МИНИСТЕРСТВО НАУКИ И ВЫСШЕГО ОБРАЗОВАНИЯ РОССИЙСКОЙ ФЕДЕРАЦИИ

Федеральное государственное бюджетное образовательное учреждение высшего образования

«Забайкальский государственный университет»

(ФГБОУ ВО «ЗабГУ»)

Энергетический факультет

Кафедра прикладной информатики и математики

**КУРСОВАЯ РАБОТА**

по дисциплине: «Базы данных»  
на тему: «Проектирование и разработка базы данных «Троллейбусное управление»»

Семестр 4

Выполнил: ст. гр. ПИ-21

Бурлакова В. Н.

Проверил: старший преподаватель кафедры ПИМ Демченко К.А.

Чита

2023

МИНИСТЕРСТВО НАУКИ И ВЫСШЕГО ОБРАЗОВАНИЯ РОССИЙСКОЙ ФЕДЕРАЦИИ

Федеральное государственное бюджетное образовательное учреждение высшего образования

«Забайкальский государственный университет»

(ФГБОУ ВО «ЗабГУ»)

Энергетический факультет

Кафедра прикладной информатики и математики

ЗАДАНИЕ

на курсовую работу (проект)

Студенту \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

(фамилия, имя, отчество)

Направление подготовки: 09.03.03 Прикладная информатика

Профиль подготовки Прикладная информатика в экономике

1. Тема курсовой работы (проекта)

Проектирование и разработка базы данных «Троллейбусное управление»

2. Срок подачи студентом законченной работы\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

3. Перечень подлежащих разработке в курсовой работе вопросов:

1. Анализ предметной области: троллейбусное управление
2. Проектирование инфологической и даталогической модели базы троллейбусного управления
3. Проектирование базы данных в среде MS SQL для троллейбусного управления

4. Перечень графического материала:

* Инфологическая модель базы данных;
* Диаграмма базы данных.

Дата выдачи задания \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

Руководитель курсовой работы (проекта)\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

(подпись, расшифровка подписи)

Задание принял к исполнению

«\_\_» \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_ 20\_\_г.

Подпись студента \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_ / \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_/

(ФИ.О.)

**Оглавление**

[1. Введение 4](#_Toc135775081)

[2. Проектирование базы данных 5](#_Toc135775082)

[2.1. Краткая характеристика предметной области. 5](#_Toc135775083)

[2.2. Инфологическая модель базы данных 6](#_Toc135775084)

[2.3. Даталогическая модель базы данных 8](#_Toc135775085)

[2.4. Стратегия резервного копирования и восстановления 14](#_Toc135775086)

[3. Реализация базы данных 19](#_Toc135775087)

[3.1. Характеристика СУБД и других программных средств 19](#_Toc135775088)

[3.2. Создание структуры базы данных 20](#_Toc135775089)

[3.2.1. БД и таблицы 20](#_Toc135775090)

[3.2.2. Индексы 26](#_Toc135775091)

[3.3. Создание представлений 27](#_Toc135775092)

[3.3.1. Однотабличный запрос на выборку по условиям 28](#_Toc135775093)

[3.3.2. Многотабличный запрос с использованием соединения 29](#_Toc135775094)

[3.3.3. Запрос с использованием подзапроса 30](#_Toc135775095)

[3.3.4. Итоговый запрос 30](#_Toc135775096)

[3.3.5. Итоговый запрос с применением оконных функций 31](#_Toc135775097)

[3.4. Примеры запросов на модификацию данных 33](#_Toc135775098)

[3.4.1. Простые запросы 33](#_Toc135775099)

[3.4.2. Запросы с использованием подзапросов 35](#_Toc135775100)

[3.5. Хранимые процедуры и функции, определяемые пользователями 36](#_Toc135775101)

[3.5.1. Создание хранимых процедур с выходными параметрами 36](#_Toc135775102)

[3.5.2. Создание скалярной функции 38](#_Toc135775103)

[3.5.3. Создание функций с типом данных TABLE 39](#_Toc135775104)

[3.6. Описание триггеров 40](#_Toc135775105)

[3.7. Создание пользователей и назначение привилегий 42](#_Toc135775106)

[Заключение 47](#_Toc135775107)

[Список использованной литературы 48](#_Toc135775108)

[Приложение 1 49](#_Toc135775109)

[Приложение 2 50](#_Toc135775110)

# **1. Введение**

Наиболее популярным видом передвижения людей по городу является наземный городской транспорт. Троллейбусное управление занимается организацией перевозок внутри города и в пригороде. В ведении предприятия содержится автотранспорт. Ведется учет числа перевозимых пассажиров, затрачиваемого бензина, времени транспорта на рейсе, на основании чего производится перераспределение транспорта между маршрутами. Учитывается также пробег, число ремонтов и затраты на них по каждому автобусу и троллейбусу в отдельности и по предприятию в целом. Это необходимо для минимизации износа автотранспорта.

# **2. Проектирование базы данных**

## 2.1. Краткая характеристика предметной области.

Наиболее популярным видом передвижения людей по городу является наземный городской транспорт. Троллейбусное управление занимается организацией перевозок внутри города и в пригороде. В ведении предприятия содержится такой автотранспорт как автобусы и троллейбусы. Ведется учет числа перевозимых пассажиров, затрачиваемого бензина, времени транспорта на рейсе, на основании чего производится перераспределение транспорта между маршрутами. Учитывается также пробег, число ремонтов и затраты на обслуживание по каждому транспорту в отдельности и по предприятию в целом. Это необходимо для увеличения прибыли предприятия и минимизации износа автотранспорта.

Для устройства на работу в троллейбусное управление в зависимости от желаемой должности требуются различные документы:

* паспорт, копия паспорта;
* трудовая книжка;
* водительское удостоверение;
* диплом;
* военный билет;
* СНИЛС, копия СНИЛС;
* справка об отсутствии судимости;
* справка из психо- и нарко- диспансеров;
* реквизиты банковской карты;
* договор.

В зависимости от категории водительских прав водителю присваивается класс 1-3 или указывается 0 при его отсутствии.

Водитель может работать в 1, 2 смену или продленную смену. Он может работать только на троллейбусах или только на автобусах, при этом номера маршрутов могут меняться.

Весь транспорт имеет свой идентификатор – бортовой номер.

Каждому водителю перед отъездом выдается терминал, назначенный накануне. При выезде/заезде в депо водителю необходимо отметиться.

Многие расчеты требуют поднятия бумажной волокиты, что не удобно для предприятия. Также информация может быть утеряна. Вследствие этого невозможно быстро сформировать отчет.

Для решения этой проблемы необходимо спроектировать и разработать базу данных (БД) для контроля времени работы водителей и учета количества пассажиров.

## 2.2. Инфологическая модель базы данных

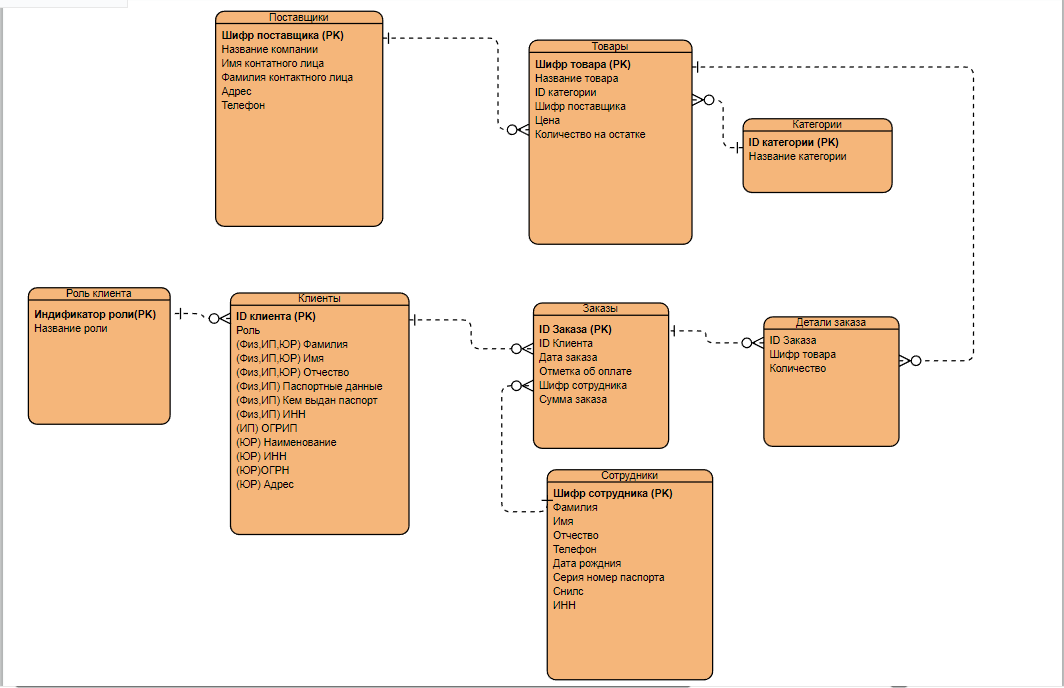
Инфологическая модель данных называется обобщенным неформальным описанием создаваемой базы, выполненным на естественном языке, математическими формулами, таблицами, графиками и другими средствами, понятными для всех людей, работающих над созданием базы.

