Министерство науки и высшего образования Российской Федерации
федеральное государственное автономное образовательное учреждение высшего образования
«НАЦИОНАЛЬНЫЙ ИССЛЕДОВАТЕЛЬСКИЙ УНИВЕРСИТЕТ ИТМО»

ОТЧЕТ

по лабораторной работе «Процедуры, функции, триггеры в PostgreSQL»

по дисциплине «**Проектирование и реализация баз данных**»

Автор: Кузьмина Марина Леонидовна

Факультет: ИКТ

Группа: К32421

Преподаватель: Говорова М.М.

Санкт-Петербург 2023

СОДЕРЖАНИЕ

1 Цель работы и практическое задание

Цель работы:

Овладеть практическими создания и использования процедур, функций и триггеров в базе данных PostgreSQL.

Практическое задание:

Вариант 1:

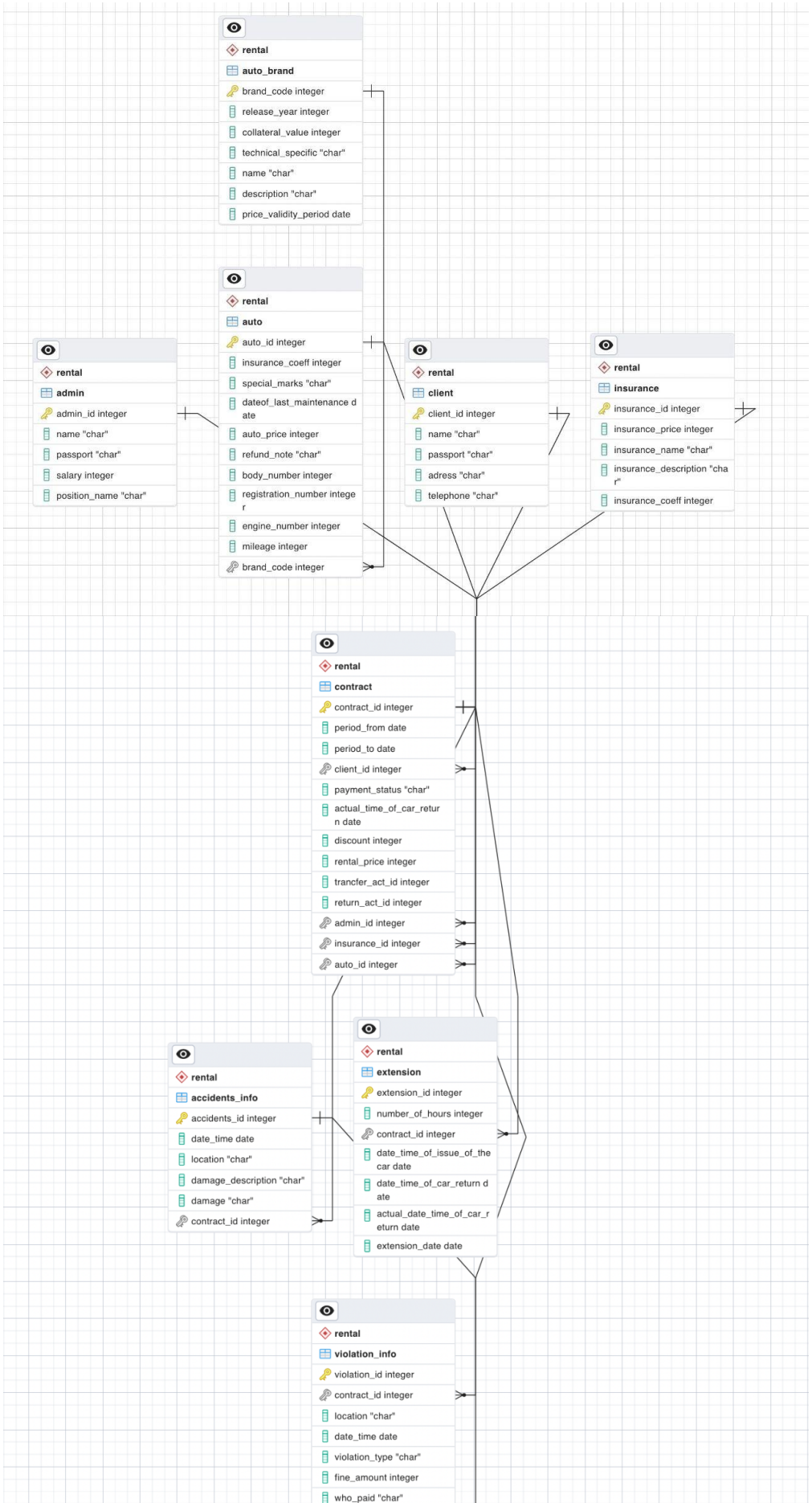
1. Создать процедуры/функции согласно индивидуальному заданию и (согласно индивидуальному заданию, часть 4).
2. Создать триггер для логирования событий вставки, удаления, редактирования данных в базе данных PostgreSQL (согласно индивидуальному заданию, часть 5). Допустимо создать универсальный триггер или отдельные триггеры на логирование действий.

