|  |  |
| --- | --- |
| Министерство транспорта Российской Федерации | |
| Федеральное государственное автономное образовательное | |
| учреждение высшего образования | |
|  |  |
| «Российский университет транспорта» (РУТ (МИИТ) | |
| Институт транспортной техники и систем управления | |
|  |  |
| Кафедра «Управление и защита информации» | |
|  |  |
|  |  |
| Курсовая работа | |
|  |  |
|  |  |
| **«Информационное обеспечение систем управления»** | |
|  |  |
| На тему: | |
| «Проектирование реляционной базы данных» | |
|  |  |
|  | Выполнили: ст. гр. ТУУ-411 |
|  | Белинский А.С. |
|  | Вариант №3 |
|  | Проверил: доц. Васильева М. А. |
|  |  |
|  |  |
| Москва 2024 | |

**Содержание**

[1. Введение 2](#_Toc163736581)

[1.1. Цель работы 2](#_Toc163736582)

[1.2. Описание задачи 2](#_Toc163736583)

[1.3. Исходные данные 2](#_Toc163736584)

[2. Проектирование БД 3](#_Toc163736585)

[2.1. Инфологическое проектирование 3](#_Toc163736586)

[2.1.1. Анализ предметной области 3](#_Toc163736587)

[2.1.2. Анализ информационных задач и круга пользователей системы 5](#_Toc163736588)

[2.2. Логическое проектирование реляционной БД 5](#_Toc163736589)

[2.2.1. Преобразование ER-диаграммы в схему баз данных 5](#_Toc163736590)

[2.2.2. Составление реляционных отношений 5](#_Toc163736591)

[2.2.3. Нормализация полученных отношений 7](#_Toc163736592)

[2.3. Физическое проектирование БД 8](#_Toc163736593)

[2.3.1. Разработка скриптов на создание базы данных и таблиц 8](#_Toc163736594)

[2.3.2. Разработка необходимых представлений (view) 11](#_Toc163736595)

[2.3.3. Разработка необходимых функций и процедур 14](#_Toc163736596)

[3. Заключение 21](#_Toc163736597)

[3.1. Вывод 21](#_Toc163736598)

# Введение

## Цель работы

Получить первичные навыки работы с проектированием реляционных баз данных.

## Описание задачи

Спроектировать базу данных по следующим шагам:

* Инфологическое проектирование
  + Анализ предметной области
  + Анализ информационных задач и круга пользователей системы
* Логическое проектирование реляционной БД
  + Преобразование ER-диаграммы в схему баз данных
  + Составление реляционных отношений
  + Нормализация полученных отношений
* Физическое проектирование БД
  + Разработка скриптов на создание и изменение таблиц БД
  + Разработка скриптов на добавление данных в таблицы БД
  + Разработка необходимых представлений
  + Разработка необходимых функций и процедур
  + Разработка необходимых триггеров

## Исходные данные

В качестве тематики базы данных используется Таксопарк и соответствующие ему сущности.

# Проектирование БД

## Инфологическое проектирование

### Анализ предметной области

Анализ предметной области на основании представленной схемы базы данных

ER-диаграмма БД таксопарка представлена на рисунке 1.

|  |
| --- |
|  |
| 1. – ER-диаграмма БД таксопарка |

Таблицы и их поля:

Таблица - users

id: bigint — уникальный идентификатор пользователя (первичный ключ).

driving\_experience: integer — опыт вождения в годах.

name: character varying(255) — имя пользователя.

passport\_serial\_number: character varying(255) — серийный номер паспорта.

patronymic: character varying(255) — отчество пользователя.

phone\_number: character varying(255) — телефонный номер.

qualification\_upgrade\_required: boolean — флаг необходимости повышения квалификации.

surname: character varying(255) — фамилия пользователя.

Таблица - aggregators

id: bigint — уникальный идентификатор агрегатора.

commission: double precision — комиссия агрегатора.

name: character varying(255) — название агрегатора.

office\_address: character varying(255) — адрес офиса агрегатора.

partner\_count: integer — количество партнёров агрегатора.