Она представляет собой человеко-ориентированную моделью, которая полностью независима от физических параметров среды хранения данных.

Инфологическая модель данных представляет собой описание сущностей с набором атрибутов как входных, так и выходных данных. Инфологическая модель включает в себя процесс нормализации.

Нормализация является процессом организации данных в базе данных, в том числе создания таблицы и установления взаимосвязей между ними по правилам, обеспечивающими защиту и делающий базу гибкой, и устраняет избыточную зависимость. Нормализация – это разбиение таблицы на две и более, обладающих лучшими свойствами при включении, изменении и удалении данных.

Нормализация направлена на приведение структуры БД к виду, обеспечивающему минимальную логическую избыточность, и не предполагает уменьшения или увеличения производительности работы, а так же уменьшения или увеличения физического объёма базы. Конечная цель нормализации заключается в уменьшении потенциальной противоречивости, хранящейся в базе данных информации. Общее назначение процесса нормализации заключается в следующем:

* исключение некоторых типов избыточности;
* устранение некоторых аномалий обновления;
* разработка проектов базы данных, которые достаточно «качественно» представляют реальный мир, интуитивное понятие и могут служить хорошим фундаментом для последующих расширений;
* упрощение процедуры применения необходимых ограничений целостности

Необходимо ввести следующие ограничения:

* у сотрудника указывается только тот документ об образовании, который соответствует последней профессиональной подготовке по профилю для конкретной должности;
* за водителем не закрепляется определенный маршрут;
* справка об отсутствии судимости, а также справки из нарко- и психдиспансера предоставляются единожды при трудоустройстве и внесения в БД не требуют;
* категории водительских прав необходимы лишь для установления классности водителя, поэтому внесения в БД не требуют;
* терминал и бортовой номер закрепляются за маршрутом на определенную дату накануне;
* информация о дате получения водительских прав и классности заполняется только для водителей;
* транспортное средство может обслуживаться разными работниками;
* работник может обслуживать несколько транспортных средств;
* поле «дата прибытия» представляется собой дату прибытия транспорта в ведение предприятия;
* количество машин на маршруте статично.

Таким образом, можно выделить следующие сущности:

1. Сущность «Должности» – содержит информацию о должностях.
2. Сущность «Персонал» – содержит личные сведения работника.
3. Сущность «Транспорт» - содержит информацию о транспорте, который содержится в ведении предприятия.
4. Сущность «Виды обслуживания» – содержит информацию о видах обслуживания.
5. Сущность «Обслуживание» - содержит информацию об обслуживании, проводимом на данном транспорте.
6. Сущность «Виды транспорта» – содержит информацию о видах транспорта в ведении предприятия.
7. Сущность «Маршруты» - содержит информацию о маршрутах.
8. Сущность «Выезды» – содержит информацию о фактических выездах водителей на маршрут.

Инфологическая модель данных представлена в [Приложении 1](#_Приложение_1).

## 2.3. Даталогическая модель базы данных

Даталогическим (логическим) проектированием называют проектирование логической структуры базы данных в среде конкретной СУБД. В качестве модели данных выступает реляционная база данных.

После анализа инфологической модели и перехода от нее к реляционным отношениям мы можем спроектировать логическую структуру реляционной базы данных.

Для реляционной базы данных проектирование логической структуры заключается в том, чтобы разбить всю информацию по таблицам, а также определить состав атрибутов для каждой из этих таблиц. От ER-модели перейдем к реляционной модели данных.

В результате получили следующие таблицы, для которых были определены идентификаторы таблиц и полей, атрибуты, ключевые поля, размер одной записи, а также наложены ограничения, если они необходимы:

**Таблица №1**

**Даталогическая модель базы данных для сущности «Должности»**

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **Идентификатор таблицы** | **Атрибут** | **Идентификатор поля** | **Тип поля** | **Ограничения / Шаблон** |
| Должности | Идентификатор должности (первичный ключ) | Идент\_должн | INTEGER | PRIMARY KEY IDENTITY (1,1) |
| Должность | Должность | VARCHAR (10) | NOT NULL |

**Таблица №2**

**Даталогическая модель базы данных для сущности «Персонал»**

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **Идентификатор таблицы** | **Атрибут** | **Идентификатор поля** | **Тип поля** | **Ограничения / Шаблон** |
| Персонал | Идентификатор работника (первичный ключ) | Идент\_работн | INTEGER | IDENTITY (1,1) |
| Фамилия | Фамилия | VARCHAR (20) | NOT NULL |
| Имя | Имя | VARCHAR (15) | NOT NULL |
| Отчество | Отчество | VARCHAR (25) |  |
| Идентификатор должности (внешний ключ) | Идент\_должн | INTEGER | NOT NULL |
| Паспортные данные | Паспорт | CHAR (11) | NOT NULL  UNIQUE |

**Продолжение таблицы №2**

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **Идентификатор таблицы** | **Атрибут** | **Идентификатор поля** | **Тип поля** | **Ограничения / Шаблон** |
| Персонал | Номер телефона | Телефон | CHAR(12) | NOT NULL  UNIQUE |
| Дата рождения | Дата\_рожд | DATE | NOT NULL |
| Город | Город | VARCHAR(25) | NOT NULL |
| Адрес | Адрес | VARCHAR(40) | NOT NULL |
| Номер трудовой книжки | Номер\_труд\_книжки | VARCHAR(14) | NOT NULL  UNIQUE |
| Документ об образовании | Док\_об\_образ | VARCHAR (35) | NOT NULL |
| Номер документа об образовании | Номер\_док\_об\_образ | VARCHAR (20) | NOT NULL |
| Военный билет | Военный\_билет | CHAR (10) | DEFAULT ‘0’ |
| СНИЛС | СНИЛС | CHAR (17) | NOT NULL  UNIQUE |
| Реквизиты банковской карты | Рекв\_банк\_карты | CHAR (20) | NOT NULL  UNIQUE |
| Классность | Класс | SMALLINT | CHECK (Class >= 0 AND Class <= 3) |
| Дата получения водит. прав | Дата\_получ\_вод\_прав | DATE |  |

**Таблица №3**

**Таблица даталогической модели базы данных для сущности «Транспорт»**

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **Идентификатор таблицы** | **Атрибут** | **Идентификатор поля** | **Тип поля** | **Ограничения / Шаблон** |
| Транспорт | Бортовой номер (первичный ключ) | Борт\_номер | CHAR (3) | PRIMARY KEY DEFAULT ‘000’ |

**Продолжение таблицы №3**

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **Идентификатор таблицы** | **Атрибут** | **Идентификатор поля** | **Тип поля** | **Ограничения / Шаблон** |
| Транспорт | Марка транспорта | Марка | VARCHAR(10) | NOT NULL |
| Кол-во посадочных мест | Кол\_во\_посад\_мест | INTEGER | NOT NULL DEFAULT 0 |
| Объем топливного бака | Объем\_топл\_бака | INTEGER | NOT NULL |
| Год постановки на учет предприятия | Год\_постановки | INTEGER | NOT NULL |
| Год выпуска | Год\_выпуска | INTEGER | NOT NULL |

**Таблица №4**

**Таблица даталогической модели базы данных для сущности  
«Виды обслуживания»**

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **Идентификатор таблицы** | **Атрибут** | **Идентификатор поля** | **Тип поля** | **Ограничения / Шаблон** |
| Вид\_обсл | Идентификатор вида обслуживания (первичный ключ) | Идент\_обсл | INTEGER | PRIMARY KEY  IDENTITY (1,1) |
| Вид обслуживания | Вид\_обсл | VARCHAR (8) | NOT NULL |

**Таблица №5**

**Таблица даталогической модели базы данных для сущности «Обслуживание»**

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **Идентификатор таблицы** | **Атрибут** | **Идентификатор поля** | **Тип поля** | **Ограничения / Шаблон** |
| Обслуживание | Номер заявки (первичный ключ) | Номер\_заявки | INTEGER | PRIMARY KEY  IDENTITY (1,1) |
| Идентификатор работника (внешний ключ) | Идент\_работн | INTEGER | FOREIGN KEY  NOT NULL |

**Продолжение таблицы №5**

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **Идентификатор таблицы** | **Атрибут** | **Идентификатор поля** | **Тип поля** | **Ограничения / Шаблон** |
| Обслуживание | Бортовой номер (внешний ключ) | Борт\_номер | CHAR (3) | FOREIGN KEY  NOT NULL |
| Идентификатор обслуживания (внешний ключ) | Идент\_обсл | INTEGER | FOREIGN KEY  NOT NULL |
| Дата | Дата\_обслуж | DATE | NOT NULL |

**Таблица №6**

**Таблица даталогической модели базы данных для сущности «Виды транспорта»**

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **Идентификатор таблицы** | **Атрибут** | **Идентификатор поля** | **Тип поля** | **Ограничения / Шаблон** |
| Виды\_транспорта | Идентификатор вида транспорта | Идент\_вид\_трансп | INTEGER | PRIMARY KEY  IDENTITY (1,1) |
| Вид транспорта | Вид\_трансп | VARCHAR (10) | NOT NULL |