2 Выполнение работы

2.1 Наименование БД

БД «Прокат автомобилей»

2.2 Схема логической модели базы данных, сгенерированная в Generate ERD



2.3 Хранимые процедуры

1. Выполнить списание автомобилей, выпущенных ранее заданного года.

```
CREATE PROCEDURE write_off_cars_by_year (IN year_to_write_off INT) BEGIN -- получаем id всех машин, выпущенных ранее заданного года

 DECLARE car_ids

CURSOR

FOR

SELECT auto_id

FROM auto

WHERE YEAR(release_year) < year_to_write_off; DECLARE car_id INT; DECLARE done INT DEFAULT FALSE; -- идем по списку машин и устанавливаем отметку о списании

 DECLARE CONTINUE

  HANDLER FOR NOT FOUND

  SET done = TRUE; OPEN car_ids; car_loop: LOOP FETCH car_ids INTO car_id; IF done THEN LEAVE car_loop; END IF;

UPDATE auto

SET refund_note = 'written off'
WHERE auto_id = car_id; END LOOP; CLOSE car_ids; END
```

Эта хранимая процедура принимает в качестве параметра год, до которого нужно списать все машины. Она открывает курсор для получения id всех машин, выпущенных ранее заданного года, и затем устанавливает отметку "written off" для каждой машины в этом списке. Курсор закрывается после завершения цикла.

Чтобы вызвать эту процедуру и списать все машины, выпущенные ранее 2010 года, можно использовать следующую команду:

```
CALL write_off_cars_by_year(2010);
```

Результат выполнения:

До:

	brand_code [PK] integer	release_year integer	collateral_value integer	technical_specific "char" (1)
1	1	2009	20	-
2	2	2015	30	-
3	3	2000	15	-
4	4	2018	50	-
5	5	2005	15	-

После:

	brand_code [PK] integer	release_year integer	collateral_value integer	technical_specific "char" (1)
1	1	2009	20	-
2	2	2015	30	-
3	4	2018	50	-

2. Выдачи автомобиля и расчета стоимости с учетом скидки постоянным клиентам.

```
CREATE PROCEDURE rent_car (IN client_id INT, IN auto_id INT, IN rental_period INT, OUT rental_price DECIMAL(10, 2)) BEGIN -- проверяем, является ли клиент постоянным

 DECLARE is_regular_client BOOL;

SET is_regular_client = FALSE;

SELECT COUNT(*) INTO is_regular_client

FROM contract

WHERE client_id = client_id; -- получаем информацию о машине и ее цене

 DECLARE auto_price DECIMAL(10, 2);

  SELECT auto_price INTO auto_price
```

```
FROM auto

WHERE auto_id = auto_id; -- расчет стоимости проката
с учетом скидки

 IF is_regular_client THEN

   SET rental_price = auto_price * rental_period *
0.9; ELSE

  SET rental_price = auto_price * rental_period; END
IF; -- создаем запись в таблице контрактов


    INSERT INTO contract (period_from, period_to,
client_id, payment_status, rental_price, auto_id)

VALUES (NOW(),

              DATE_ADD(NOW(), INTERVAL rental_period
HOUR),

        client_id,

        'unpaid',

        rental_price,

              auto_id); -- обновляем информацию об
автомобиле


UPDATE auto

SET dateof_last_maintenance = NOW()

WHERE auto_id = auto_id; -- помечаем автомобиль как
выданный в аренду


  UPDATE auto

   SET refund_note = 'unreturned' WHERE auto_id =
auto_id; -- увеличиваем количество прокатов у клиента
```

```
UPDATE client
  SET rentals = rentals + 1 WHERE client_id = client_id; -- выводим стоимость проката

SELECT rental_price; END;
```

В данной процедуре мы передаем параметры client_id, auto_id и rental_period, которые определяют id клиента, id автомобиля и период аренды соответственно. Затем мы проверяем, является ли клиент постоянным, определяя количество его контрактов в таблице контрактов.

Далее мы получаем цену машины из таблицы auto и вычисляем стоимость аренды с учетом скидки для постоянных клиентов. Затем мы создаем новую запись в таблице контрактов, обновляем информацию об автомобиле и увеличиваем количество прокатов у клиента.

3. Для вычисления количества автомобилей заданной марки.

```
CREATE PROCEDURE count_cars_by_brand(IN brand_code_param INT, OUT car_count INT) BEGIN
SELECT COUNT(*) INTO car_count
FROM auto
WHERE brand_code = brand_code_param; END
```

Здесь создается хранимая процедура с именем count_cars_by_brand, которая принимает один параметр brand_code_param типа INT и один выходной параметр car_count типа INT.

