FXCollections.observableArrayList(rowData));  }  return data;  }  private static GridPane createButtonPane(Runnable backButtonAction, String tableName, TableView<ObservableList<String>> tableView) {  GridPane buttonPane = new GridPane();  buttonPane.setHgap(10);  buttonPane.setVgap(10);  buttonPane.setPadding(new Insets(10, 10, 10, 10));  Button backButton = createButton("К списку", e -> backButtonAction.run());  Button editButton = createButton("Изменить", e -> tableView.setEditable(true));  Button saveButton = createButton("Сохранить", e -> handleSave(tableView, tableName));  Button deleteButton = createButton("Удалить", e -> handleDeleteRow(tableView, tableName));  Button addButton = createButton("Добавить", e -> handleAddRow(tableView));  GridPane.setConstraints(backButton, 0, 0);  GridPane.setConstraints(editButton, 1, 0);  GridPane.setConstraints(saveButton, 2, 0);  GridPane.setConstraints(deleteButton, 3, 0);  GridPane.setConstraints(addButton, 4, 0);  buttonPane.getChildren().addAll(backButton, editButton, saveButton, deleteButton, addButton);  return buttonPane;  }  private static Button createButton(String text, EventHandler<ActionEvent> action) {  Button button = new Button(text);  button.setOnAction(action);  button.getStyleClass().add("button");  return button;  }  private static void handleSave(TableView<ObservableList<String>> tableView, String tableName) {  List<List<String>> updatedData = new ArrayList<>();  List<List<String>> newData = new ArrayList<>();  for (ObservableList<String> rowData : tableView.getItems()) {  if (rowData.contains("")) {  newData.add(new ArrayList<>(rowData));  } else {  updatedData.add(new ArrayList<>(rowData));  }  }  try (Connection connection = DataBaseConnection.connect()) {  if (!newData.isEmpty()) {  DataBaseUtils.insertRows(connection, tableName, newData);  }  if (!updatedData.isEmpty()) {  DataBaseUtils.updateTableData(connection, tableName, updatedData);  }  } catch (SQLException e) {  e.printStackTrace();  }  }  private static GridPane createGridPane(TableView<ObservableList<String>> tableView, GridPane buttonPane) {  GridPane gridPane = new GridPane();  ColumnConstraints column1 = new ColumnConstraints();  column1.setHgrow(javafx.scene.layout.Priority.ALWAYS);  gridPane.getColumnConstraints().addAll(column1);  RowConstraints row1 = new RowConstraints();  row1.setMaxHeight(Control.USE\_COMPUTED\_SIZE);  row1.setMinHeight(Control.USE\_COMPUTED\_SIZE);  row1.setVgrow(Priority.NEVER);  RowConstraints row2 = new RowConstraints();  row2.setVgrow(Priority.ALWAYS);  gridPane.getRowConstraints().addAll(row1, row2);  gridPane.add(buttonPane, 0, 0);  gridPane.add(tableView, 0, 1);  return gridPane;  }  private static void handleDeleteRow(TableView<ObservableList<String>> tableView, String tableName) {  ObservableList<ObservableList<String>> selectedRows = tableView.getSelectionModel().getSelectedItems();  List<List<String>> rowsToRemove = new ArrayList<>();  for (ObservableList<String> selectedRow : selectedRows) {  rowsToRemove.add(new ArrayList<>(selectedRow));  }  try (Connection connection = DataBaseConnection.connect()) {  DataBaseUtils.removeRows(connection, tableName, rowsToRemove);  } catch (SQLException e) {  e.printStackTrace();  }  tableView.getItems().removeAll(selectedRows);  }  private static void handleAddRow(TableView<ObservableList<String>> tableView) {  ObservableList<String> newRow = FXCollections.observableArrayList();  for (int i = 0; i < tableView.getColumns().size(); i++) {  newRow.add("");  }  tableView.getItems().add(newRow);  }  } |

Листинг 20 – Table.

# **4. Руководство администратора**

Когда пользователь запускает приложение он должен ввести свой логин и пароль (Рис.11).

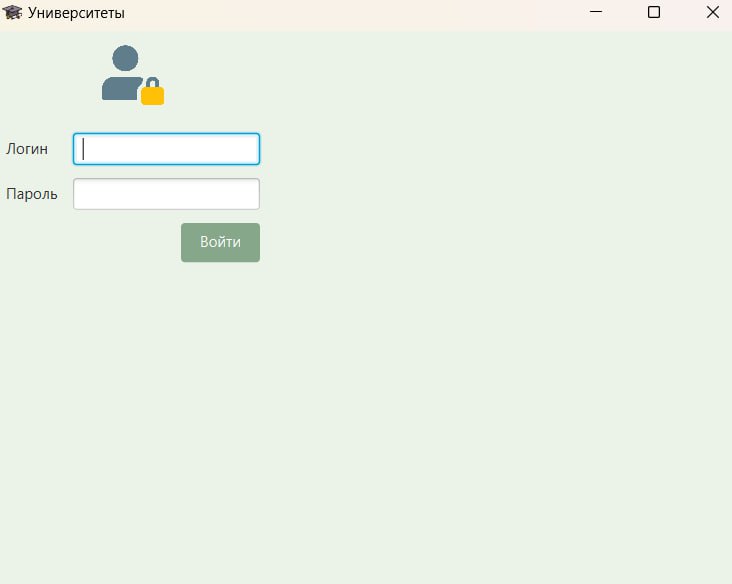


Рис.11 – окно авторизации.