**Таблица №7**

**Таблица даталогической модели базы данных для сущности «Маршрут»**

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **Идентификатор таблицы** | **Атрибут** | **Идентификатор поля** | **Тип поля** | **Ограничения / Шаблон** |
| Маршрут | Номер маршрута (первичный ключ) | Номер\_марш | INTEGER | PRIMARY KEY  NOT NULL |
| Идентификатор вида транспорта | Идент\_вид\_траснп | INTEGER | FOREIGN KEY  NOT NULL |
| Протяженность маршрута | Протяж\_марш | FLOAT | NOT NULL |
| Кол-во машин | Кол\_во\_машин | SMALLINT | NOT NULL CHECK (Number\_cars >= 0) |
| Стоимость проезда | Тариф | MONEY | NOT NULL |

**Продолжение таблицы №7**

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **Идентификатор таблицы** | **Атрибут** | **Идентификатор поля** | **Тип поля** | **Ограничения / Шаблон** |
| Маршрут | Наличие льготного проезда | Наличие\_льгот | BIT | DEFAULT 1 |
| Время выезда из депо | Время\_выезда | TIME | NOT NULL |
| Время заезда в депо | Время\_заезда | TIME | NOT NULL |

**Таблица №8**

**Таблица даталогической модели базы данных для сущности «Выезды»**

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **Идентификатор таблицы** | **Атрибут** | **Идентификатор поля** | **Тип поля** | **Ограничения / Шаблон** |
| Выезды | Идентификатор рейса (первичный ключ) | Идент\_выезда | INTEGER | PRIMARY KEY  IDENTITY (1,1) |
| Идентификатор работника | Идент\_работн | INTEGER | FOREIGN KEY  NOT NULL |
| Номер маршрута | Номер\_марш | INTEGER | FOREIGN KEY  NOT NULL |
| Дата | Дата\_выезда | DATE | NOT NULL |
| Номер смены | Смена | INTEGER | NOT NULL CHECK (Number\_shift >= 0 AND Number\_shift <= 2 |
| Кол-во пассажиров по терминалу | Пасс\_терм | INTEGER | DEFAULT 0 |
| Кол-во пассажиров по терминалу льгота | Пасс\_терм\_льгот | INTEGER | DEFAULT 0 |
| Кол-во пассажиров по билетам | Пасс\_билет | INTEGER | DEFAULT 0 |

**Продолжение таблицы №8**

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **Идентификатор таблицы** | **Атрибут** | **Идентификатор поля** | **Тип поля** | **Ограничения / Шаблон** |
| Выезды | Кол-во пассажиров по билетам льгота | Пасс\_билет\_льгот | INTEGER | DEFAULT 0 |
| Время факт выезда из депо | Время\_выезда | TIME |  |
| Время факт заезда в депо | Время\_заезда | TIME |  |

## 2.4. Стратегия резервного копирования и восстановления

Компонент резервного копирования и восстановления SQL Server обеспечивает необходимую защиту важных данных, которые хранятся в базах данных SQL Server. Чтобы минимизировать риск необратимой потери данных, необходимо регулярно создавать резервные копии баз данных, в которых будут сохраняться производимые изменения данных.

Резервная копия состоит из базы данных и log-файла. Важно понимать, чтобы удачно выполнить резервное копирование данных, нужно, чтобы база данных находилась в неповрежденном состоянии, поэтому перед выполнением резервного копирования базу данных следует проверить на наличие ошибок.

Существуют различные методы резервного копирования базы данных:

1. Полное резервное копирование является основополагающим методом создания резервных копий, при котором выбранный массив данных копируется целиком. Это наиболее полный и надежный вид резервного копирования, хотя и самый затратный. В случае необходимости сохранить несколько копий данных общий хранимый объем будет увеличиваться пропорционально их количеству. Для предотвращения большого объёма использованных ресурсов используют алгоритмы сжатия, а также сочетание этого метода с инкрементным или дифференциальным видами резервного копирования.

Основным достоинством данного метода является простота восстановления с нуля. Так как массив сохранен полностью, также не составляет труда восстановить лишь часть необходимых данных. Из недостатков можно выделить избыточность данного метода. В процессе работы многие файлы могут остаться неизмененными, тем не менее, они также будут включены в резервную копию. Таким образом, потребуется достаточно большой объем носителя. Полное резервное копирование занимает не только излишнее пространство на хранилище, оно также может потребовать больших затрат по времени.

1. Инкрементное резервное копирование в отличие от полного резервного копирования копирует не все данные, а только те, которые были изменены с момента последнего копирования. Понятно, что схема с применением данного вида резервного копирования будет неполноценной, если время от времени не проводить полное резервное копирование. При полном восстановлении системы нужно провести восстановление из последней копии, а потом поочередно восстановить данные из инкрементных копий в порядке их создания. Данный вид используется для того, чтобы в случае создания архивных копий сократить расходуемые объемы на устройствах хранения информации. Также это позволит минимизировать время выполнения заданий резервного копирования, что может быть крайне важно в условиях, когда платформа работает постоянно. У инкрементного копирования есть один нюанс: поэтапное восстановление возвращает и ненужные удаленные файлы за период восстановления. Поэтому при последовательном восстановлении данных из архива имеет смысл резервировать больше дискового пространства, чтобы смогли поместиться в том числе и удаленные файлы.

Достоинством метода можно назвать эффективное использование носителей. Поскольку сохраняются только файлы, измененные с момента последнего полного или инкрементного резервного копирования, резервные копии занимают меньше места.

Недостаток метода заключается в том, что поскольку резервные копии расположены на нескольких носителях, восстановление устройства после аварии может занять больше времени. Кроме того, для эффективного восстановления работоспособности системы носители должны обрабатываться в правильном порядке.

1. Дифференциальное резервное копирование отличается от инкрементного тем, что копируются данные с последнего момента выполнения полного копирования. Данные при этом помещаются в архив «нарастающим итогом». В силу того, что каждая новая копия, созданная таким образом, содержит данные из предыдущей, это более удобно для полного восстановления данных на момент аварии. Для этого нужны только две копии: полная и последняя из дифференциальных, поэтому вернуть утерянные данные можно гораздо быстрее, чем поэтапно восстанавливать все инкременты. К тому же, этот вид копирования избавлен от вышеперечисленных особенностей инкрементного, когда при полном восстановлении старые файлы восстанавливаются без необходимости. При данном методе возникает меньше несоответствий. Но дифференциальное копирование значительно проигрывает инкрементному в экономии требуемого пространства. Так как в каждой новой копии хранятся данные из предыдущих, суммарный объем зарезервированных данных может быть сопоставим с полным копированием. Недостатком метода, как при создании полной архивной копии, является избыточная защита данных. Сохраняются все файлы, измененные с момента последнего инкрементального резервного копирования.

Для проектируемой БД троллейбусного управления будет применяться полное резервное копирование и восстановление.

Создание полной резервной копии базы данных будет происходить с помощью SQL запроса, представляющего собой следующий код:

BACKUP DATABASE АвтоУправление

TO DISK = 'C:\BACKUP\BusManage.bak'

Далее задается простая модель восстановления данных:

ALTER DATABASE АвтоУправление

SET RECOVERY SIMPLE

В случае, когда база данных была утеряна, необходимо восстановить её следующим SQL запросом:

RESTORE DATABASE АвтоУправление

FROM DISK='C:\BACKUP\ BusManage.bak'

Полная резервная копия необходима для восстановления данных, если они будут ошибочно удалены. Таким образом, можно будет не волноваться о полной потери всей базы данных из-за ошибки работников или сбоя в системе.

Создание инкрементной резервной копии будет происходить при помощи следующего SQL запроса.

BACKUP DATABASE АвтоУправление

TO DISK='C:\BACKUP\ BusManage.bak'

WITH DIFFERENTIAL

Инкрементная резервная копия обозначает, что копируются только части базы данных, измененные с момента последнего полного резервного копирования базы данных, для экономии времени от постоянных созданий копий уже имеющихся данных. Из-за того, что база данных троллейбусного управления ежедневно будет заполняться большим объемом новых данных, необходима будет инкрементная резервная копия, чтобы экономить время сотрудников и ресурсы системы, путем копирования только тех данных, которые были добавлены с момента последнего резервного копирования.

В процессе выполнения резервного копирования данных появляется проблема выбора технологии хранения резервных копий и данных. В настоящее время наибольшее распространение получили следующие виды носителей: накопители на магнитных лентах, сетевые технологии, дисковые накопители.

В проектируемой базе данных в качестве носителя будут использоваться сетевое хранилище и жесткий диск. Информация будет храниться на дисковых накопителях, только в удаленном хранилище и связь будет осуществляться посредством сетевых технологий. Основным преимуществом является простота в подключении дополнительных платформ для хранения данных и необязательность их размещения в непосредственной близости от серверов, на которых размещены данные подлежащие копированию. А для того, чтобы была возможность создать резервную копию, при условии, что отсутствует связь с сетевым хранилищем будут использоваться жесткие диски.