В теле процедуры используется оператор SELECT с функцией COUNT(*), чтобы подсчитать количество автомобилей с заданным кодом марки (brand_code_param) в таблице auto. Результат этого запроса сохраняется в выходной параметр car_count.

```
DECLARE @car_count INT;
EXEC count_cars_by_brand 123,
                              @car_count OUTPUT;
SELECT @car_count;
```

Здесь используется оператор DECLARE для объявления переменной @car_count типа INT, которая будет использоваться для хранения результата. Затем вызывается процедура count_cars_by_brand, передавая ей значение 123 для параметра brand_code_param и переменную @car_count для выходного параметра. В конце выводится значение переменной @car_count.

2.3 Триггеры

Создание таблицы логирования событий:

```
CREATE TABLE event_log (id SERIAL PRIMARY KEY,
event_type VARCHAR(50) NOT NULL,
TABLE_NAME VARCHAR(50) NOT NULL,
record_id INTEGER NOT NULL,
event_time TIMESTAMP NOT NULL);
```

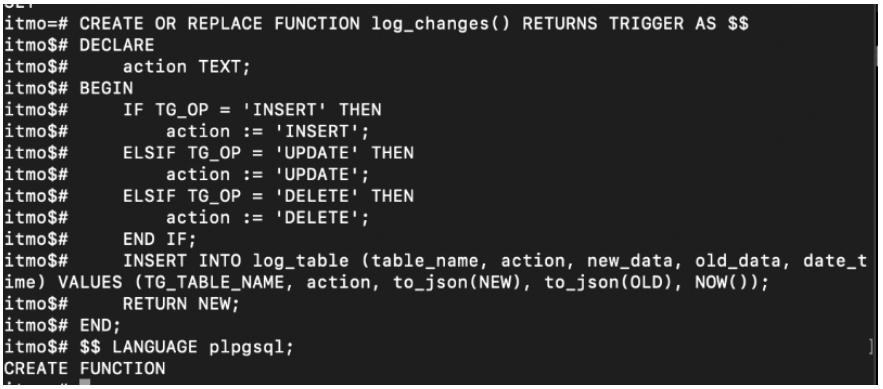
```
itmo=# CREATE TABLE event_log (id SERIAL PRIMARY KEY,
itmo(#                                          event_type VARCHAR(50) NOT NULL
,
itmo(#                                                                  TABLE_NA
ME VARCHAR(50) NOT NULL,
itmo(#
              record_id INTEGER NOT NULL,
itmo(#                                                                          )
                                event_time TIMESTAMP NOT NULL);
CREATE TABLE
```

Указываем схему:

```
SET search_path = rental;
```

Функция, которая будет выполнять логирование:

```
CREATE OR REPLACE FUNCTION log_changes() RETURNS TRIGGER AS $$
DECLARE
    action TEXT;
BEGIN
    IF TG_OP = 'INSERT' THEN
        action := 'INSERT';
    ELSIF TG_OP = 'UPDATE' THEN
        action := 'UPDATE';
    ELSIF TG_OP = 'DELETE' THEN
        action := 'DELETE';
    END IF;
      INSERT INTO event_log (table_name, action, new_data, old_data, date_time) VALUES (TG_TABLE_NAME, action, to_json(NEW), to_json(OLD), NOW());
    RETURN NEW;
END;
$$ LANGUAGE PLPGSQL;
```



Создается функция log_changes(), которая будет вызываться при событиях вставки, удаления и обновления данных. В функции

определяется тип события (INSERT, UPDATE, DELETE) и происходит запись данных в таблицу логирования event_log. Здесь TG_TABLE_NAME - это имя таблицы, которая вызвала триггер. to_json(NEW) и to_json(OLD) используются для преобразования новых и старых значений записей в формат JSON, который затем записывается в таблицу логирования.

Для создания триггера на события вставки, удаления и обновления данных:

```
CREATE TRIGGER log_changes_trigger AFTER
INSERT
OR
UPDATE
OR
DELETE ON admin
FOR EACH ROW EXECUTE FUNCTION log_changes();
```

```
itmo=# CREATE TRIGGER log_changes_trigger AFTER
itmo-# INSERT
itmo-# OR
itmo-# UPDATE
itmo-# OR
itmo-# DELETE ON admin
itmo-# FOR EACH ROW EXECUTE FUNCTION log_changes();
CREATE TRIGGER
```

Триггер будет вызываться после каждого события вставки, удаления и обновления данных в таблицах и записывать данные в таблицу логирования event_log.

3 Выводы

В рамках лабораторной работы стояла задача овладеть практическими навыками создания и использования процедур, функций и триггеров в базе данных PostgreSQL. Для этого были созданы процедуры согласно индивидуальному заданию и и создан триггер для логирования событий вставки, удаления, редактирования данных в базе данных PostgreSQL.

Таким образом, за выполнение данной лабораторной работы удалось овладеть практическими навыками создания и использования процедур, функций и триггеров в базе данных PostgreSQL.