Когда пользователь нажимает на одну из кнопок, например на кнопку «lecturer», тогда появится новое окно с таблицей, куда занесена информация о преподавателях. Аналогично и с другими кнопками (Рис.12).

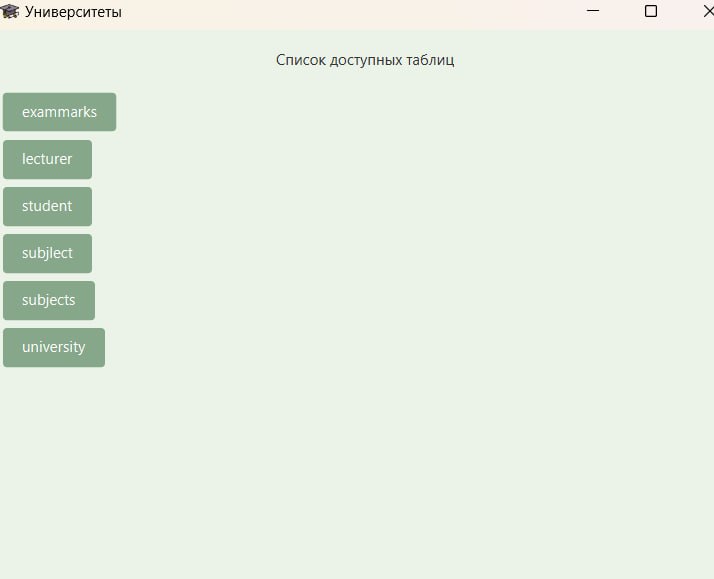


Рис. 12 – активные кнопки.

На форме с таблицей тоже есть 4 кнопки «Сохранить», «Изменить», «Добавить» и «Удалить» (Рис.13).

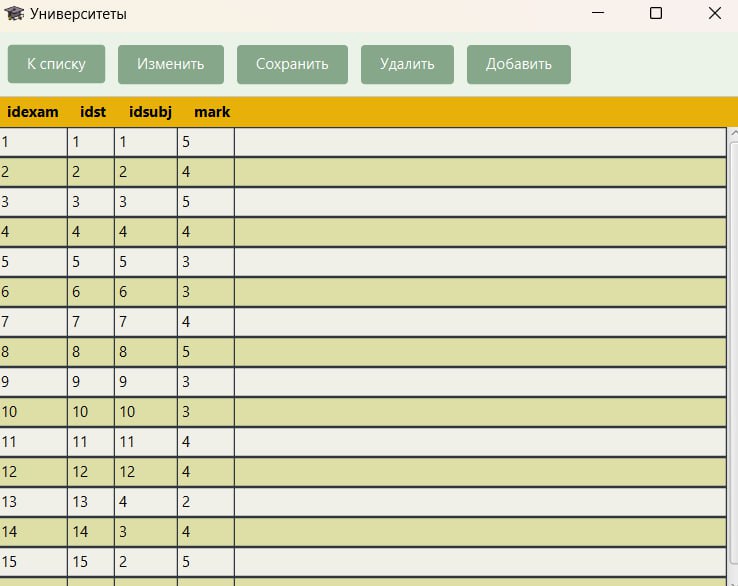


Рис.13 – кнопки действий.

Если нажать на кнопку изменить, то можно изменить любую имеющуюся запись. При нажатии на добавить – добавление, удалить – удаление существующей записи. Кнопка сохранить используется для сохранения каких-либо изменений.

# **Заключение**

В данной курсовой работе была создана база данных по предметной области «Университеты» с помощью инструмента для визуального проектирования баз данных SSMS, спроектирована ERD-модель, создан интерфейс приложения с помощью JavaFX и база данных соединена с интерфейсом.

Из полученных результатов в результате работы над курсовой работой можно сделать вывод, что цель достигнута и все поставленные задачи выполнены.

# **Список литературы**

1. [ВУЗ: организационная структура и особые статусы (akvobr.ru)](https://akvobr.ru/vuz_organizacionnaya_struktura_i_osobye_statusy.html)
2. [Download SQL Server Management Studio (SSMS) - SQL Server Management Studio (SSMS) | Microsoft Learn](https://learn.microsoft.com/en-us/sql/ssms/download-sql-server-management-studio-ssms?view=sql-server-ver16)
3. [Знакомство с запросами - Служба поддержки Майкрософт (microsoft.com)](https://support.microsoft.com/ru-ru/office/%D0%B7%D0%BD%D0%B0%D0%BA%D0%BE%D0%BC%D1%81%D1%82%D0%B2%D0%BE-%D1%81-%D0%B7%D0%B0%D0%BF%D1%80%D0%BE%D1%81%D0%B0%D0%BC%D0%B8-a9739a09-d3ff-4f36-8ac3-5760249fb65c)
4. [CREATE TRIGGER (Transact-SQL) - SQL Server | Microsoft Learn](https://learn.microsoft.com/ru-ru/sql/t-sql/statements/create-trigger-transact-sql?view=sql-server-ver16)