Сетевой вид носителя необходим для быстрого пополнения БД информацией о новых сотрудниках и совершенных за день рейсах.

В нашем случае жесткий магнитный диск необходим для создания резервной копии в условиях отсутствия сети или неполадок с сетевым хранилищем. По мере того, как сетевое хранилище будет восстановлено, необходимо будет перенести данные с жесткого магнитного накопителя, а затем очистить его для повторного использования.

Выполнение резервного копирования является необходимой ежедневной задачей. Это связано с тем, что один сбой может повлечь за собой затруднения. Но не все данные необходимо сохранять ежедневно. Например, данные о новых сотрудниках необходимо сохранять лишь в день изменения, что происходит не часто.

# **3. Реализация базы данных**

## 3.1. Характеристика СУБД и других программных средств

Для проектируемой базы данных используется Microsoft SQL Server 2018.

Microsoft SQL Server – система управления реляционными базами данных (СУБД), разработанная корпорацией Microsoft. Основной язык, который используется для запросов – Transact–SQL.

Помимо стандартных для СУБД функций, SQL Server 2018 содержит большой набор интегрированных служб по анализу данных. Доступ к данным, расположенным на SQL Server, могут получить любые приложения, разработанные на .Net и VisualStudio, а также приложения пакета Microsoft Office 2007. SQL Server 2012 обеспечивает высочайшую в своём классе масштабируемость, производительность и безопасность.

SQL (Structured Query Language) – это язык структурированных запросов. Данный язык дает возможность создавать и функционировать в реляционных базах данных, которые представляют собой наборы связанной информации, сохраняемой в таблицах. Из-за того, что для нашей предметной области необходимо использовать реляционное представление данных, мы будем использовать язык SQL, который будет способен реализовать все нужные операции с базой данных.

SQL Server 2018 предоставляет:

* критически важную уверенность при работе с данными и характеризующийся повышенной доступностью, высокой производительностью и улучшенными функциональными возможностями в части безопасности для критически важных рабочих нагрузок;
* передовой анализ с управляемым самостоятельным изучением данных и великолепными возможностями интерактивной визуализации данных;
* облачные технологии в соответствии с нуждами клиентов, которые позволяют создавать и расширять решения как при локальных развертываниях, так и в «облаке».

Достоинства:

* наличие мощных средств разработки;
* возможность интеграции с IDE Microsoft Visual Studio с помощью технологии ADO;
* простота использования с разными языками программирования;
* возможность хранения больших объемов данных;
* клиент-серверный способ доступа к данным.

Недостатки:

* высокая цена профессиональных версий СУБД;
* необходимость написания клиентского приложения вручную;
* сложность в освоении.

## 3.2. Создание структуры базы данных

### 3.2.1. БД и таблицы

Для создания базы данных необходимо описать сценарии создания базы данных и создания всех таблиц базы данных с указанием полей, назначенных для них ограничений, обозначая первичные и внешние ключи.

Создание базы данных производится с помощью команды CREATE:

CREATE DATABASE АвтоУправление

ON

PRIMARY (NAME=BusManage,

FILENAME='D: \bm\BusManage.mdf',

SIZE=5,

MAXSIZE=10,

FILEGROWTH=10%)

LOG ON (NAME= BusManage\_log,

FILENAME='D:\ bm\BusManage.ldf',

SIZE=1,

MAXSIZE=5,

FILEGROWTH=10%)

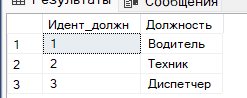
После создания базы данных приступаем к созданию таблиц в базе данных троллейбусного управления.

1. Создание таблицы «Должности»

CREATE TABLE Должности

(Идент\_должн INTEGER IDENTITY(1,1) PRIMARY KEY,

Должность VARCHAR(10) NOT NULL)



**Рис 1 - Отношение «Должности»**

1. Создание таблицы «Персонал»

CREATE TABLE Персонал

(Идент\_работн INTEGER IDENTITY(1,1) PRIMARY KEY,

Фамилия VARCHAR(20) NOT NULL,

Имя VARCHAR(15) NOT NULL,

Отчество VARCHAR(25),

Идент\_должн INTEGER NOT NULL,

FOREIGN KEY (Идент\_должн) REFERENCES Должности (Идент\_должн),

Паспорт CHAR(11) NOT NULL UNIQUE,

Телефон CHAR(12) NOT NULL UNIQUE,

Дата\_рожд DATE NOT NULL,

Город VARCHAR(25) NOT NULL,

Адрес VARCHAR(40) NOT NULL,

Номер\_труд\_книжки VARCHAR(14) NOT NULL UNIQUE,

Док\_об\_образ VARCHAR(35) NOT NULL,

Номер\_док\_об\_образ VARCHAR(20) NOT NULL,

Военный\_билет CHAR(10) DEFAULT '0',

СНИЛС CHAR(17) NOT NULL UNIQUE,

Рекв\_банк\_карты CHAR(20) NOT NULL UNIQUE,

Класс SMALLINT CHECK (Класс >= 0 AND Класс <= 3),

Дата\_получ\_вод\_прав DATE)

****

**Рис 2 - Отношение «Персонал»**

1. Создание таблица «Транспорт»

CREATE TABLE Транспорт

(Борт\_номер CHAR(3) DEFAULT '000' PRIMARY KEY,

Марка VARCHAR(10) NOT NULL,

Кол\_во\_посад\_мест INTEGER NOT NULL DEFAULT 0,

Объем\_топл\_бака INTEGER NOT NULL,

Год\_постановки INTEGER NOT NULL,

Год\_выпуска INTEGER NOT NULL)

****

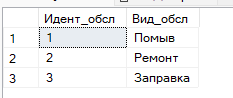
**Рис 3 - Отношение «Транспорт»**

1. Создание таблицы «Вид\_обсл»

CREATE TABLE Вид\_обсл

(Идент\_обсл INTEGER IDENTITY(1,1) PRIMARY KEY,

Вид\_обсл VARCHAR(8))

****

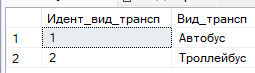
**Рис 4 - Отношение «Вид\_обсл»**

1. Создание таблицы «Виды\_транспорта»

CREATE TABLE Виды\_транспорта

(Идент\_вид\_трансп INTEGER IDENTITY(1,1) PRIMARY KEY,

Вид\_трансп VARCHAR(10) NOT NULL)

****

**Рис 5 - Отношение «Виды\_транспорта»**

1. Создание таблицы «Обслуживание»

CREATE TABLE Обслуживание

(Номер\_заявки INTEGER IDENTITY(1,1),

Идент\_работн INTEGER NOT NULL,

FOREIGN KEY (Идент\_работн) REFERENCES Персонал (Идент\_работн),

Борт\_номер CHAR(3) NOT NULL,

FOREIGN KEY (Борт\_номер) REFERENCES Транспорт (Борт\_номер),

Идент\_обсл INTEGER NOT NULL,

FOREIGN KEY (Идент\_обсл) REFERENCES Вид\_обсл (Идент\_обсл),

Дата\_обслуж DATE NOT NULL)

****

**Рис 6 - Отношение «Обслуживание»**

1. Создание таблицы «Маршрут»

CREATE TABLE Маршрут

(Номер\_марш INTEGER NOT NULL PRIMARY KEY,

Идент\_вид\_трансп INTEGER NOT NULL,

FOREIGN KEY (Идент\_вид\_трансп) REFERENCES Виды\_транспорта (Идент\_вид\_трансп),

Протяж\_марш FLOAT NOT NULL,

Кол\_во\_машин SMALLINT NOT NULL CHECK (Кол\_во\_машин>=0),

Тариф MONEY NOT NULL,

Наличие\_льгот BIT DEFAULT 1,

Время\_выезда TIME NOT NULL,

Время\_заезда TIME NOT NULL)

****

**Рис 7 - Отношение «Маршрут»**

1. Создание таблицы «Выезды»