Таблица - cars

id: bigint — уникальный идентификатор автомобиля.

brand: character varying(255) — марка автомобиля.

insurance\_expiration\_date: Date — срок окончания страховки.

mileage: integer — пробег автомобиля.

repair\_required: boolean — указывает, требуется ли ремонт.

year\_of\_manufacture: integer — год выпуска автомобиля.

Таблица - taxi\_parks

id: bigint — уникальный идентификатор автопарка.

address: character varying(255) — адрес автопарка.

aggregator\_id: bigint — связь с агрегаторами, указывает на агрегатора, которому принадлежит автопарк.

owner\_id: bigint — идентификатор владельца автопарка.

Таблица - taxi\_parks\_cars

taxi\_park\_id: bigint — идентификатор автопарка.

cars\_id: bigint — идентификатор автомобиля, принадлежащего к автопарку

### Анализ информационных задач и круга пользователей системы

Основная задача системы — управление информацией о пользователях, автомобилях, автопарках и их взаимоотношениях с агрегаторами. Для пользователей системы (водителей, владельцев автопарков, сотрудников агрегаторов) необходимо обеспечивать оперативный доступ к данным о состоянии техники, договорных обязательствах и финансовых расчетах. Система должна также помогать в планировании регулярного обслуживания автомобилей, обновлении квалификации водителей, и прослеживать сроки страховки.

## Логическое проектирование реляционной БД

### Преобразование ER-диаграммы в схему баз данных

Для формализации структуры данных системы я начал с создания ER-диаграммы (Entity-Relationship Diagram), чтобы визуализировать связи между сущностями. Ключевые сущности — это users, aggregators, cars и taxi\_parks. Сущность taxi\_parks\_cars выступает в качестве связующего элемента между автопарками и автомобилями, показывая, какие автомобили к каким автопаркам относятся.

### Составление реляционных отношений

Для эффективной реализации базы данных, реляционные отношения должны быть тщательно спланированы и реализованы. Отношения между таблицами определяются на основе ключей, обеспечивающих связь данных между сущностями системы. Ниже представлены ключевые таблицы и их связи:

1. Таблица users

- Columns: id, name, surname, patronymic, passport\_serial\_number, phone\_number, driving\_experience, qualification\_upgrade\_required

- Primary Key: id

- Description: Хранит информацию о пользователях (водителях).

2. Таблица aggregators

- Columns: id, name, commission, office\_address, partner\_count

- Primary Key: id

- Description: Содержит информацию о агрегаторах, включая их комиссию и количество партнеров.

3. Таблица cars

- Columns: id, brand, year\_of\_manufacture, mileage, insurance\_expiration\_date, repair\_required

- Primary Key: id

- Description: Хранит информацию об автомобилях, включая марку, год выпуска, и срок окончания страховки.

4. Таблица taxi\_parks

- Columns: id, address, aggregator\_id, owner\_id

- Primary Key: id

- Foreign keys:

- aggregator\_id references aggregators(id)

- owner\_id references users(id)

- Description: Содержит информацию об автопарках, включая адрес и связь с агрегаторами и владельцами.

5. Таблица taxi\_parks\_cars

- Columns: taxi\_park\_id, cars\_id

- Primary Key: (taxi\_park\_id, cars\_id)

- Foreign keys:

- taxi\_park\_id references taxi\_parks(id)

- cars\_id references cars(id)

- Description: Многие ко многим связь между автопарками и автомобилями показывает, какие автомобили к каким автопаркам относятся.

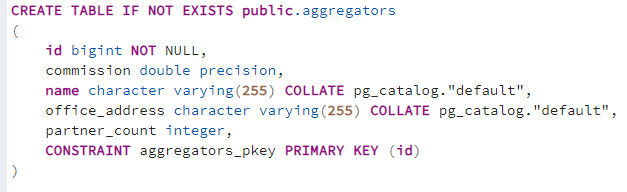
### Нормализация полученных отношений

Для оптимизации структуры базы данных я провел нормализацию до третьей нормальной формы. Целью нормализации было избежание избыточности данных и потенциальных аномалий при модификации данных. Так, я убедился, что в каждой таблице данные связаны только с ключом, данные не зависят нефункционально от части первичного ключа, и нефункциональные зависимости каждого атрибута исключены.

