МИНИСТЕРСТВО ОБРАЗОВАНИЯ РЕСПУБЛИКИ БЕЛАРУСЬ

Учреждение образования «БЕЛОРУССКИЙ ГОСУДАРСТВЕННЫЙ

ТЕХНОЛОГИЧЕСКИЙ УНИВЕРСИТЕТ»

Факультет Информационных технологий

Кафедра Информационные системы и технологии

Специальность 1–40 05 01 Информационные системы и технологии

Специализация Базы данных

**ПОЯСНИТЕЛЬНАЯ ЗАПИСКА**

**К КУРСОВОМУ ПРОЕКТУ НА ТЕМУ:**

«База данных для программного средства «Автовокзал» с применением технологии мониторинга базы данных»

Выполнил студент Телица Анна Дмитриевна

(Ф.И.О.)

Руководитель проекта асс. Н.И. Уласевич

(учен. степень, звание, должность, Ф.И.О., подпись)

Консультант: асс. Н.И. Уласевич.

(учен. степень, звание, должность, Ф.И.О., подпись)

Нормоконтролер: асс. Н.И. Уласевич.

(учен. степень, звание, должность, Ф.И.О., подпись)

Курсовой проект защищен с оценкой

Содержание

[Введение 5](#_Toc167059166)

[1 Постановка задачи 6](#_Toc167059167)

[1.1 Описание функционала и объектов базы данных 6](#_Toc167059168)

[1.2 Описание используемых технологий 7](#_Toc167059169)

[2 Проектирование базы данных 8](#_Toc167059170)

[2.1 Описание структуры базы данных 8](#_Toc167059171)

[3 Разработка объектов базы данных 10](#_Toc167059172)

[3.1 Таблицы 10](#_Toc167059173)

[3.2 Представления 11](#_Toc167059174)

[3.3 Индексы 12](#_Toc167059175)

[3.4 Тригеры 12](#_Toc167059176)

[3.5 Функции 14](#_Toc167059177)

[3.6 Процедуры 14](#_Toc167059178)

[3.7 Пользователи 15](#_Toc167059179)

[3.8 Вывод 15](#_Toc167059180)

[4 Описание процедур импорта и экспорта 16](#_Toc167059181)

[4.1 Процедура импорта данных из XML-файла 16](#_Toc167059182)

[4.1 Описание экспорта данных 16](#_Toc167059183)

[4.3 Вывод 17](#_Toc167059184)

[5 Тестирование производительности 18](#_Toc167059185)

[5.1 Тестирование производительности базы данных 18](#_Toc167059186)

[Заключение 20](#_Toc167059187)

[Список используемых источников 21](#_Toc167059188)

[Приложение А Листинг создания таблиц 22](#_Toc167059189)

[Приложение Б Листинг создания триггеров 24](#_Toc167059190)

[Приложение В Листинг процедуры для заполнения таблицы 26](#_Toc167059191)

[Приложение Г Листинг процедуры извлечение информации из файла 27](#_Toc167059192)

[Приложение Д Листинг создания процедур 28](#_Toc167059193)

[Приложение Е Листинг создания представлений 36](#_Toc167059194)

# Введение

В современном мире транспорт играет важную роль в обеспечении мобильности и связи между различными регионами. Автовокзалы являются ключевым звеном в пассажирской транспортной инфраструктуре, обеспечивая перевозку пассажиров на дальние расстояния. В связи с этим, эффективное управление и организация данных в базе данных автовокзала имеет важное значение для обеспечения качественного обслуживания пассажиров, оптимизации операций и анализа ключевых показателей.  
Данная база данных предназначена для хранения информации о маршрутах, расписании движения автобусов, билетах, пассажирах, автобусах и их водителях.   
 В ходе выполнения данного проекта будет осуществлена аналитика требований, проектирование схемы базы данных, создание необходимых объектов, разработка SQL-запросов и интерфейса для взаимодействия с базой данных. Для обеспечения функциональности приложения используются технология разработки системы мониторинга за состоянием базы данных. Результатом данного проекта станет создание функциональной базы данных, способной эффективно поддерживать операционную деятельность автовокзала, обеспечивая надежное хранение и быстрый доступ к информации.   
 В пояснительной записке содержится информация о сопоставимых продуктах, структуре и реализации проекта, а также инструкции по использованию приложения.

# Постановка задачи

* 1. **Описание функционала и объектов базы данных**

Целью данного курсового проекта является проектирование базы данных для автовокзала с применением технологии мониторинга базы данных. В качестве модели данных следует использовать реляционную модель. Проектирование необходимо произвести таким образом, чтобы конечные данные соответствовали общим требованиям к информации в базе данных

Функционал должен позволять:

* Подсчета общей стоимости билетов пользователя;
* Поиска всех билетов, приобретенных пассажиром;
* Подсчета количества билетов, приобретенных на определенное направление;
* Поиска всех рейсов для определенного направления;
* Поиска всех билетов, которые были куплены в указанную дату;
* Заказ транспорта
* Просмотр свободных мест;
* Отмена заказанного транспорта;
* Добавление новых автобусов;
* Добавление новых водителей;
* Добавление новых пассажиров;
* Добавление новых маршрутов;
* Добавление новых рейсов;
* Добавление новых пользователей;
* Добавление новых билетов;
* Удаление автобусов;
* Удаление водителей;
* Удаление пассажиров;
* Удаление маршрутов;
* Удаление рейсов;
* Удаление пользователей;
* Удаление билетов;

Для реализации функционала необходимо разработать ряд объектов базы данных. К таким объектам относятся:

* роли;
* пользователи;
* таблицы;
* представления;
* индексы;
* хранимые процедуры;
* функции;
* триггеры.
  1. **Описание используемых технологий**

Существует огромное количество систем управления базами данных (СУБД), которые могут быть использованы для организации работы с данными в компьютерных системах. Некоторые из наиболее популярных СУБД включают в себя Oracle Database, MS SQL Server, NoSQL, MySQL, PostgreSQL и другие.

В данной работе было принято решение использовать MS SQL Server в качестве СУБД, так как она предоставляет широкий спектр возможностей для оперирования объектами базы данных, включая поддержку стандарта SQL, хранение и управление табличными данными, поддержку транзакций и многое другое.

В работе с базой данных автовокзала, особенное внимание уделяется мониторингу, так как на сервисе всегда должна отображаться актуальная информация, для того чтобы не вводить пользователей в заблуждения и избежать негативных отзывов на продукт. Поэтому в реализуемой базе данных было принято решение использовать технологию мониторинга за состоянием базы данных.

# Проектирование базы данных

## Описание структуры базы данных

Диаграмма структуры полученной базы данных, представлена на рисунке 2.1. Для базы данных было разработано 7 таблиц, которые связаны друг с другом внешними ключами.

Roles - таблица, содержащая роли пользователей. В этой таблице будут храниться записи с ролями, такими как администратор и пользователь.