CREATE TABLE Выезды

(Идент\_выезда INTEGER IDENTITY(1,1) PRIMARY KEY,

Идент\_работн INTEGER NOT NULL,

FOREIGN KEY (Идент\_работн) REFERENCES Персонал (Идент\_работн),

Номер\_марш INTEGER NOT NULL,

FOREIGN KEY (Номер\_марш) REFERENCES Маршрут (Номер\_марш),

Дата\_выезда DATE NOT NULL,

Смена INTEGER NOT NULL CHECK (Смена>=0 AND Смена<=2),

Пасс\_терм INTEGER DEFAULT 0,

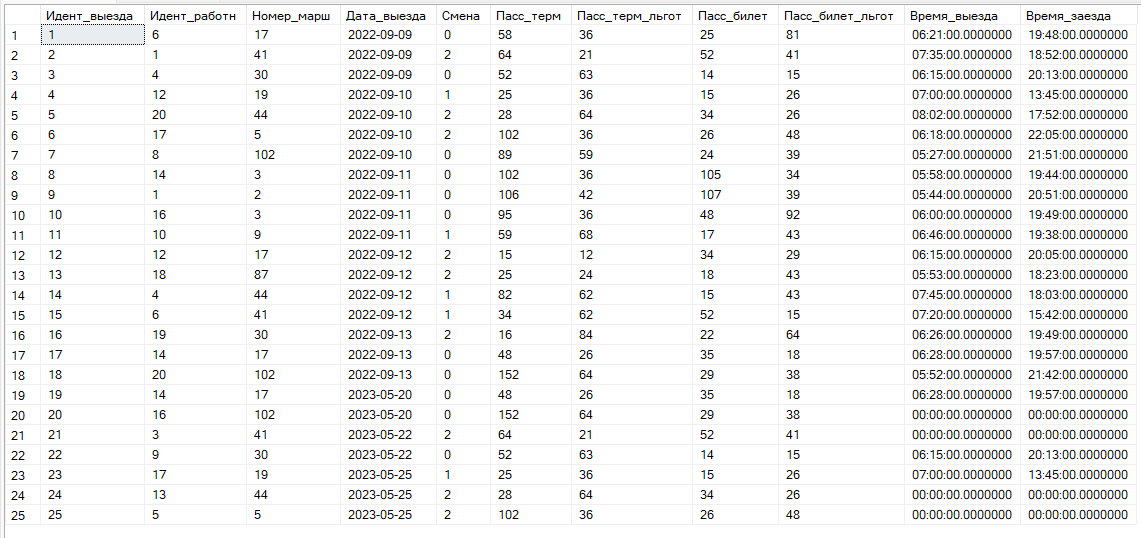
Пасс\_терм\_льгот INTEGER DEFAULT 0,

Пасс\_билет INTEGER DEFAULT 0,

Пасс\_билет\_льгот INTEGER DEFAULT 0,

Время\_выезда TIME,

Время\_заезда TIME)

****

**Рис 8 - Отношение «Выезды»**

После выполнения всех вышеуказанных запросов будут созданы необходимые таблицы, определены поля и ключи. Диаграмма базы данных при этом будет выглядеть согласно [Приложению 2](#_Приложение_2).

### 3.2.2. Индексы

Одним из главных объектов в базе данных является индекс. Согласно определению, индекс представляет собой набор уникальных значений для некоторой таблицы с соответствующими ссылками на данные.

Индекс содержит ключи, построенные из одного или нескольких столбцов в таблице или представлении. Эти ключи хранятся в виде структуры сбалансированного дерева, которая поддерживает быстрый поиск строк по их ключевым значениям в SQL Server.

В Microsoft SQL Server существуют следующие типы индексов:

* Кластеризованные;
* Некластеризованные;
* Уникальные;
* Специальные.

Необходимо избегать создания кластеризованного индекса для часто изменяемых столбцов, поскольку сервер должен будет выполнять физическое перемещение всех данных в таблице, чтобы они находились в упорядоченном состоянии.

В некластеризованном индексе каждая запись значения ключа содержит указатель на строку данных, содержащую значение ключа.

Стоит также учесть, что часть индексов создается автоматически при задании первичного ключа в таблице.

Выбор правильных индексов для базы данных и ее рабочей нагрузки – это решение сложной задачи о соотношении скорости обработки запроса и стоимости обновления. Плохо спроектированные индексы или их недостаточное количество – основной источник узких мест в приложениях баз данных. Проектирование эффективных индексов имеет первостепенную важность для достижения высокой производительности баз данных и приложений.

Для отношения «Персонал» создадим некластеризованный составной индекс по столбцам Фамилия, Имя, Отчество. Это поможет осуществить быстрый поиск информации о сотрудниках.

Запрос на создание индекса будет выглядеть следующим образом:

CREATE NONCLUSTERED INDEX index\_ФИО

ON Персонал (Фамилия, Имя, Отчество)

## 3.3. Создание представлений

Представление – это логическая таблица, представляющая собой поименованный запрос, который будет подставлен как подзапрос при использовании представления. Пока представление не будет проиндексировано, оно не существует в базе данных как хранимая совокупность значений.

Вся информация о представлениях, которые находятся в базе данных, динамически вычисляется на основании данных, находящихся в реальных таблицах. Непосредственное изменение данных в существующей таблице базы данных сразу же отражается в содержимом тех представлений, которые построены на основе этой таблицы.

Использование представлений может скрывать структуру самой базы данных от прикладного программного обеспечения и предоставлять пользователям права только на данные, которые ему предназначены.

### 3.3.1. Однотабличный запрос на выборку по условиям

Однотабличный запрос обращается к определенной таблице в базе данных троллейбусного управления и осуществляет выборку по нужным критериям. Наиболее правильно будет записать результат этого запроса в представление, чтобы пользователи определённых групп могли использовать его для решения определенных задач.

Для базы данных троллейбусного управления был создан однотабличный запрос, который осуществляет поиск сотрудников, которым в этом году исполнится 40 и более лет, имеющих класс 1. Результат запроса будет записан в представление «Классность». Запрос выглядит следующим образом:

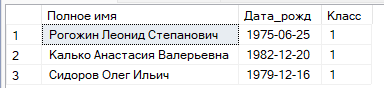
CREATE VIEW Классность

AS

SELECT Фамилия+' '+Имя+' '+Отчество as [Полное имя], Дата\_рожд, Класс FROM Персонал

WHERE Класс = 1 AND YEAR(Дата\_рожд) < YEAR(GETDATE())-40

Результат выполнения запроса представлен на рис. 9.

****

**Рис 9 - Однотабличный запрос на выборку по условию в представлении**

### 3.3.2. Многотабличный запрос с использованием соединения

Многотабличный запрос SQL позволяет оперировать данными из нескольких таблиц одновременно и выбрать нужную информацию по заданным критериям из разных таблиц базы данных.

Для базы данных троллейбусного управления был создан многотабличный запрос. Он осуществляет выборку водителей, которые в текущий день работали на маршруте с тарифом 32 руб.

Выборка из базы данных осуществляется из отношений «Персонал», «Выезды» и «Маршрут». Результат запроса будет записан в представление «Тариф32».

CREATE VIEW Тариф32

AS

SELECT Фамилия+' '+Имя+' '+Отчество as [Полное имя], Выезды.Номер\_марш FROM Персонал, Выезды

INNER JOIN Маршрут

ON Выезды.Номер\_марш = Маршрут.Номер\_марш

WHERE Тариф = 32 AND Персонал.Идент\_работн = Выезды.Идент\_работн

GROUP BY Фамилия+' '+Имя+' '+Отчество, Выезды.Номер\_марш

Результат выполнения данного запроса представлен на рис. 10.

****

**Рис 10 - Многотабличный запрос на выборку по условию в представлении**

### 3.3.3. Запрос с использованием подзапроса

Подзапрос – это SQL запрос, который применяется внутри другого запроса.

Для базы данных троллейбусного управления был создан запрос с подзапросом, который осуществляет поиск водителей, которые не вышли на рейс по распределению в текущий день. Выборка осуществляется из отношений «Выезды» и «Персонал». Результат данного запроса будет храниться в представлении «НеВыходы».

Код данного запроса выглядит следующим образом:

CREATE VIEW НеВыходы

AS

SELECT Фамилия+' '+Имя+' '+Отчество as [Полное имя] FROM Персонал

WHERE (Идент\_работн IN (SELECT Идент\_работн FROM Выезды

WHERE Дата\_выезда = CONVERT(DATE, GETDATE()) AND (Время\_выезда = '')))

Результат выполнения данного запроса представлен на рис. 11.

****

**Рис 11 - Запрос с подзапросом в представлении**

### 3.3.4. Итоговый запрос

Итоговые запросы выполняют выборку информации из одной или нескольких таблиц, выполняя группировку записей по одному из полей, и вычисляя итоговые значения с использованием агрегатных функций по другим полям за определенный период времени. Данные запросы напрямую связаны с анализом данных, так как именно в итоговых запросах зачастую используется агрегирующие и статистические функции СУБД MS SQL Server. В прикладном программном обеспечении итоговые функции могут применяться при создании отчетов.

Для базы данных троллейбусного управления был создан итоговый запрос. Он осуществляет подсчет каждого вида обслуживания, осуществленного для каждого транспортного средства по бортовому номеру, за 2022 год. Данный запрос был записан в представление «Сервис».

CREATE VIEW Сервис

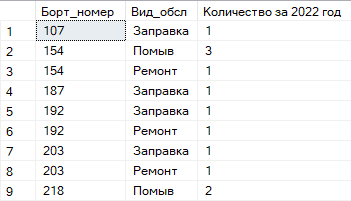
AS

SELECT Борт\_номер, Вид\_обсл, COUNT(Вид\_обсл) FROM Вид\_обсл, Обслуживание

WHERE Обслуживание.Идент\_обсл = Вид\_обсл.Идент\_обсл AND YEAR(Дата\_обслуж) = 2022

GROUP BY Вид\_обсл, Борт\_номер

Результат выполнения данного запроса представлен на рис. 12.

****

**Рис 12 - Результат итогового запроса в представлении**

## 3.3.5. Итоговый запрос с применением оконных функций

Оконная функция – функция, которая работает с выделенным набором строк и выполняет вычисление для него в отдельном столбце.

Аналитические, или оконные, функции принимают в качестве аргумента столбец промежуточного результата вычисления и возвращают столбец.