## Физическое проектирование БД

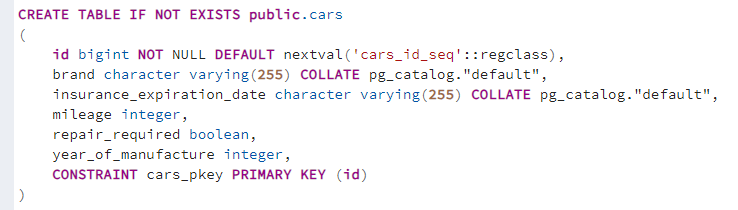
### Разработка скриптов на создание базы данных и таблиц

Создание таблицы aggregators представлено на рисунке 2.



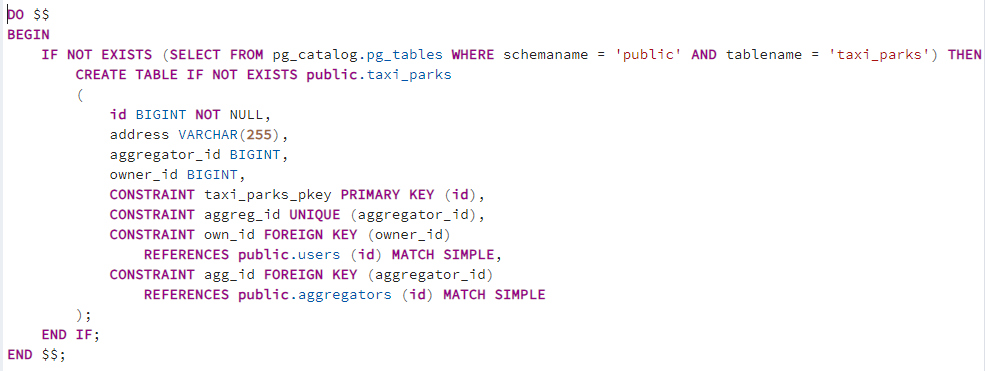
1. - Создание таблицы aggregators

Создание таблицы cars представлено на рисунке 3.



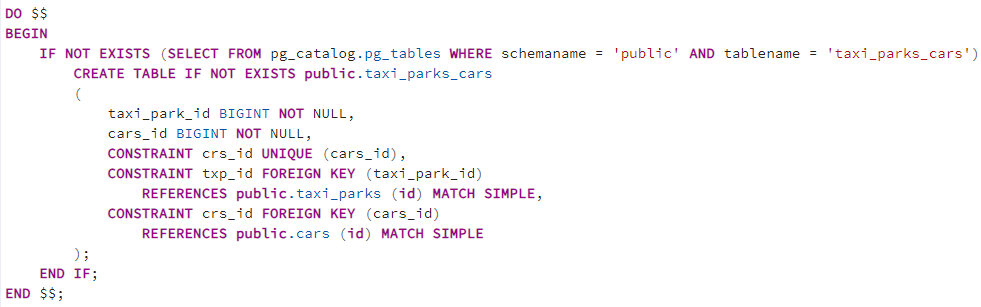
1. - Создание таблицы cars

Создание таблицы taxi\_parks представлено на рисунке 4.



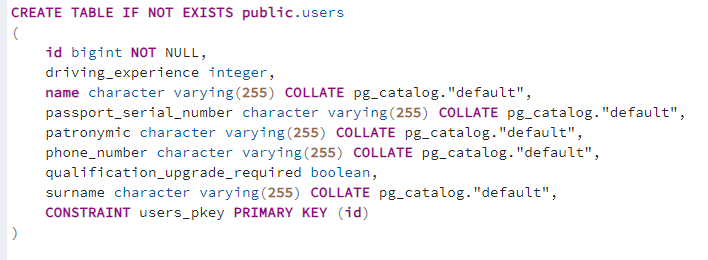
1. - Создание таблицы taxi\_parks

Создание таблицы taxi\_parks\_cars представлено на рисунке 5.