Users - таблица, содержащая данные о пользователе. В этой таблице будут храниться информация о пользователе, такая как имя, фотография, электронная почта, дата рождения и т.д.

Drivers - таблица, содержащая информацию о водителях. В этой таблице будут храниться личные данные водителей такие как фамилия, имя, отчество, возраст, номер телефона, а также номер лицензии на вождение транспортом.

Buses - таблица, содержащая информацию о автобусах. В этой таблице будут храниться данные о марке автобуса, году выпуска транспорта и количество пассажирских мест.

Passengers- таблица, содержащая информацию о пассажирах. В ней будут храниться личные данные пассажира, такие как фамилия, имя, номер телефона и т.д.

Route - таблица, содержащая информацию о маршрутах транспорта. В этой таблице будет храниться место отправления и место прибытия автобусов, а также остановки по дороге.

Directions - таблица, содержащая информацию о доступных маршрутах. В этой таблице будут храниться данные о времени и датах прибытия и отправления.

Tickets - таблица, содержащая отношение между почти всеми таблицами. В этой таблице будут храниться записи, связывающие билет с пассажиром, маршрутом, водителем, автобусом, а также данных о стоимости билета и дате покупки.

Все таблицы связаны между собой с помощью внешних ключей, что позволяет эффективно и быстро обрабатывать данные и выполнять запросы к базе данных.

Для более наглядного представления возможностей каждой роли в системе, необходимо предоставить диаграмму вариантов использования продукта, показа она на рисунке 2.2. На данной диаграмме должны быть отображены все возможности, которые доступны конкретному пользователю с учетом его роли, будь это обычный пользователь или администратор.

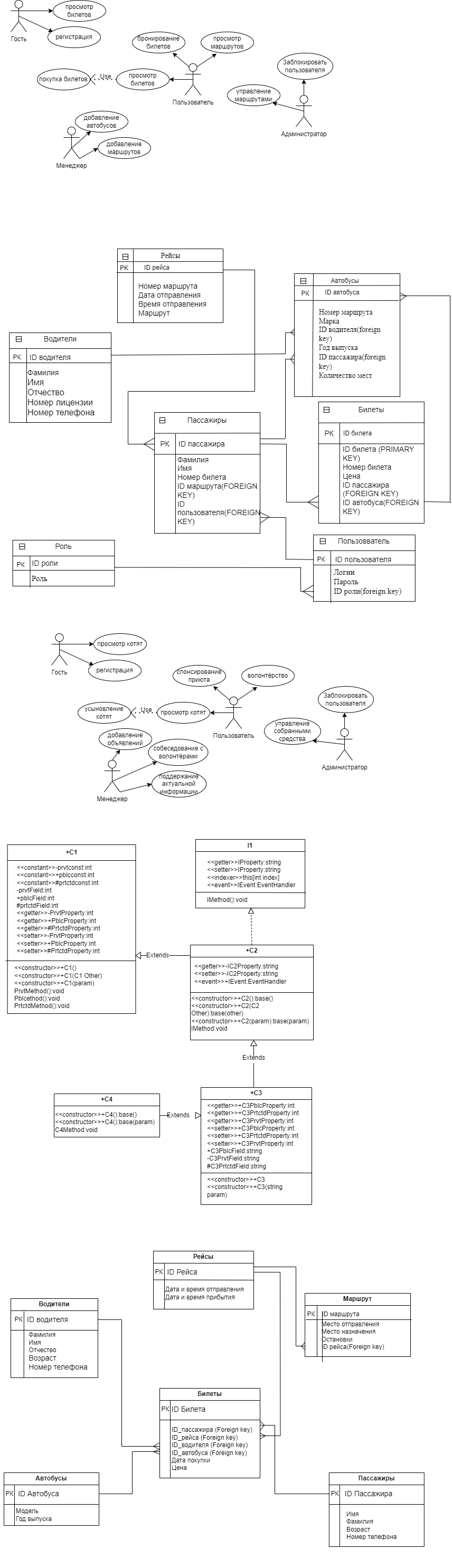


Рисунок 2.1 – Диаграмма структуры базы данных

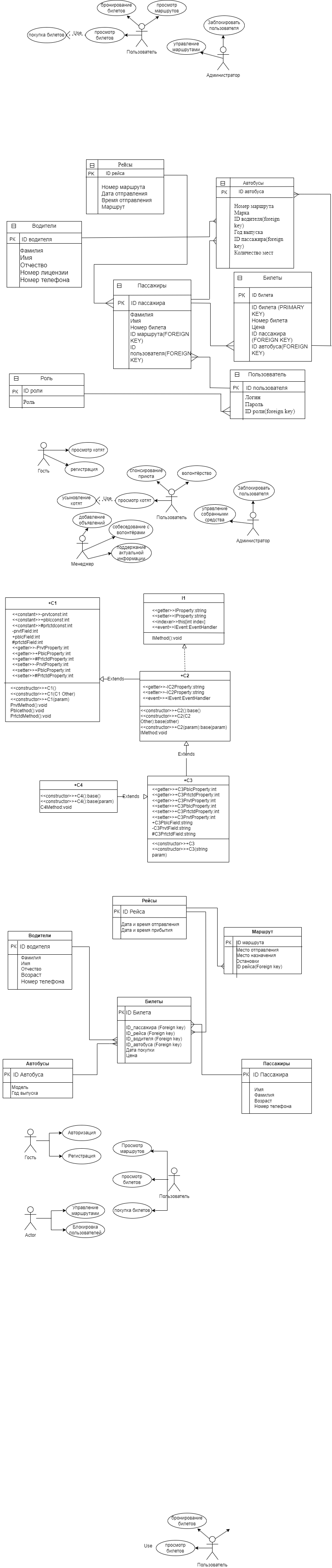


Рисунок 2.1 – Диаграмма вариантов использования

Такая диаграмма позволит легче воспринимать возможности каждой роли и понимать, какие возможности расширяются при переходе на более высокую роль в системе.

# Разработка объектов базы данных

## Таблицы

Таблицы являются неотъемлемой частью любой реляционной базы данных. Краткая характеристика каждой из таблиц была предоставлена в разделе 2, а код их создания можно увидеть в Приложении А. Ниже мы рассмотрим каждую таблицу подробнее.