Окном называют некоторое выражение, описывающее набор строк, которые будет обрабатывать функция, и порядок этой обработки.

Итоговый запрос с применением оконных функций осуществляет выборку информации из одной или нескольких таблиц с помощью предложения OVER.

Предложение OVER определяет секционирование и упорядочение набора строк до применения соответствующей оконной функции.

Для базы данных разработаем запрос с применением оконной функции, вычисляющий количество выездов для каждого водителя в текущем году, количество пассажиров без льгот, с льготами и общее их число.

SELECT DISTINCT(Фамилия+' '+Имя+' '+Отчество) as ФИО,

COUNT(Идент\_выезда) OVER (PARTITION BY Фамилия+' '+Имя+' '+Отчество) [Количество выездов в этом году],

SUM(Пасс\_билет+Пасс\_терм) OVER (PARTITION BY Фамилия+' '+Имя+' '+Отчество) [Пассажиров без льгот],

SUM(Пасс\_билет\_льгот+Пасс\_терм\_льгот) OVER (PARTITION BY Фамилия+' '+Имя+' '+Отчество) [Пассажиров с льготами],

SUM(Пасс\_билет+Пасс\_билет\_льгот+Пасс\_терм+Пасс\_терм\_льгот) OVER (PARTITION BY Фамилия+' '+Имя+' '+Отчество) [Общее число пассажиров]

FROM Выезды В

INNER JOIN Персонал П

ON В.Идент\_работн = П.Идент\_работн

WHERE YEAR(Дата\_выезда) = YEAR(GETDATE())

GROUP BY Фамилия+' '+Имя+' '+Отчество, Идент\_выезда, Пасс\_билет, Пасс\_терм, Пасс\_билет\_льгот, +Пасс\_терм\_льгот

Результат выполнения данного запроса представлен на рис. 13.

****

**Рис 13 - Итоговый запрос с применением оконных функций**

## 3.4. Примеры запросов на модификацию данных

Запросы, связанные с модификацией данных, позволяют пользователем выполнять такие действия как обновление, добавления и удаление данных. Примечательно, что в большинстве баз данных пользователи должны быть наделены определёнными правами на работу с данными. Существует три вида запросов модификации данных:

INSERT INTO – запрос для добавления;

DELETE – запрос удаления;

UPDATE – запрос обновления.

### 3.4.1. Простые запросы

Простые запросы для работы с данными применяются очень часто для выполнения элементарных задач. Практически все виды прикладного программного обеспечения работают с группой CRUD (CREATE, READ, UPDATE, DELETE) задач. Именно эти операции с данными позволяют непременно функционировать с базой данных для работы с данными.

В данном разделе приводятся примеры простых однотабличных запросов, регулярно используемых для выполнения вставки, обновления и удаления данных: INSERT, DELETE и UPDATE.

Операция вставки INSERT – это оператор языка SQL, который позволяет добавлять необходимые строки в таблицу. Сами значения можно вставлять перечислением с помощью VALUES.

Запрос на добавление записи в отношение «Обслуживание» выглядит следующим образом:

INSERT INTO Обслуживание (Идент\_работн, Борт\_номер, Идент\_обсл, Дата\_обслуж)

SELECT 9, 154, 1, '09-09-2022'

UNION ALL

SELECT 9, 154, 1, '11-09-2022'

После данного запроса в отношение были добавлены записи со следующими полями:



Из определения операция обновления UPDATE – оператор языка SQL, который позволяет обновлять строки в таблицах, заполняя их измененными значениями.

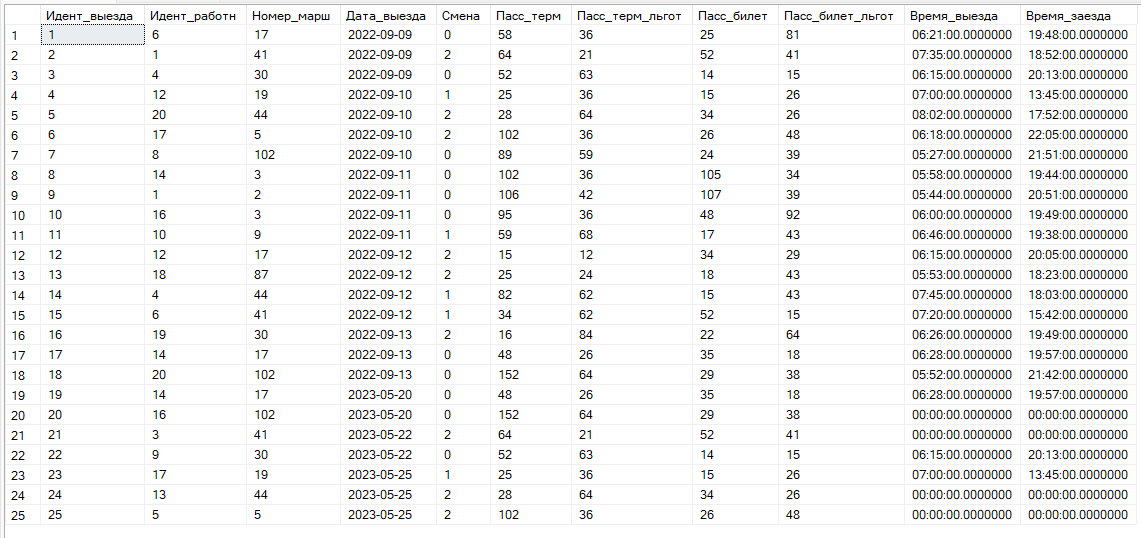
Так как время выезда и заезда записываются позже остальных данных, удобно внести уже имеющиеся данные и обновить поле в отношении «Выезды» с информацией о времени выезда из депо и заезда обратно позже. Тогда запрос на обновление будет выглядеть следующим образом:

UPDATE Выезды

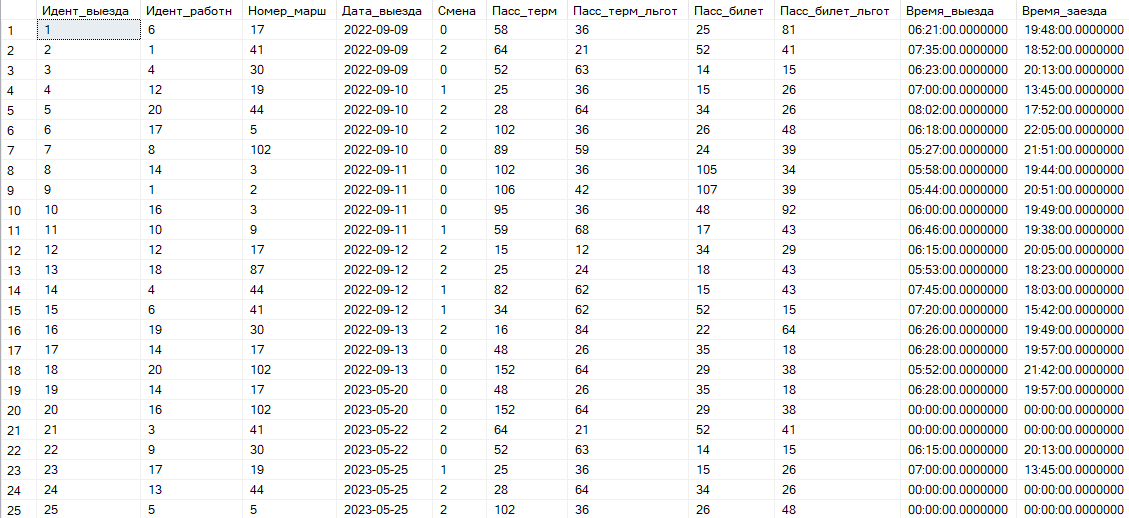
SET Время\_выезда = '6:23'

WHERE Идент\_выезда = 3

Результатом данного запроса будет измененное поле Время\_выезда, Идент\_выезда которого имеет значение 3. Изменение можно наблюдать на рис. 14-15.

****

**Рис 14 - Отношение «Выезды» до изменения**

****

**Рис 15 - Отношение «Выезды» после изменения**

Из понятия SQL, операции DELETE можно сделать вывод о том, что операция удаления DELETE – это DML-операция удаления записей из таблицы. Условие для удаления определяется выражением, прописанным после WHERE. В случае, если условие удаления не определенно, то из отношения удаляются все записи.

В качестве примера удаления был создан запрос, который удаляет все записи из отношения «Обслуживание», дата которых ‘2022-09-09’. Запрос выглядит следующим образом:

DELETE Обслуживание

WHERE Дата\_обслуж = '2022-09-09'

Результат выполнения данного запроса представлен на рис. 16.

****

**Рис 16 - Отношение «Обслуживание» после удаления записей**

### 3.4.2. Запросы с использованием подзапросов

В данном разделе приводятся тексты запросов на модификацию данных, в теле которых присутствуют подзапросы для извлечения данных.