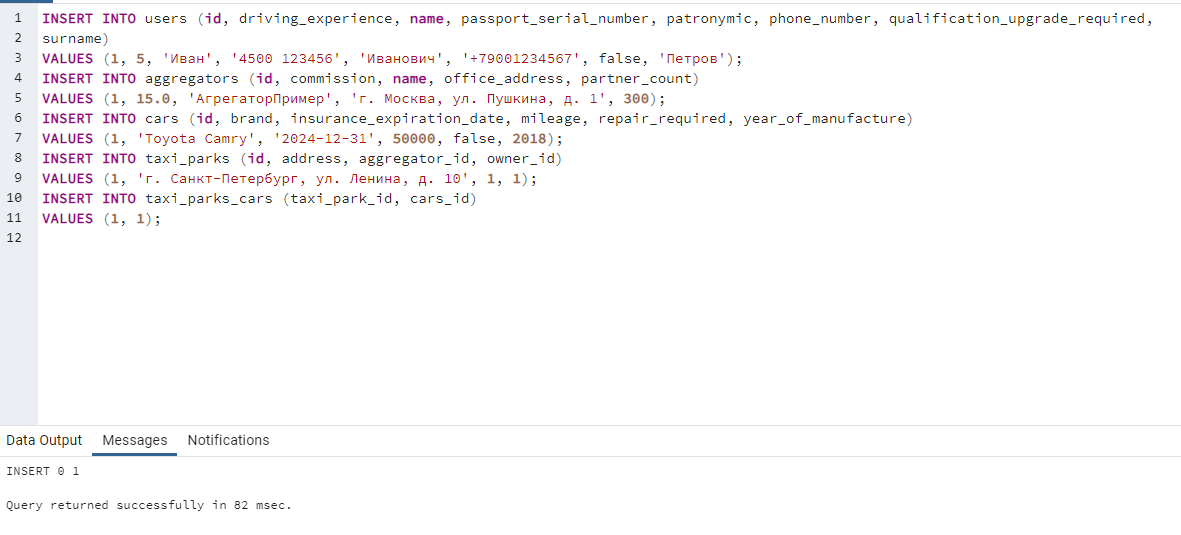
1. - Создание таблицы taxi\_parks\_cars

Создание таблицы usersпредставлено на рисунке 6.



1. - Создание таблицы users

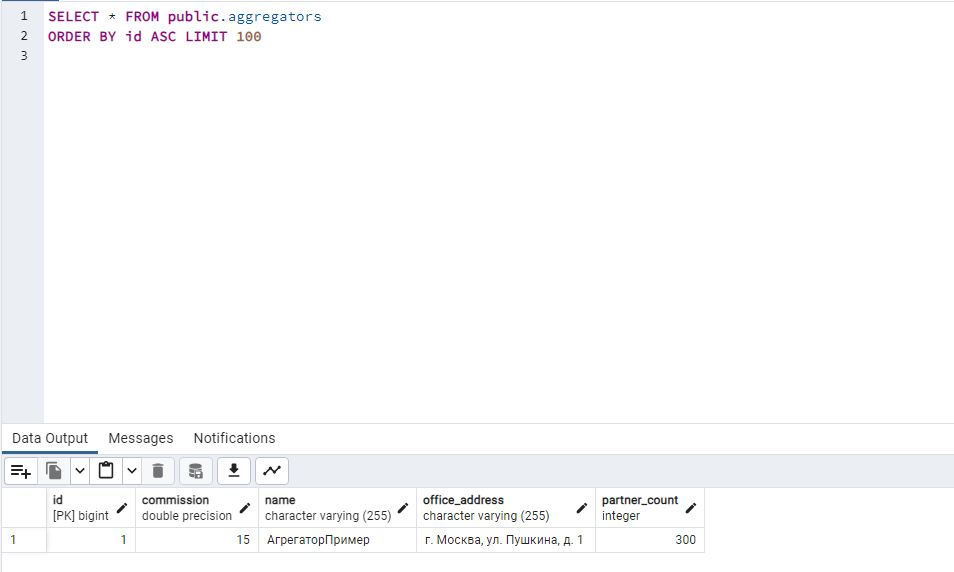
Заполнение таблиц представлено на рисунке 7.