Таблица Buses состоит из четырёх столбцов и представлена на таблице 3.1:

Таблица 3.1 «Buses»

|  |  |  |
| --- | --- | --- |
| Название | Тип данных | Описание |
| ID\_bus | int | Хранит идентификатор автобуса |
| Bus\_brand | varchar | Хранит марку автобуса |
| Release\_year | date | Хранит год выпуска автобуса |
| Seats\_number | int | Хранит количество мест в автобусе |

Таблица Drivers состоит из восьми столбцов и показана на таблице 3.2:

Таблица 3.2 «Drivers»

|  |  |  |
| --- | --- | --- |
| Название | Тип данных | Описание |
| ID\_driver | int | Хранит идентификатор водителя |
| Surname | varchar | Хранит фамилию водителя |
| Name | varchar | Хранит имя водителя |
| Patronymic | varchar | Хранит отчество водителя |
| License\_number | int | Хранит номер лицензии водителя |
| Telephone\_number | varchar | Хранит номер телефона водителя |
| Age | int | Хранит возраст водителя |

Таблица Passengers состоит из пяти столбцов и представлена на таблице 3.3:

Таблица 3.3 «Passengers»

|  |  |  |
| --- | --- | --- |
| Название | Тип данных | Описание |
| ID\_passenger | int | хранит идентификатор пассажира |
| Surname | varchar | хранит фамилию пассажира |
| Name | varchar | хранит имя пассажира |
| Telephone\_number | varchar | Хранит номер телефона пассажира |
| Age | int | хранит возраст пассажира |

Таблица Directions состоит из пяти столбцов и представлена на таблице 3.4:

Таблица 3.4 «Directions»

|  |  |  |
| --- | --- | --- |
| Название | Тип данных | Описание |
| ID\_direction | int | хранит идентификатор маршрута |
| Departure\_time | time | Хранит время отправления |
| Departure\_date | date | Хранит дату отправления |
| Direction\_time | time | Хранит время прибытия |
| Direction | date | Хранит маршрут |

Таблица Route состоит из семи столбцов и представлена на таблице 3.5:

Таблица 3.5 «Route»:

|  |  |  |
| --- | --- | --- |
| Название | Тип данных | Описание |
| ID\_route | int | Хранит идентификатор рейса |
| Departure\_place | varchar | Хранит место отправления |
| Destination | varchar | Хранит место прибытия |
| Stops | varchar | Хранит информацию об остановках |
| ID\_driver | int | Хранит идентификатор водителя, внешний ключ на таблицу Drivers |
| ID\_bus | int | Хранит идентификатор автобуса, внешний ключ на таблицу Buses |
| ID\_direction | int | Хранит идентификатор маршрута, внешний ключ на таблицу Directions |

Таблица Tickets состоит из семи столбцов и представлена на таблице 3.6:

3.6 «Tickets»

|  |  |  |
| --- | --- | --- |
| Название | Тип данных | Описание |
| ID\_ticket | int | Хранит идентификатор билета |
| ID\_passenger | int | Хранит идентификатор пассажира, внешний ключ на таблицу Directions |
| ID\_route | int | Хранит идентификатор рейса, внешний ключ на таблицу Route |
| ID\_user | int | Хранит идентификатор пользователя, внешний ключ к таблице Users |
| Purchase\_date | date | Хранит дату покупки билета |
| Seats\_number | int | Хранит номер места |
| Cost | int | Хранит цену билета |

Таблица Users состоит из четырёх столбцов и представлена на таблице 3.7:

Таблица 3.7 «Users»

|  |  |  |
| --- | --- | --- |
| Название | Тип данных | Описание |
| ID\_user | int | Хранит идентификатор пользователя |
| Login | varchar | Хранит логин пользователя |
| Password | varchar | Хранит пароль пользователя |
| ID\_role | int | Хранит идентификатор роли, внешний ключ к таблице Roles |

Таблица Roles состоит из двух столбцов и представлена на таблице 3.8:

Таблица 3.8 «Roles»

|  |  |  |
| --- | --- | --- |
| Название | Тип данных | Описание |
| ID\_role | int | Хранит идентификатор роли |
| Role\_name | varchar | Хранит имя роли |

## Представления

Представление (view) в базе данных представляет собой виртуальную таблицу, которая создается на основе запроса к одной или нескольким таблицам в базе данных. Представления позволяют обращаться к данным из нескольких таблиц одновременно, при этом не изменяя структуру этих таблиц.

В данном проекте были созданы два представления:

RoleUsers, которое объединяет данные таблиц Users и Role;

TicketDetails, которое содержит подробную информацию о билете.

Представление RoleUsers было создано для того, чтобы получить полную информацию о пользователях и их ролях, объединив данные из двух таблиц. Оно будет на листинге 3.1. Остальные представления будут аналогичны, только будут работать с другими таблицами и представлены в приложении Е.

|  |
| --- |
| CREATE VIEW RoleUsers AS  SELECT R.Role\_name,  U.ID\_user,  U.Login,  U.Password  FROM Roles R  JOIN Users U ON R.ID\_role = U.ID\_role; |

Листинг 3.1 – Представление all\_info\_track

Представление TicketDetails было создано для того, чтобы получить подробную информацию о планируемых поездках пассажира, объединив данные из таблиц Passengers, Directions, Drivers, Buses, Users и Tickets.

## Индексы

Индекс — объект базы данных, который используется для ускорения поиска данных. В случае большого количества строк в таблице, последовательный поиск данных может занимать много времени. Индекс формируется на основе значений одного или нескольких столбцов таблицы и указателей на соответствующие строки таблицы, что позволяет быстро искать строки, удовлетворяющие заданному критерию поиска.

Для того, чтобы быстро находить пассажиров по их фамилиям, был создан индекс idx\_SurnamePassenger на таблице Passengers по полю Surname. Индекс представлены на листинге 3.2.

|  |
| --- |
| create INDEX idx\_SurnamePassenger ON Passengers (Surname); |

Листинг 3.2 Индекс idx\_SurnamePassenger

В целом, использование индексов позволяет существенно ускорить операции поиска, сортировки и фильтрации данных в базе данных, особенно в случае большого объема данных. Однако создание индексов может занять дополнительное время при добавлении или изменении данных в таблицах, поэтому необходимо сбалансировать количество и тип индексов для оптимальной производительности базы данных.

## Тригеры

Триггер базы данных — это объект базы данных, который выполняет некоторое действие автоматически при определенных событиях в таблице или представлении базы данных. Триггер может быть запрограммирован на срабатывание при вставке, обновлении или удалении строк в таблице.

Триггеры используются для обеспечения целостности данных и контроля доступа к данным, а также для автоматической обработки данных при выполнении определенных операций в таблице. Создание триггеров представлено в приложении Б.