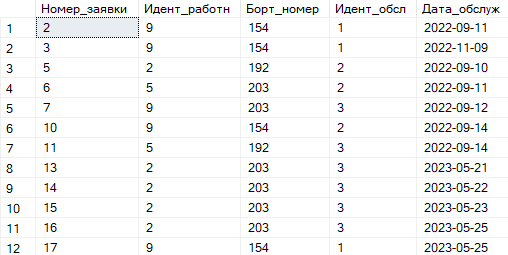
Запрос, созданный с использованием подзапроса, выполняет удаление транспортных средств, срок жизни которых превышает 40 лет.

DELETE Обслуживание

WHERE Борт\_номер IN (SELECT Борт\_номер FROM Транспорт

WHERE (YEAR(GETDATE())-Год\_выпуска > 40))

Результат выполнения данного запроса представлен на рис. 17.

****

**Рис 17 - Отношение «Обслуживание» после выполнения запроса**

## 3.5. Хранимые процедуры и функции, определяемые пользователями

Функции и хранимые процедуры представляют собой блок кода или запросов, хранящихся в базе данных. Они обеспечивают возможность повторного использования и гибкость.

### 3.5.1. Создание хранимых процедур с выходными параметрами

Под хранимой процедурой понимают оформленный особым образом сценарий, который хранится в базе данных и выполняется при его вызове.

К свойствам пользовательских процедур относят:

* процедуры позволяют передавать входные параметры и получать сформированные в них значения в виде возвращаемых выходных параметров;
* возможность предусмотреть возврат значения в точку вызова;
* возможность обеспечить возврат результирующих наборов.

Выходные параметры позволяют хранимой процедуре возвращать данные вызывающей программе.

В следующей процедуре выходной параметр используется для возврата уникального идентификатора добавленного рейса:

CREATE PROCEDURE Добавление\_выезда

@ИД\_выезда INT OUT,

@ИД\_раб INTEGER,

@Номер\_марш INTEGER,

@Дата DATE,

@Смена INTEGER,

@Пасс\_терм INTEGER,

@Пасс\_терм\_льгот INTEGER,

@Пасс\_билет INTEGER,

@Пасс\_билет\_льгот INTEGER,

@Время\_выезда TIME,

@Время\_заезда TIME

AS

INSERT Выезды (Идент\_работн, Номер\_марш, Дата\_выезда, Смена, Пасс\_терм, Пасс\_терм\_льгот, Пасс\_билет, Пасс\_билет\_льгот, Время\_выезда, Время\_заезда)

VALUES (@ИД\_раб, @Номер\_марш, @Дата, @Смена, @Пасс\_терм, @Пасс\_терм\_льгот, @Пасс\_билет, @Пасс\_билет\_льгот, @Время\_выезда, @Время\_заезда)

SET @ИД\_выезда=@@IDENTITY

В данной процедуре мы заполняем информацию о совершенном рейсе и получаем присвоенный ему номер (идентификатор).

Вызов данной процедуры будет осуществляться при помощи операторов DECLARE и EXEC | EXECUTE. Тогда запрос будет выглядеть следующим образом:

DECLARE @ИД\_выезда INT

EXEC Добавление\_выезда

@ИД\_раб = 14,

@Номер\_марш = 102,

@Дата = '2023-05-25',

@Смена = 2,

@Пасс\_терм = 54,

@Пасс\_терм\_льгот = 23,

@Пасс\_билет = 82,

@Пасс\_билет\_льгот = 71,

@Время\_выезда = '6:48',

@Время\_заезда = '20:51',

@ИД\_выезда = @ИД\_выезда OUTPUT

SELECT @ИД\_выезда as 'Номер последнего добавленного рейса'

Результат выполнения процедуры представлен на рис. 18.

****

**Рис 18 - Результат выполнения хранимой процедуры с выходными параметрами**

### 3.5.2. Создание скалярной функции

Скалярные функции – это функции, возвращающие скалярное значение заданного типа данных.

Применим функцию скалярного типа для вычисления количества запланированных, но не состоявшихся рейсов за текущий день. У таких записей дата равна текущей, а время выезда представляет собой пустое поле.

Создание функции происходит с помощью команды CREATE FUNCTION с указанием возвращаемого значения после оператора RETURN:

CREATE FUNCTION Выезды\_сегодня (@data DATE)

RETURNS INT AS

BEGIN

DECLARE @cnt INT

SET @cnt = (SELECT COUNT(Идент\_выезда) FROM Выезды

WHERE Дата\_выезда = @data AND Время\_выезда='')

RETURN (@cnt)

END

Вызов скалярной функции с указанием возвращаемого значения выглядит следующим образом:

SELECT dbo.Выезды\_сегодня(CONVERT(DATE, GETDATE()))

Результат выполнения скалярной функции представлен на рис. 19.

****

**Рис 19 - Результат выполнения скалярной функции**

### 3.5.3. Создание функций с типом данных TABLE

Табличные функции – функции, возвращающие данные типа TABLE.

Табличная функция похожа на объект просмотра, но имеет большую эластичность благодаря использованию параметров и расширяет возможности индексированного объекта просмотра.

Создадим функцию для подсчета пассажиров с льготами и без них, а также найдем общее число пассажиров на данном рейсе:

CREATE FUNCTION Пассажиры()

RETURNS TABLE

AS RETURN

(SELECT Номер\_марш, Дата\_выезда, Смена,

Пасс\_билет + Пасс\_терм as [Пассажиров без льгот],

Пасс\_билет\_льгот + Пасс\_терм\_льгот as [Пассажиров с льготами],

Пасс\_билет + Пасс\_билет\_льгот + Пасс\_терм + Пасс\_терм\_льгот as [Общее число пассажиров]

FROM Выезды

WHERE Дата\_выезда = CONVERT(DATE, GETDATE()))

Вызов функции происходит обычным запросом:

SELECT \* FROM Пассажиры()

Результат выполнения функции с типом данных TABLE представлена на рис. 20.

****

**Рис 20 - Результат выполнения табличной функции**

## 3.6. Описание триггеров

Триггер – это откомпилированная SQL-процедура, исполнение которой обусловлено наступлением определенных событий внутри реляционной базы данных.

С помощью ограничений и значений по умолчанию не всегда можно добиться нужного уровня функциональности. Часто требуется реализовать сложные алгоритмы проверки данных, гарантирующие их достоверность и реальность. Кроме того, иногда необходимо отслеживать изменения значений таблицы, чтобы нужным образом изменить связанные данные.

Триггеры можно рассматривать как своего рода фильтры, вступающие в действие после выполнения всех операций в соответствии с ограничениями, стандартными значениями.

С помощью триггеров достигаются следующие цели:

* проверка корректности введенных данных и выполнение сложных ограничений целостности данных, которые трудно поддерживать с помощью ограничений целостности, установленных для таблицы;
* выдача предупреждений, напоминающих о необходимости выполнения некоторых действий при обновлении таблицы, реализованном определенным образом;
* накопление аудиторской информации посредством фиксации сведений о внесенных изменениях и тех лицах, которые их выполнили.

Можно сказать о том, что триггеры представляют специальный тип хранимой процедуры в СУБД SQL Server, которая выполняется автоматически при выполнении определенного действия над таблицами или представлениями, в частности, при добавлении, изменении или удалении данных, то есть при выполнении команд INSERT, UPDATE, DELETE.

Существует три типа триггеров:

1. INSERT TRIGGER – запускаются (выполняются) при попытке вставки данных с помощью оператора INSERT.
2. UPDATE TRIGGER – запускаются (выполняются) при попытке изменения данных с помощью команды UPDATE.
3. DELETE TRIGGER – запускаются (выполняются) при попытке удаления данных из таблиц или представлений с помощью команды DELETE.

Для базы данных троллейбусного управления были созданы следующие триггеры:

1. Триггер INSERT, не позволяющий добавлять новые виды транспорта. В случае попытки добавить запись, выводится сообщение о невозможности этого сделать. Запрос выглядит следующим образом:

CREATE TRIGGER [Добавление видов транспорта] ON Виды\_транспорта

INSTEAD OF INSERT

AS

BEGIN

PRINT 'Добавление записей в данную таблицу невозможно.'

END

1. Триггер на отслеживание изменений в таблице:

В троллейбусном управлении имеется несколько диспетчеров, способных добавлять, изменять или удалять данные. Чтобы видеть, кто и когда совершал данные операции, директор должен иметь механизм для получения подобных сведений.

Таким механизмом является следующий триггер:

CREATE TRIGGER [Отслеживание изменений] ON Выезды

FOR INSERT, UPDATE, DELETE

AS

UPDATE Выезды

SET Выезды.UserBy = USER\_NAME(),

Выезды.UpdateWhen = GETDATE()

FROM inserted, Выезды

WHERE inserted.Идент\_выезда = Выезды.Идент\_выезда;

При необходимости любой триггер можно отключить командой:

DISABLE TRIGGER [Название триггера] ON [Название таблицы | DATABASE | ALL SERVER];

И включить обратно:

ENABLE TRIGGER [Название триггера] ON [Название таблицы | DATABASE | ALL SERVER];

Запрос на удаление триггера имеет вид:

DROP TRIGGER [Название триггера]

## 3.7. Создание пользователей и назначение привилегий

Пользователь базы данных – это идентификатор имени при подключении к базе данных. Имя пользователя базы данных может совпадать с именем входа, но это не является обязательным требованием. Для соединения с базой данных имя входа должно быть сопоставлено с пользователем базы данных.