1. – Заполнение таблиц

Каждый из этих запросов вносит строки в соответствующие таблицы, и вместе они формируют начальное наполнение вашей базы данных такси

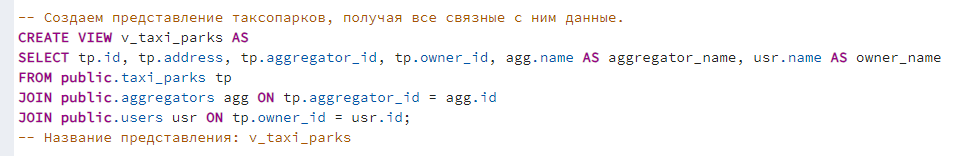
Заполненные данные на примере таблицы aggregators представлены на рисунке 8.



1. – Результат работы insert

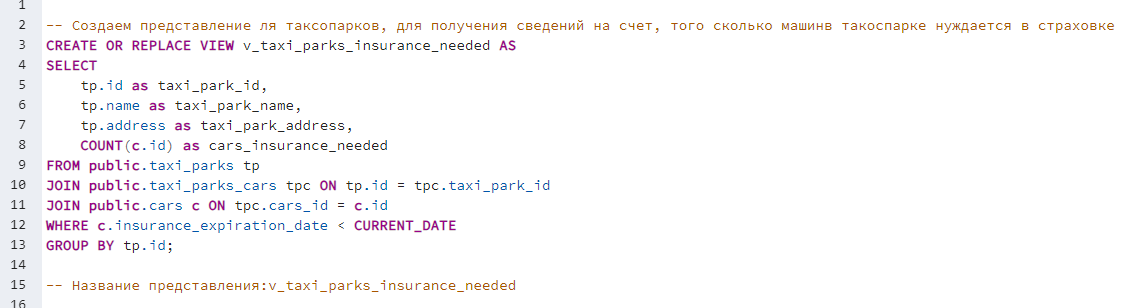
### Разработка необходимых представлений (view)

Создание представления v\_taxi\_parks представлено на рисунке 9.



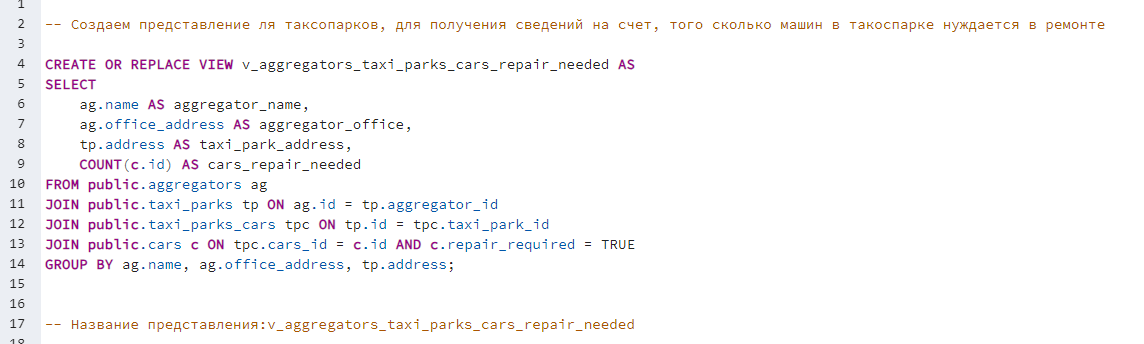
1. - Создание представления v\_taxi\_parks

Создание представления v\_taxi\_parks\_insurance\_needed представлено на рисунке 10.