Первый триггер, созданный в таблице Users, проверяет, что возраст нового пользователя не является недопустимым. Если дата рождения нового пользователя меньше 16 или больше 110 лет, то триггер генерирует исключение с сообщением об ошибке "Возраст должен быть от 16 до 110 лет". Триггер запускается до вставления новой строки и проверяет каждую новую введённую строку на корректность возраста. Скрипт триггера будет представлен на листинге 3.3.

|  |
| --- |
| CREATE TRIGGER CheckUserAge  ON Passengers  AFTER INSERT  AS  BEGIN  IF (SELECT COUNT(\*) FROM inserted WHERE Age < 16 OR Age > 110) > 0  BEGIN  RAISERROR('Ошибка: Возраст пользователя должен быть от 16 до 110 лет', 16, 1)  ROLLBACK TRANSACTION  END  END |
|  |

Листинг 3.3 – Скрип триггера c CheckUserAge

Второй триггер проверяет, что длина пароля нового пользователя не менее 6 символов. Если длина пароля меньше 6 символов, то триггер генерирует исключение с сообщением об ошибке "Password must be at least 6 characters". Триггер создается аналогично предыдущему примеру. Скрипт триггера будет представлен на листинге 3.4.

|  |
| --- |
| CREATE TRIGGER CheckPasswordLength  ON Users  AFTER INSERT  AS  BEGIN  IF (SELECT MIN(LEN(Password)) FROM inserted) < 6  BEGIN  RAISERROR('Ошибка: Пароль должен содержать не менее 6 символов', 16, 1)  ROLLBACK TRANSACTION  END  END |

Листинг 3.3 – Скрип триггера c CheckPasswordLength

## Функции

Для управления данными через приложение все пользователи использует функции. Функция – объект базы данных, представляющий собой набор SQL-инструкций, который компилируется один раз и хранится на сервере. Стоит отметить, что функции не могут содержать группы операторов DML и DDL. В моём приложении реализовано 15 функций. Пример создания функции представлен на листинге 3.5.

|  |
| --- |
| CREATE or alter FUNCTION totalDateTickets ( @Purchase\_date date) RETURNS int  BEGIN  DECLARE @dateTickets int ;    set @dateTickets = (SELECT count(ID\_ticket)  FROM Tickets  WHERE Purchase\_date = @Purchase\_date)    RETURN @dateTickets;  END |

Листинг 3.5 – Функция поиска всех билетов, купленных в определённую дату

Эта функция totalDateTickets возвращает общее количество билетов, купленных в указанную дату. В качестве входного параметра принимается дата @Purchase\_date. Далее в функции создается переменная @dateTickets типа int, в которую записывается количество билетов, у которых Purchase\_date равна заданной дате. Затем функция возвращает значение @dateTickets..

## Процедуры

По большому счёту, процедуры выполняют аналогичные действия с функциями, за тем исключением, что они не могут возвращать значений (но могут иметь выходные параметры) и имеют возможность в теле использовать DML-операторы. Всего было разработано мною 15 процедур. На листинге 3.6 представлен пример реализации одной из процедур. Создание и вызов всех процедур базы данных представлено в приложении Д.

|  |
| --- |
| CREATE OR ALTER PROCEDURE OrderTransport  @p\_ID\_ticket INT,  @p\_ID\_passenger INT,  @p\_ID\_route INT,  @p\_ID\_user INT  AS  BEGIN  DECLARE @v\_Seats\_number INT;  DECLARE @v\_Cost FLOAT;  SELECT @v\_Seats\_number = b.Seats\_number - ISNULL(COUNT(t.ID\_ticket), 0)  FROM Route r  INNER JOIN Buses b ON r.ID\_bus = b.ID\_bus  LEFT JOIN Tickets t ON r.ID\_route = t.ID\_route  WHERE r.ID\_route = @p\_ID\_route  GROUP BY r.ID\_route, b.Seats\_number;  SELECT @v\_Cost = Cost  FROM Route  WHERE ID\_route = @p\_ID\_route;  INSERT INTO Tickets (ID\_ticket, ID\_passenger, ID\_route, ID\_user, Purchase\_date, Seats\_number, Cost)  VALUES (@p\_ID\_ticket, @p\_ID\_passenger, @p\_ID\_route, @p\_ID\_user, GETUTCDATE(), @v\_Seats\_number, @v\_Cost);  END;  EXEC OrderTransport @p\_ID\_ticket = 8, @p\_ID\_passenger = 2, @p\_ID\_route = 3, @p\_ID\_user = 1; |

Листинг 3.6 Процедура заказа транспорта

## Пользователи

Пользователь базы данных — это физическое или юридическое лицо, которое имеет доступ к БД и пользуется услугами информационной системы для получения информации. На каждом этапе развития БД (проектирование, реализация, эксплуатация, модернизация и развитие, полная реорганизация) с ней связаны разные категории пользователей.

При проектировании базы данных было создано 2 типа пользователей: пользователь и администратор.

Создание пользователей представлено в листинге 3.7.

|  |
| --- |
| CREATE USER admin\_busstation FOR LOGIN admin\_busstation\_log;  CREATE USER user\_busstation FOR LOGIN user\_busstation\_log;  CREATE ROLE administrator\_role;  CREATE ROLE user\_role;  ALTER ROLE administrator\_role ADD MEMBER admin\_busstation;  ALTER ROLE user\_role ADD MEMBER user\_busstation; |

Листинг 3.7 – Создание пользователей

## Вывод

В данном разделе была рассмотрена разработка объектов базы данных для автовокзала. Все рассмотренные объекты вместе создают сложную, но гибкую и эффективную структуру, которая обеспечивает правильное функционирование и управление данными в системе. Благодаря хорошо спроектированным объектам базы данных система способна обеспечивать надежность, своевременное быстродействие и целостность данных.

# Описание процедур импорта и экспорта

При всей своей отлаженности и очевидности, классический способ хранения и представления объектов развитой структуры имеет и вполне определенные недостатки и может вызывать проблемы, с которыми сталкивался любой разработчик, пытавшийся реализовать таким способом достаточно сложную систему. В некоторых ситуациях, решить эти проблемы позволяет хранение объекта в виде JSON и XML.

## Процедура импорта данных из XML-файла

Импортировать данные из файлов Excel в SQL Server или базу данных SQL SSMS можно несколькими способами. Некоторые методы позволяют импортировать данные за один шаг непосредственно из файлов Excel. Для других методов необходимо экспортировать данные Excel в виде текста (CSV-файла), прежде чем их можно будет импортировать. Импорт осуществляется с помощью мастера импорта и экспорта SQL Server. Для этого происходит подключение к экземпляру, выбирается база данных для импорта, затем в пункте “Задачи” необходимо выбрать “Импортировать данные” и ввести нужные данные.

Для импорта данных из файла в формате CSV был разработан скрипт, который представлен в листинге 4.1.

В качестве разделителя в исходном CSV-файле используются запятые, а для разделителя строки – символ переноса на новую строку. Как правило, данные из CSV-файла нужно сохранять во временную таблицу, а затем из нее вставлять в основную, т.к. зачастую необходимо парсить даты, числа с плавающей запятой и другие форматы. Их можно спарсить в строку во временной таблице и привести к нужному формату в основной.

|  |
| --- |
| CREATE TABLE #csv\_temp (  ID\_bus NVARCHAR(100),  Bus\_brand NVARCHAR(100),  Release\_year NVARCHAR(100),  Seats\_number NVARCHAR(100)  );  --  bulk insert #csv\_temp  from 'D:\import\import\_bus.csv'  with (Fieldterminator = ',', RowTerminator = '\n', CODEPAGE = '1251')  --  select \* from #csv\_temp; |

Листинг 4.1 – Импорт данных из CSV

1. **Описание экспорта данных**

Для таблицы клиенты в базе данных доступна процедура экспорта данных в формате JSON. Это может пригодиться, если необходимо переместить данные на другой сервер или создать резервную копию.