Каждый пользователь в SQL базе данных имеет набор привилегий – набор допустимых действий с элементами базы данных. Они могут изменяться со временем: удаляться или обновляться. Некоторые из этих привилегий определены в ANSI SQL, но имеются и дополнительные привилегии, которые являются также необходимыми. SQL привилегии, как определено ANSI, не достаточны в большинстве ситуаций реальной жизни.

SQL привилегии определенные ANSI – это привилегии объекта. Это означает что пользователь имеет привилегию чтобы выполнить данную команду только на определенном объекте в базе данных.

Привилегии, которые можно назначить пользователю:

* SELECT – пользователь с этой привилегией может выполнять запросы в таблице.
* INSERT – пользователь с этой привилегией может выполнять команду INSERT в таблице.
* UPDATE – пользователь с этой привилегией может выполнять команду UPDATE на таблице. Вы можете ограничить эту привилегию для определенных столбцов таблицы.
* DELETE – пользователь с этой привилегией может выполнять команду DELETE в таблице.

Для базы данных почта создадим следующую группу пользователей: администратор, директор, диспетчер.

**Таблица №9**

**Права доступа к таблицам для групп пользователей**

|  |  |  |  |
| --- | --- | --- | --- |
| **Таблица** | **Группа пользователей** | | |
| **Администратор** | **Директор** | **Диспетчер** |
| Должности | ALL | SELECT, INSERT, UPDATE, DELETE | SELECT |
| Персонал | ALL | SELECT, INSERT, UPDATE, DELETE | SELECT |
| Транспорт | ALL | SELECT, INSERT, UPDATE, DELETE | SELECT, UPDATE |
| Виды обслуживания | ALL | SELECT, INSERT, UPDATE, DELETE | SELECT |
| Обслуживание | ALL | SELECT | SELECT, INSERT, UPDATE, DELETE |
| Виды транспорта | ALL | SELECT, INSERT, UPDATE, DELETE | SELECT |
| Маршрут | ALL | SELECT, INSERT, UPDATE, DELETE | SELECT, INSERT, UPDATE, DELETE |
| Выезды | ALL | SELECT, DELETE | SELECT, INSERT, UPDATE, DELETE |

Запрос на создание учетной записи администратора выглядит следующим образом:

CREATE LOGIN Admins WITH PASSWORD='ADMINpass'

GO

CREATE USER AdminUser FOR LOGIN Admins

GO

GRANT ALL ON Должности TO AdminUser

GRANT ALL ON Персонал TO AdminUser

GRANT ALL ON Транспорт TO AdminUser

GRANT ALL ON Вид\_обсл TO AdminUser

GRANT ALL ON Обслуживание TO AdminUser

GRANT ALL ON Виды\_транспорта TO AdminUser

GRANT ALL ON Маршрут TO AdminUser

GRANT ALL ON Выезды TO AdminUser

Пользователь с администраторской учетной записью имеет все права для работы с таблицами, указанными в запросе.

Запрос на создание учетной записи директора в базе данных троллейбусного управления:

CREATE LOGIN Director WITH PASSWORD='DIRECTORpass'

GO

CREATE USER DirectorUser FOR LOGIN Director

GO

GRANT SELECT, INSERT, UPDATE, DELETE ON Должности TO DirectorUser

GRANT SELECT, INSERT, UPDATE, DELETE ON Персонал TO DirectorUser

GRANT SELECT, INSERT, UPDATE, DELETE ON Транспорт TO DirectorUser

GRANT SELECT, INSERT, UPDATE, DELETE ON Вид\_обсл TO DirectorUser

GRANT SELECT ON Обслуживание TO DirectorUser

GRANT SELECT, INSERT, UPDATE, DELETE ON Виды\_транспорта TO DirectorUser

GRANT SELECT, INSERT, UPDATE, DELETE ON Маршрут TO DirectorUser

GRANT SELECT, DELETE ON Выезды TO DirectorUser

Учетная запись директора лишена некоторых возможностей по работе с таблицами. Это объясняется разделением обязанностей и отсутствием необходимости для данной должности.

Запрос на создание учетной записи диспетчера выглядит следующим образом:

CREATE LOGIN Disp\_Helen WITH PASSWORD='DISPpass'

GO

CREATE USER DispUser1 FOR LOGIN Disp\_Helen

GO

GRANT SELECT ON Должности TO DispUser1

GRANT SELECT ON Персонал TO DispUser1

GRANT SELECT, UPDATE ON Транспорт TO DispUser1

GRANT SELECT ON Вид\_обсл TO DispUser1

GRANT SELECT, INSERT, UPDATE, DELETE ON Обслуживание TO DispUser1

GRANT SELECT, INSERT, UPDATE, DELETE ON Виды\_транспорта TO DispUser1

GRANT SELECT, INSERT, UPDATE, DELETE ON Маршрут TO DispUser1

GRANT SELECT, INSERT, UPDATE, DELETE ON Выезды TO DispUser1

Безопасность баз данных – это понятие, которое относится к использованию широкого спектра средств защиты информации для защиты конфиденциальности, целостности и доступности баз данных. Он включает в себя различные типы категории контроля, такие как технические, процедурные/административные и физические. Могут использоваться комбинации различных методов защиты. Важно понимать, что комплексную безопасность баз данных нужно строить сочетанием разных способов и методов.

В данный момент времени SQL Server предоставляет архитектуру безопасности, которая позволяет администраторам баз данных и разработчикам создавать защищенные приложения баз данных и бороться с угрозами. Одним из механизмов архитектуры безопасности, безусловно, является правильное разделение полномочий пользователей.

# **Заключение**

В процессе курсового проектирования я научилась применять на практике знания, полученные в процессе изучения курса «Базы данных» и получил практические навыки проектирования и реализации баз данных, для последующего создания информационных систем.

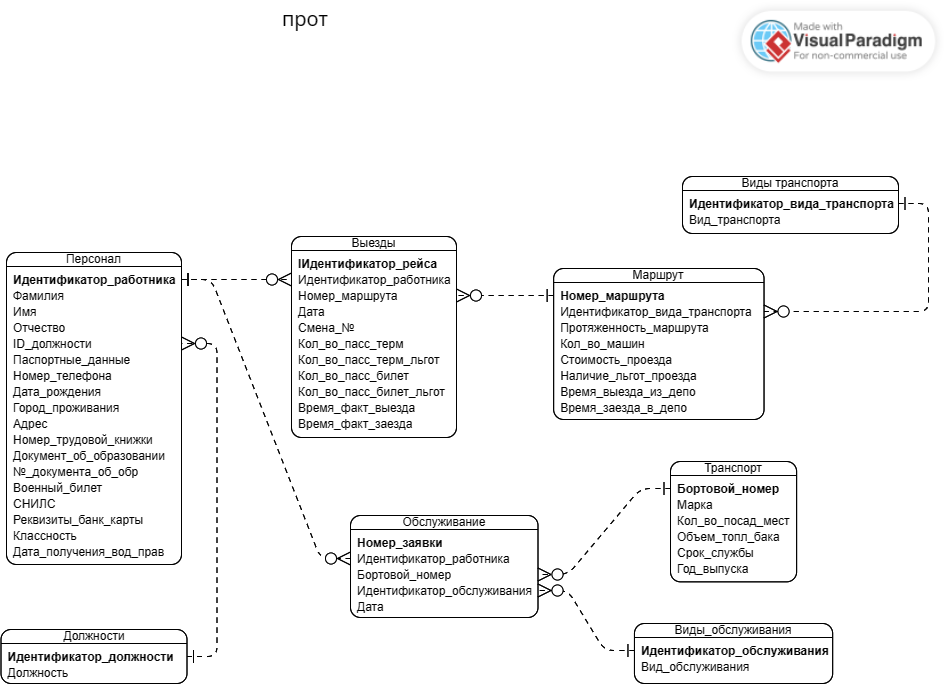
Результатом работы является разработанная реляционная база данных троллейбусного управления, которая упрощает регистрацию запланированных и совершенных рейсов, отслеживание информации о них и дальнейший анализ.

# **Список использованной литературы**

1. Копирование баз данных путем создания и восстановления резервных копий. Режим доступа: <https://learn.microsoft.com/Ru-Ru/sql/relational-databases/databases/copy-databases-with-backup-and-restore?view=sql-server-2016>, дата обращения: 13.05.2023.
2. CREATE TRIGGER (Transact-SQL). Режим доступа: <https://learn.microsoft.com/ru-ru/sql/t-sql/statements/create-trigger-transact-sql?view=sql-server-ver16>, дата обращения: 20.05.2023.
3. SELECT — предложение OVER. Режим доступа: <https://learn.microsoft.com/ru-RU/sql/t-sql/queries/select-over-clause-transact-sql?view=sql-server-linux-2017>, дата обращения: 20.05.2023.
4. Оконные функции SQL простым языком с примерами. Режим доступа: <https://habr.com/ru/articles/664000/>, дата обращения: 21.05.2023.

# **Приложение 1**

**Инфологическая модель базы данных**



# **Приложение 2**

**Диаграмма базы данных**