1. - Создание представления v\_taxi\_parks\_insurance\_needed

Создание представления v\_aggregators\_taxi\_parks\_cars\_repair\_needed представлено на рисунке 11.



1. - Создание представления v\_aggregators\_taxi\_parks\_cars\_repair\_needed

Каждое из созданных представлений (views) в базе данных служит определенной цели, облегчая доступ к часто запрашиваемой или специфически составленной информации относительно различных аспектов управления таксопарком и взаимодействия с агрегаторами. Ниже описаны цели каждого представления:

1. Представление v\_aggregators\_taxi\_parks\_cars\_repair\_needed

Цель:

Представление создано для облегчения получения информации о количестве автомобилей, требующих ремонта, в каждом таксопарке в зависимости от агрегатора. Это позволяет установить, какие таксопарки и их агрегаторы имеют наибольшее количество автомобилей, которые требуют внимания. Используя эту информацию, агрегаторы могут планировать ресурсы на ремонт или обновление автопарка.

Детали:

- Представление объединяет информацию из таблиц aggregators, taxi\_parks, taxi\_parks\_cars и cars.

- Фильтрация выполняется для автомобилей, которым требуется ремонт (условие repair\_required = TRUE).

- Агрегация данных выполняется по агрегатору и адресу таксопарка для подсчета количества автомобилей нуждающихся в ремонте.

2. Представление v\_taxi\_parks\_insurance\_needed

Цель:

Представление предназначено для отслеживания автомобилей в таксопарках, страховка которых истекает (дата окончания страховки меньше текущей даты). Это критично для управления рисками и избежания ситуаций, которые могут привести к юридическим проблемам или финансовым потерям.

Детали:

- Соединение осуществляется между таблицами taxi\_parks, taxi\_parks\_cars и cars.

- Происходит фильтрация автомобилей по полю insurance\_expiration\_date, чтобы найти те, чья страховка уже истекла.

- Результат группируется по ID и адресу таксопарка для подсчета количества автомобилей с истекшей страховкой.

3. Представление v\_taxi\_parks

Цель:

Это представление предоставляет полную информацию о таксопарках, включая информацию о соответствующих агрегаторах и владельцах. Оно упрощает управление и отслеживание связей между различными сущностями в базе данных, особенно полезно для оперативного анализа и вывода отчетов.

Детали:

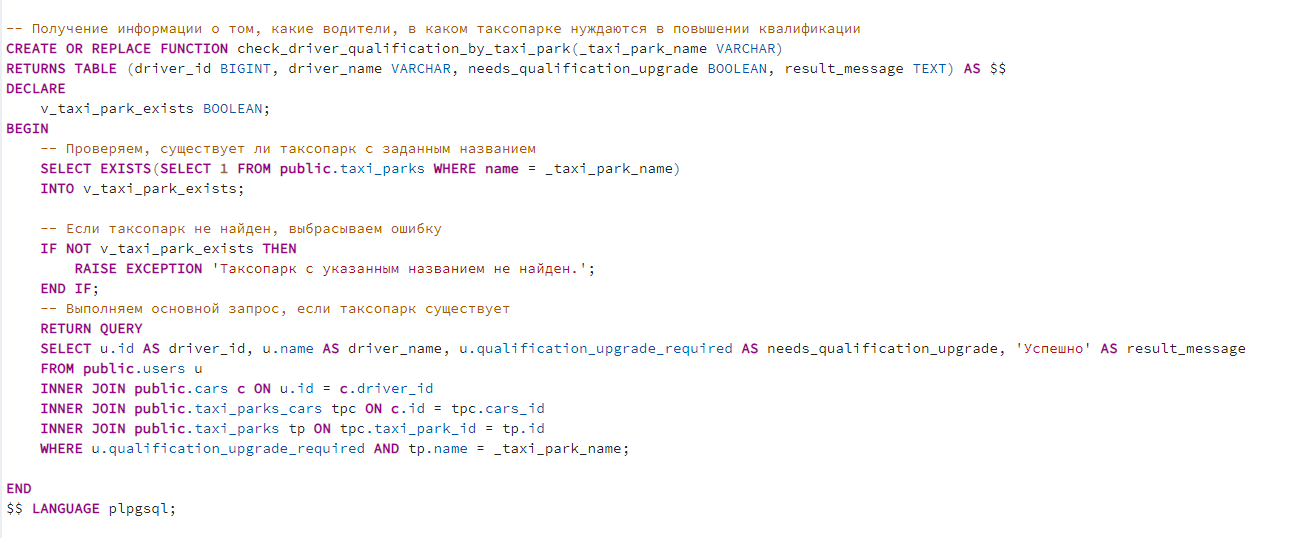
- Соединяет основные данные о таксопарках с дополнительной информацией о агрегаторах и пользователях (владельцах).