Для реализации экспорта данных в JSON, была разработана функция, результатом которой стало создание JSON файла, с записанными строками из определенной таблицы в JSON формат.

Скрипт запроса для экспорта данных представлен в листинге 4.2.

|  |
| --- |
| CREATE OR ALTER PROCEDURE SaveTableDataToJson  @tableName NVARCHAR(128)  AS  BEGIN  DECLARE @fileName NVARCHAR(128);  SET @fileName = REPLACE(@tableName, ' ', '') + '.json';  DECLARE @sql NVARCHAR(1000);  SET @sql = 'bcp "SELECT \* FROM ' + QUOTENAME(@tableName) + ' ' +  'FOR JSON PATH, INCLUDE\_NULL\_VALUES" ' +  'queryout "D:\' + @fileName + '" ' +  '-c -S LAPTOP-F1RJ50JM -d Busstation\_db -T';  EXEC sys.XP\_CMDSHELL @sql;  END;  EXEC SaveTableDataToJson @tableName = 'Tickets';  select \* from Tickets |

Листинг 4.2 – Процедура экспорта SaveTableDataToJson

**4.3 Вывод**

В данном разделе были представлены различные способы импорта данных из файлов Excel в SQL Server или базу данных SQL SSMS. Кроме того, была представлена процедура SaveTableDataToJson, которая позволяет экспортировать данные из определенной таблицы в базе данных в формате JSON. В целом, возможность импорта данных из CVS и экспорта в JSON формат значительно облегчает управление базой данных и позволяет пользователям эффективно перемещать и сохранять данные.

# Тестирование производительности

## Тестирование производительности базы данных

Для тестирования производительности базы данных, нам нужно заполнить таблицы достаточным количеством записей. Для данного случая мы заполнили одну из таблиц 100.000 строками, скрипт заполнения таблиц приложен в приложение Д.

Далее, для улучшения производительности запросов, мы создали несколько индексов. Для таблицы Passengers мы создали индексы на столбцы Surname. Для таблицы Buses мы создали индекс на столбец Seats\_number, а для таблицы Route - на столбец Departure\_place. Для таблицы Tickets мы создали индекс на столбец Cost. Пример для создания индексов показан в листинге 5.1.

Теперь давайте рассмотрим производительность базы данных в контексте запросов на выборку данных.

|  |
| --- |
| create INDEX idx\_SurnamePassenger ON Passengers(Surname);  create index idx\_Seats\_number\_Bus on Buses(Seats\_number);  create index idx\_Departure\_place on Route(Departure\_place);  create index idx\_CostTickets on Tickets(Cost); |

Листинг 5.1 – Индексы для таблицы

Результаты выполнения запроса к таблице указывают на значительные затраты времени и ресурсов, особенно при сканировании всей таблицы и применении фильтра. Время выполнения запроса составило 19с.

Для получения выборки данных использовался запрос, который представлен на листинге 5.2.

|  |
| --- |
| SET SHOWPLAN\_ALL ON;  GO  SELECT Surname  FROM Passengers  WHERE Surname = 'Surname9969';  GO  SET SHOWPLAN\_ALL OFF;  GO |

Листинг 5.2 – Запрос к таблице Passengers



Рисунок 5.1 – Затраты по времени на выполнение запроса

Результаты выполнения запроса к таблице указывают на значительные затраты времени и ресурсов, особенно при сканировании всей таблицы и применении фильтра.

Для ускорения данного процесса создадим индекс на поле ID\_passenger так как именно по этому полю выполняется фильтрация. После создания индекса, можно повторить запрос и он выполнится быстрее.



Рисунок 5.2 – Затраты по времени на выполнение запроса после создания индекса

Заключение

В результате выполнения данного курсового проекта были реализованы ключевые элементы базы данных, включая таблицы для хранения информации. Индексы были применены для оптимизации производительности запросов, а процедуры обеспечивают управление данными, соблюдая принципы безопасности.

Также были созданы пользователи базы данных с различными уровнями доступа – пользователь и администратор, что обеспечивает отделение прав доступа и повышает безопасность системы.

Разработанные процедуры обеспечивают базовые операции управления данными, а также функционал поиска, подсчёта и фильтрации, что сделает взаимодействие с системой интуитивно понятным и удобным для пользователей.

Благодаря внедрению индексов, процедур, и оптимизированным структурам таблиц, система способна эффективно обрабатывать запросы, в том числе на больших объемах данных, что существенно повышает производительность системы.

В целом, разработанная база данных представляет собой мощный инструмент, сочетающий в себе гибкость, производительность и безопасность.

# Список используемых источников

1. Атлас [Электронный ресурс] / Режим доступа: https://atlasbus.by/– Дата доступа: 18.04.2024
2. Автовокзал Центральный [Электронный ресурс] / Режим доступа: https://minsktrans.by/avtovokzal-czentralnyj/ – Дата доступа: 18.04.2024.
3. MySQL Документация [Электронный ресурс] / Режим доступа: https://www.mysql.com/ – Дата доступа: 20.04.2024.
4. metanit.com [Электронный ресурс] / Режим доступа: https://metanit.com/sql/mysql/ – Дата доступа: 22.04.2024.
5. Триггеры в MySQL [Электронный ресурс] / Режим доступа: https://habr.com/ru/articles/37693/ – Дата доступа: 02.05.2024.
6. Автовокзал «Центральный» [Электронный ресурс] ­­­– Режим доступа: <https://minsktrans.by/avtovokzal-czentralnyj/>. – Дата доступа: 29.02.2024
7. Автовокзал «Центральный» [Электронный ресурс] ­­­– Режим доступа: <https://atlasbus.by/>. – Дата доступа: 29.02.2024