- Предоставляет централизованное место для доступа к объединенной информации, что упрощает процессы принятия решений и управления.

Каждое из этих представлений специально разработано для повышения эффективности аналитических и операционных задач, связанных с управлением таксопарками и их взаимодействием с агрегаторами и владельцами.

### Разработка необходимых функций и процедур

Функция check\_driver\_qualification\_by\_taxi\_park в PostgreSQL представленна на рисунке 12. Получение информации о том, какие водители, в каком таксопарке нуждаются в повышении квалификации



1. - Процедура для получение информации о том, какие водители, в каком таксопарке нуждаются в повышении квалификации

Эта процедура, оформленная в виде пользовательской функции SQL, предназначена для получения информации о водителях в определенном таксопарке, которым необходимо повышение квалификации. Вот подробное описание ее функционала:

Назначение функции

Функция check\_driver\_qualification\_by\_taxi\_park предназначена для проверки потребности в повышении квалификации водителей конкретного таксопарка. Это позволяет управляющей компании или администратору таксопарка определять, какие водители нуждаются в дополнительном обучении или тренинге, что, в свою очередь, может помочь в улучшении качества услуг и повышении безопасности перевозок.

Работа функции

1. Ввод параметра: Функция принимает название таксопарка в виде строки (\_taxi\_park\_name).

2. Проверка существования таксопарка: Сначала функция проверяет, существует ли таксопарк с предоставленным названием в базе данных. Если таксопарк не найден, функция генерирует исключение с сообщением, что таксопарк с указанным названием не найден.

3. Основной запрос:

- В случае успешного нахождения названия таксопарка выполняется запрос к базе данных.

- Функция извлекает информацию о водителях (id и name из таблицы public.users), которым требуется повышение квалификации (qualification\_upgrade\_required = TRUE).

- Для связи водителей с автомобилями используется таблица public.cars, а связь между автомобилями и таксопарками осуществляется через таблицу public.taxi\_parks\_cars.

- Окончательный список формируется только для тех водителей, чьи автомобили зарегистрированы в соответствующем таксопарке, имя которого было передано в функцию.

4. Возвращаемые значения:

- driver\_id: Идентификатор водителя.

- driver\_name: Имя водителя.

- needs\_qualification\_upgrade: Логическое значение, указывающее на необходимость повышения квалификации водителя.

- result\_message: Текстовое сообщение с результатами выполнения запроса. В случае успешного выполнения возвращает "Успешно".

Эта функция играет важную роль в управлении кадрами и обеспечении качества в таксопарках, позволяя быстро и эффективно идентифицировать водителей, которым требуется дополнительное обучение или сертификация. Это способствует повышению профессионализма и улучшению обслуживания клиентов.

Функция add\_taxi\_park в PostgreSQL представленна на рисунке 13.



1. – Функция add\_taxi\_park

Эта процедура предназначена для добавления нового таксопарка в базу данных. Процедура выполняет ряд проверок для валидации введенных данных и убедится, что не возникнет дублирования или некорректности информации. Ниже приведено детальное описание работы процедуры и ее компонентов.

Параметры:

- i\_park\_name: Имя таксопарка.

- i\_park\_address: Адрес таксопарка.

- i\_aggregator\_name: Имя агрегатора, к которому будет привязан таксопарк.

- o\_added: Флаг, показывающий, был ли таксопарк успешно добавлен.

- o\_park\_id: Идентификатор добавленного таксопарка.

- o\_park\_name: Имя добавленного таксопарка.

- o\_address: Адрес добавленного таксопарка.

- o\_aggregator\_name: Имя агрегатора, к которому привязан таксопарк.

Процесс выполнения:

1. Валидация входных данных: Процедура проверяет, не пусты ли предоставленные значения имени таксопарка, адреса и имени агрегатора. В случае обнаружения пустого или нулевого значения будет сгенерировано исключение.

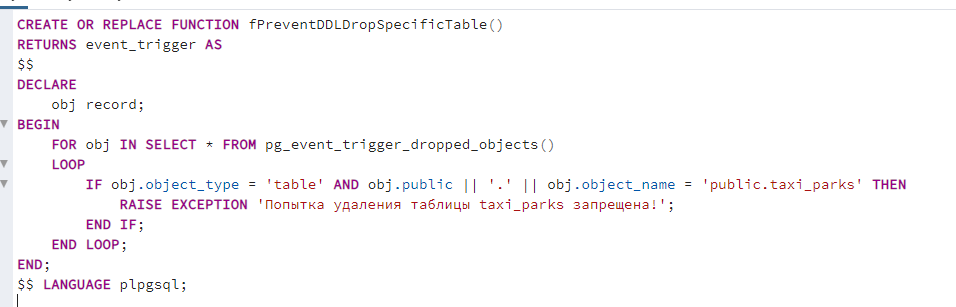
2. Проверка существования таксопарка: Далее происходит проверка, существует ли уже таксопарк с таким именем. Если таксопарк найден, процедура устанавливает флаг o\_added в значение FALSE и генерирует исключение о существовании таксопарка.

3. Поиск ID агрегатора: Процедура ищет в базе данных ID агрегатора по его имени. Если агрегатор не найден, генерируется соответствующее исключение.

4. Добавление нового таксопарка: Если все проверки пройдены успешно, процедура добавляет запись о новом таксопарке в базу данных и возвращает его ID.

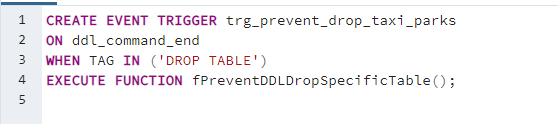
5. Установка возвращаемых значений и уведомления: Процедура устанавливает возвращаемые значения и генерирует уведомление о успешном добавлении таксопарка.

Триггер удаления таблицы представлен на рисунке 14



1. – Логика тригерра для таблицы taxi\_parks

Триггер – ивент удаления таблицы представлен на рисунке 15



1. – Логика тригерра - ивента для таблицы taxi\_parks

Название процедуры: fPreventDDLDropSpecificTable

Назначение: предотвращение удаления таблицы restaurant\_employee в базе данных.

Описание: процедура представляет собой функцию на языке PL/pgSQL, которая создаёт или заменяет существующую функцию. Функция является триггером событий и выполняется при попытке удаления объекта (таблицы) в базе данных PostgreSQL.

Функция перебирает все объекты, которые были удалены, и проверяет, является ли объект таблицей и совпадает ли имя таблицы с именем restaurant\_employee. Если условие выполняется, то функция вызывает исключение, которое сигнализирует о том, что удаление таблицы запрещено.

Таким образом, функция fPreventDDLDropSpecificTable предотвращает удаление таблицы taxi\_parks, что может быть полезно для обеспечения целостности данных и защиты важных таблиц в базе данных.

# Заключение

## Вывод

Были получены первичные навыки работы с проектированием БД.