# Приложение А Листинг создания таблиц

|  |
| --- |
| create table Buses(  ID\_bus int primary key,  Bus\_brand varchar(20),  Release\_year date,  Seats\_number int  );  create table Drivers(  ID\_driver int primary key,  Surname varchar(20),  Name varchar(10),  Patronymic varchar(20),  License\_number int,  Telephone\_number varchar(20),  Age int  );  create table Passengers(  ID\_passenger int primary key,  Surname varchar(20),  Name varchar(10),  Telephone\_number varchar(20),  Age int  );  create table Directions(  ID\_direction int primary key,  Departure\_time time,  Departure\_date date,  Direction\_time time,  Direction date  );  create table Route(  ID\_route int primary key,  Departure\_place varchar(50),  Destination varchar(50),  Stops varchar(20),  Cost float,  ID\_driver int foreign key references Drivers(ID\_driver),  ID\_bus int foreign key references Buses(ID\_bus),  ID\_direction int foreign key references Directions(ID\_direction)  );  alter table Route drop constraint FK\_\_Route\_\_ID\_bus\_\_403A8C7D;  alter table Route add constraint FK\_\_Route\_\_ID\_bus\_\_403A8C7D foreign key(ID\_bus) references Buses(ID\_bus) on delete cascade;  create table Roles(  ID\_role int primary key,  Role\_name varchar(10)  );  create table Users(  ID\_user int primary key,  Login varchar(20),  Password varchar(20),  ID\_role int foreign key references Roles(ID\_role),  );  create table Tickets(  ID\_ticket int primary key,  ID\_passenger int foreign key references Passengers(ID\_passenger),  ID\_route int foreign key references Route(ID\_Route),  ID\_user int foreign key references Users(ID\_user),  Purchase\_date date,  Seats\_number int,  Cost float  );  alter table Tickets drop constraint FK\_\_Tickets\_\_ID\_rout\_\_49C3F6B7;  alter table Tickets add constraint FK\_\_Tickets\_\_ID\_rout\_\_49C3F6B7 foreign key(ID\_route) references Route(ID\_route) on delete cascade; |

# 

# Приложение Б Листинг создания триггеров

|  |
| --- |
| drop trigger CheckPasswordLength;  drop trigger CheckUserAge;  CREATE TRIGGER CheckPasswordLength  ON Users  AFTER INSERT  AS  BEGIN  IF (SELECT MIN(LEN(Password)) FROM inserted) < 6  BEGIN  RAISERROR('Ошибка: Пароль должен содержать не менее 6 символов', 16, 1)  ROLLBACK TRANSACTION  END  END;  CREATE TRIGGER CheckUserAge  ON Passengers  AFTER INSERT  AS  BEGIN  IF (SELECT COUNT(\*) FROM inserted WHERE Age < 16 OR Age > 110) > 0  BEGIN  RAISERROR('Ошибка: Возраст пользователя должен быть от 16 до 110 лет', 16, 1)  ROLLBACK TRANSACTION  END  END;  CREATE TRIGGER UpdateOrdersByUser  ON Tickets  AFTER INSERT  AS  BEGIN  SET NOCOUNT ON;  DECLARE @UserID int;  SELECT @UserID = ID\_user  FROM inserted;  UPDATE OrdersByUser  SET ID\_ticket = t.ID\_ticket,  ID\_passenger = t.ID\_passenger,  ID\_route = t.ID\_route,  ID\_user = t.ID\_user,  Purchase\_date = t.Purchase\_date,  Seats\_number = t.Seats\_number,  Cost = t.Cost  FROM OrdersByUser ou  JOIN Tickets t ON ou.ID\_ticket = t.ID\_ticket  WHERE t.ID\_user = @UserID;  END |

# Приложение В Листинг процедуры для заполнения таблицы

|  |
| --- |
| DECLARE @counter INT, @counter1 varchar(10)  SET @counter = 4  WHILE @counter <= 100000  BEGIN  SET @counter1 = CAST(@counter AS varchar(10)) -- Initialize @counter1 with the value of @counter    INSERT INTO Passengers(ID\_passenger, Surname, Name, Telephone\_number, Age)  VALUES (@counter,  CONCAT('Surname', @counter1),  CONCAT('Name', @counter1),  CONCAT('375-33', @counter),  ABS(CHECKSUM(NEWID())) % 80 + 18)  SET @counter = @counter + 1  END |

# Приложение Г Листинг процедуры извлечение информации из файла

|  |
| --- |
| EXEC sp\_configure 'show advanced options', 1;  RECONFIGURE;  EXEC sp\_configure 'xp\_cmdshell', 1;  RECONFIGURE;  ---  SELECT \* FROM Tickets FOR JSON PATH, INCLUDE\_NULL\_VALUES  DECLARE @sql varchar(1000)  SET @sql = 'bcp "SELECT \* FROM Tickets ' +  'FOR JSON PATH, INCLUDE\_NULL\_VALUES" ' +  'queryout "D:\import\tickets.json" ' +  '-c -S LAPTOP-F1RJ50JM -d Busstation\_db -T'  EXEC sys.XP\_CMDSHELL @sql  GO  select @@SERVERNAME;  CREATE OR ALTER PROCEDURE SaveTableDataToJson  @tableName NVARCHAR(128)  AS  BEGIN  DECLARE @fileName NVARCHAR(128);  SET @fileName = REPLACE(@tableName, ' ', '') + '.json';  DECLARE @sql NVARCHAR(1000);  SET @sql = 'bcp "SELECT \* FROM ' + QUOTENAME(@tableName) + ' ' +  'FOR JSON PATH, INCLUDE\_NULL\_VALUES" ' +  'queryout "D:\' + @fileName + '" ' +  '-c -S LAPTOP-F1RJ50JM -d Busstation\_db -T';  EXEC sys.XP\_CMDSHELL @sql;  END;  EXEC SaveTableDataToJson @tableName = 'Tickets'; |

# Приложение Д Листинг создания процедур

|  |
| --- |
| use Busstation\_db;  drop procedure OrderTransport;  drop procedure CalculateFreeSeats;  drop procedure sp\_AddBus;  drop procedure sp\_AddDriver;  drop procedure sp\_AddPassenger;  drop procedure sp\_AddDirection;  drop procedure dbo.AddUser;  drop procedure dbo.AddRoute;  drop procedure dbo.DeleteFromBuses;  drop procedure dbo.DeleteFromDrivers;  drop procedure dbo.DeleteFromPassengers;  drop procedure dbo.DeleteFromDirections;  drop procedure dbo.DeleteFromRoute;  drop procedure dbo.DeleteFromTickets;  drop procedure SaveTableDataToJson;  CREATE OR ALTER PROCEDURE OrderTransport  @p\_ID\_ticket INT,  @p\_ID\_passenger INT,  @p\_ID\_route INT,  @p\_ID\_user INT  AS  BEGIN  DECLARE @v\_Seats\_number INT;  DECLARE @v\_Cost FLOAT;  SELECT @v\_Seats\_number = b.Seats\_number - ISNULL(COUNT(t.ID\_ticket), 0)  FROM Route r  INNER JOIN Buses b ON r.ID\_bus = b.ID\_bus  LEFT JOIN Tickets t ON r.ID\_route = t.ID\_route  WHERE r.ID\_route = @p\_ID\_route  GROUP BY r.ID\_route, b.Seats\_number;  SELECT @v\_Cost = Cost  FROM Route  WHERE ID\_route = @p\_ID\_route;  INSERT INTO Tickets (ID\_ticket, ID\_passenger, ID\_route, ID\_user, Purchase\_date, Seats\_number, Cost)  VALUES (@p\_ID\_ticket, @p\_ID\_passenger, @p\_ID\_route, @p\_ID\_user, GETUTCDATE(), @v\_Seats\_number, @v\_Cost);  END;  EXEC OrderTransport @p\_ID\_ticket = 8, @p\_ID\_passenger = 2, @p\_ID\_route = 3, @p\_ID\_user = 1;  select \* from Tickets;  CREATE OR ALTER PROCEDURE CalculateFreeSeats  AS  BEGIN  SELECT r.ID\_route, b.Seats\_number - ISNULL(COUNT(t.ID\_ticket), 0) AS Free\_seats  FROM Route r  JOIN Buses b ON r.ID\_bus = b.ID\_bus  LEFT JOIN Tickets t ON r.ID\_route = t.ID\_route  GROUP BY r.ID\_route, b.Seats\_number;  END;  exec CalculateFreeSeats ;  CREATE PROCEDURE sp\_AddBus  @BusID int,  @Brand varchar(20),  @Year date,  @Seats int  AS  BEGIN  INSERT INTO Buses (ID\_bus, Bus\_brand, Release\_year, Seats\_number)  VALUES (@BusID, @Brand, @Year, @Seats)  END  -- Пример вызова процедуры  EXEC sp\_AddBus 3, 'BMW', '2021-01-01', 39;  select \* from buses;  CREATE PROCEDURE sp\_AddDriver  @DriverID int,  @Surname varchar(20),  @Name varchar(20),  @Patronomic varchar(20),  @License\_number int,  @Telephone\_number varchar(20),  @Age int  AS  BEGIN  INSERT INTO Drivers (ID\_driver, Surname, Name, Patronymic, License\_number, Telephone\_number, Age)  VALUES (@DriverID, @Surname, @Name, @Patronomic, @License\_number, @Telephone\_number, @Age)  END  EXEC sp\_AddDriver 5, 'Teplitsa', 'Aleksei', 'dmitrievich', 23456, '+375-33-684-32-25', 50;  select \* from Drivers;  CREATE PROCEDURE sp\_AddPassenger  @PassengerID int,  @Surname varchar(20),  @Name varchar(20),  @Telephone\_number varchar(20),  @Age int  AS  BEGIN  INSERT INTO Passengers(ID\_passenger, Surname, Name, Telephone\_number, Age)  VALUES (@PassengerID, @Surname, @Name, @Telephone\_number, @Age)  END  EXEC sp\_AddPassenger 100001, 'Teplitsa', 'Anna', '+375-33-313-75-83', 25;  select \* from Passengers where ID\_passenger=100001;  CREATE PROCEDURE sp\_AddDirection  @DirectionID int,  @Departure\_time time,  @Departure\_date date,  @Direction\_time time,  @Direction date  AS  BEGIN  INSERT INTO Directions(ID\_direction, Departure\_time, Departure\_date, Direction\_time, Direction)  VALUES (@DirectionID, @Departure\_time, @Departure\_date, @Direction\_time, @Direction)  END  EXEC sp\_AddDirection 5, '08:00:00', '2024-10-10', '13:25:00', '2024-10-10';  select \* from Directions;  CREATE OR ALTER PROCEDURE dbo.AddUser  @ID\_user int,  @Login nvarchar(16),  @Password nvarchar(7),  @ID\_role int  AS  BEGIN  IF NOT EXISTS (SELECT 1 FROM Roles WHERE ID\_role = @ID\_role)  BEGIN  RAISERROR('Данной роли не существует', 16, 1)  RETURN  END  BEGIN TRY  INSERT INTO Users (ID\_user, Login, Password, ID\_role)  VALUES (@ID\_user, @Login, @Password, @ID\_role)  END TRY  BEGIN CATCH  IF ERROR\_NUMBER() = 547  BEGIN  RAISERROR('Нарушение ограничения внешнего ключа. Проверьте правильность идентификатора клиента.', 16, 1)  END  ELSE  BEGIN  DECLARE @ErrorMessage NVARCHAR(4000) = ERROR\_MESSAGE()  DECLARE @ErrorSeverity INT = ERROR\_SEVERITY()  DECLARE @ErrorState INT = ERROR\_STATE()  RAISERROR(@ErrorMessage, @ErrorSeverity, @ErrorState)  END  END CATCH  END;  EXEC dbo.AddUser  @ID\_user = 5,  @Login = 'ex',  @Password = 'abc666',  @ID\_role=1;  select \* from Users;  CREATE OR ALTER PROCEDURE dbo.AddRoute  @ID\_route int,  @Departure\_place varchar(50),  @Destination varchar(50),  @Stops varchar(20),  @Cost float,  @ID\_driver int,  @ID\_bus int,  @ID\_direction int  AS  BEGIN  IF NOT EXISTS (SELECT 1 FROM Directions WHERE ID\_direction = @ID\_direction)  BEGIN  RAISERROR('А нету такого маршрута', 16, 1)  RETURN  END  IF NOT EXISTS (SELECT 1 FROM Buses WHERE ID\_bus = @ID\_bus)  BEGIN  RAISERROR('Такого автобусика не существует', 16, 1)  RETURN  END  IF NOT EXISTS (SELECT 1 FROM Drivers WHERE ID\_driver = @ID\_driver)  BEGIN  RAISERROR('Водилы нет', 16, 1)  RETURN  END  BEGIN TRY  INSERT INTO Route (ID\_route, Departure\_place, Destination, Stops, Cost, ID\_driver, ID\_bus, ID\_direction)  VALUES (@ID\_route, @Departure\_place, @Destination, @Stops, @Cost, @ID\_driver, @ID\_bus, @ID\_direction)  END TRY  BEGIN CATCH  IF ERROR\_NUMBER() = 547  BEGIN  RAISERROR('нарушение первичного ключа', 16, 1)  END  ELSE  BEGIN  DECLARE @ErrorMessage NVARCHAR(4000) = ERROR\_MESSAGE()  DECLARE @ErrorSeverity INT = ERROR\_SEVERITY()  DECLARE @ErrorState INT = ERROR\_STATE()  RAISERROR(@ErrorMessage, @ErrorSeverity, @ErrorState)  END  END CATCH  END  exec dbo.AddRoute  @ID\_route = 1,  @Departure\_place = 'rrr',  @Destination = 'rrrr',  @Stops = 'eee, ffff',  @Cost = 45.5,  @ID\_driver = 1,  @ID\_bus = 1,  @ID\_direction =2;  drop procedure dbo.AddRoute;  CREATE OR ALTER PROCEDURE dbo.DeleteFromBuses  @ID\_busToDelete int  AS  BEGIN  IF NOT EXISTS (SELECT 1 FROM Buses WHERE ID\_bus = @ID\_busToDelete)  BEGIN  RAISERROR ('автобусика нет', 16, 1)  RETURN  END  BEGIN TRY  DELETE FROM Buses  WHERE ID\_bus = @ID\_busToDelete  END TRY  BEGIN CATCH  DECLARE @ErrorMessage NVARCHAR(4000) = ERROR\_MESSAGE()  DECLARE @ErrorSeverity INT = ERROR\_SEVERITY()  DECLARE @ErrorState INT = ERROR\_STATE()  RAISERROR (@ErrorMessage, @ErrorSeverity, @ErrorState)  END CATCH  END  EXEC dbo.DeleteFromBuses @ID\_busToDelete = 2;  select \* from Buses;  select \* from Route;  CREATE OR ALTER PROCEDURE dbo.DeleteFromDrivers  @ID\_driverToDelete int  AS  BEGIN  IF NOT EXISTS (SELECT 1 FROM Drivers WHERE ID\_driver = @ID\_driverToDelete)  BEGIN  RAISERROR ('водилы нет', 16, 1)  RETURN  END  BEGIN TRY  DELETE FROM Drivers  WHERE ID\_driver = @ID\_driverToDelete  END TRY  BEGIN CATCH  DECLARE @ErrorMessage NVARCHAR(4000) = ERROR\_MESSAGE()  DECLARE @ErrorSeverity INT = ERROR\_SEVERITY()  DECLARE @ErrorState INT = ERROR\_STATE()  RAISERROR (@ErrorMessage, @ErrorSeverity, @ErrorState)  END CATCH  END  EXEC dbo.DeleteFromDrivers @ID\_driverToDelete = 2;  select \* from Drivers;  CREATE OR ALTER PROCEDURE dbo.DeleteFromPassengers  @ID\_passengerToDelete int  AS  BEGIN  IF NOT EXISTS (SELECT 1 FROM Passengers WHERE ID\_passenger = @ID\_passengerToDelete)  BEGIN  RAISERROR ('такого пассажира нет', 16, 1)  RETURN  END  BEGIN TRY  DELETE FROM Passengers  WHERE ID\_passenger = @ID\_passengerToDelete  END TRY  BEGIN CATCH  DECLARE @ErrorMessage NVARCHAR(4000) = ERROR\_MESSAGE()  DECLARE @ErrorSeverity INT = ERROR\_SEVERITY()  DECLARE @ErrorState INT = ERROR\_STATE()  RAISERROR (@ErrorMessage, @ErrorSeverity, @ErrorState)  END CATCH  END  EXEC dbo.DeleteFromPassengers @ID\_passengerToDelete = 44586;  drop procedure dbo.DeleteFromPassengers;  select \* from Passengers where ID\_passenger=44586;  CREATE OR ALTER PROCEDURE dbo.DeleteFromDirections  @ID\_directionToDelete int  AS  BEGIN  IF NOT EXISTS (SELECT 1 FROM Directions WHERE ID\_direction = @ID\_directionToDelete)  BEGIN  RAISERROR ('маршрута нет', 16, 1)  RETURN  END  BEGIN TRY  DELETE FROM Directions  WHERE ID\_direction = @ID\_directionToDelete  END TRY  BEGIN CATCH  DECLARE @ErrorMessage NVARCHAR(4000) = ERROR\_MESSAGE()  DECLARE @ErrorSeverity INT = ERROR\_SEVERITY()  DECLARE @ErrorState INT = ERROR\_STATE()  RAISERROR (@ErrorMessage, @ErrorSeverity, @ErrorState)  END CATCH  END  EXEC dbo.DeleteFromDirections @ID\_directionToDelete = 2;  drop procedure dbo.DeleteFromDirections;  select \* from Directions;  CREATE OR ALTER PROCEDURE dbo.DeleteFromRoute  @ID\_routeToDelete int  AS  BEGIN  IF NOT EXISTS (SELECT 1 FROM Route WHERE ID\_route = @ID\_routeToDelete)  BEGIN  RAISERROR ('водилы нет', 16, 1)  RETURN  END  BEGIN TRY  DELETE FROM Route  WHERE ID\_route = @ID\_routeToDelete  END TRY  BEGIN CATCH  DECLARE @ErrorMessage NVARCHAR(4000) = ERROR\_MESSAGE()  DECLARE @ErrorSeverity INT = ERROR\_SEVERITY()  DECLARE @ErrorState INT = ERROR\_STATE()  RAISERROR (@ErrorMessage, @ErrorSeverity, @ErrorState)  END CATCH  END  select \* from Route;  EXEC dbo.DeleteFromRoute @ID\_routeToDelete = 1;  ----отмена заказанного транспорта  CREATE OR ALTER PROCEDURE dbo.DeleteFromTickets  @ID\_ticketToDelete int  AS  BEGIN  IF NOT EXISTS (SELECT 1 FROM Tickets WHERE ID\_ticket = @ID\_ticketToDelete)  BEGIN  RAISERROR ('билета нет', 16, 1)  RETURN  END  BEGIN TRY  DELETE FROM Tickets  WHERE ID\_ticket = @ID\_ticketToDelete  END TRY  BEGIN CATCH  DECLARE @ErrorMessage NVARCHAR(4000) = ERROR\_MESSAGE()  DECLARE @ErrorSeverity INT = ERROR\_SEVERITY()  DECLARE @ErrorState INT = ERROR\_STATE()  RAISERROR (@ErrorMessage, @ErrorSeverity, @ErrorState)  END CATCH  END  select \* from Tickets;  EXEC dbo.DeleteFromTickets @ID\_ticketToDelete = 1; |

# Приложение Е Листинг создания представлений

|  |
| --- |
| CREATE VIEW RoleUsers AS  SELECT R.Role\_name,  U.ID\_user,  U.Login,  U.Password  FROM Roles R  JOIN Users U ON R.ID\_role = U.ID\_role;  select \* from RoleUsers;  CREATE VIEW TicketDetails AS  SELECT T.ID\_ticket,  P.Surname AS Passenger\_Surname,  P.Name AS Passenger\_Name,  P.Telephone\_number AS Passenger\_Telephone,  R.Departure\_place AS Departure\_place,  R.Destination,  U.Login AS User\_Login,  T.Purchase\_date,  T.Cost  FROM Tickets T  JOIN Passengers P ON T.ID\_passenger = P.ID\_passenger  JOIN Route R ON T.ID\_route = R.ID\_route  JOIN Users U ON T.ID\_user = U.ID\_user;  select \* from TicketDetails;  CREATE VIEW TicketDetails AS  SELECT T.ID\_ticket,  P.Surname AS Passenger\_Surname,  P.Name AS Passenger\_Name,  P.Telephone\_number AS Passenger\_Telephone,  R.Departure\_place AS Departure\_place,  R.Destination,  U.Login AS User\_Login,  T.Purchase\_date,  T.Cost  FROM Tickets T  JOIN Passengers P ON T.ID\_passenger = P.ID\_passenger  JOIN Route R ON T.ID\_route = R.ID\_route  JOIN Users U ON T.ID\_user = U.ID\_user;  select \* from TicketDetails; |