

**Министерство науки и высшего образования Российской Федерации
ФЕДЕРАЛЬНОЕ ГОСУДАРСТВЕННОЕ АВТОНОМНОЕ ОБРАЗОВАТЕЛЬНОЕ
УЧРЕЖДЕНИЕ ВЫСШЕГО ОБРАЗОВАНИЯ
НАЦИОНАЛЬНЫЙ ИССЛЕДОВАТЕЛЬСКИЙ УНИВЕРСИТЕТ ИТМО**

ОТЧЕТ
ПО ЛАБОРАТОРНОЙ РАБОТЕ № 5
«ПРОЦЕДУРЫ, ФУНКЦИИ, ТРИГГЕРЫ В POSTGRESQL»
по дисциплине «Проектирование и реализация баз данных»

Обучающийся Тимаков Егор Павлович
Факультет прикладной информатики
Группа K3241
Направление подготовки 09.03.03 Прикладная информатика
Образовательная программа Мобильные и сетевые технологии 2023
Преподаватель Говорова Марина Михайловна

Санкт-Петербург
2024/2025

Введение

Цель работы.

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

Практическое задание:

1. Создать 3 процедуры для индивидуальной БД согласно варианту (часть 4 ЛР 2). Допустимо использование IN/OUT параметров.
Допустимо создать авторские процедуры. (3 балла)
2. 7 оригинальных триггеров - 7 баллов (max).

Выполнение.

Схема базы данных созданная в postgres ERD изображена на рисунке 1.

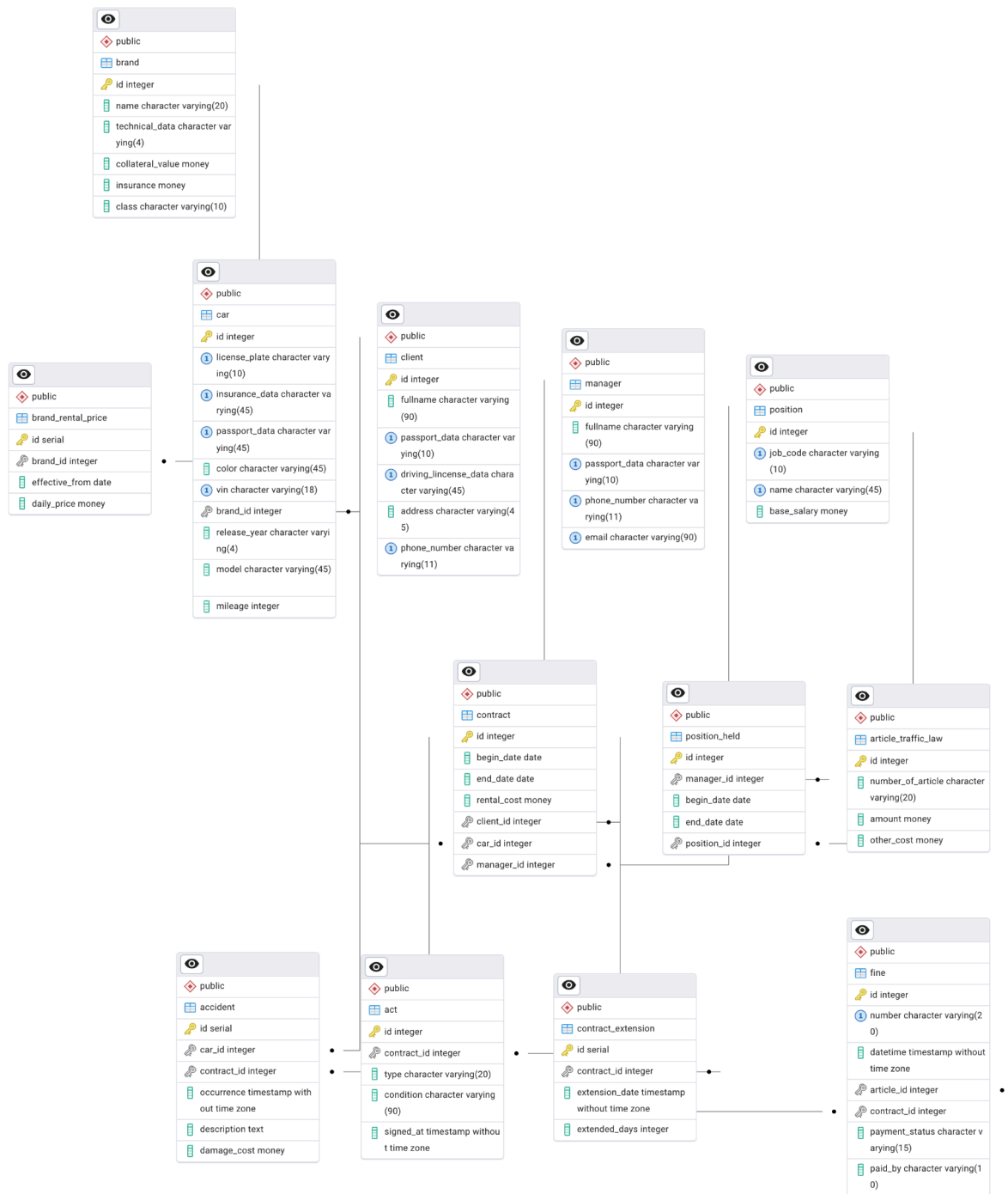


Рисунок 1 - схема базы данных согласно варианту

Далее будем писать процедуры согласно индивидуальному варианту задания.

Задание 1 - Выполнить списание автомобилей, выпущенных ранее заданного года.

SQL-скрипт который создает процедуру:

```
CREATE OR REPLACE PROCEDURE write_off_old_cars(threshold_year
INT)
LANGUAGE plpgsql
```

```

AS $$
BEGIN
    DELETE FROM Car
    WHERE release_year::int < threshold_year;
END;
$$;

```

SQL- скрипт который вызывает процедуру:

```
CALL write_off_old_cars(2020);
```

Таблица Car до выполнения процедуры изображена на рисунке 2.

id	license_plate	insurance_data	passport_data	color	vin	brand_id	release_year	model	mileage
1	A123AA77	1234567890	1111111111	черный	JTDBR32E520123456	1	2020	Corolla	20000
2	A456AA77	2345678901	2222222222	черный	1FAFP34P01W123456	2	2019	Focus	293021
3	A789AA77	3456789012	3333333333	белый	JTDBR32E520123457	1	2025	Corolla	1500
4	A777AA77	1234567891	4444444444	желтый	JTDBR32E520123451	1	2024	SMT	2222
5	A123AA89	1234567894	5555555555	черный	JTDBR32E510123456	1	2014	Corolla	20000

Рисунок 2 - таблица до выполнения процедуры.

Таблица Car после выполнения процедуры изображена на рисунке 3.

id	license_plate	insurance_data	passport_data	color	vin	brand_id	release_year	model	mileage
1	A123AA77	1234567890	1111111111	черный	JTDBR32E520123456	1	2020	Corolla	20000
2	A456AA77	2345678901	2222222222	черный	1FAFP34P01W123456	2	2019	Focus	293021
3	A789AA77	3456789012	3333333333	белый	JTDBR32E520123457	1	2025	Corolla	1500
4	A777AA77	1234567891	4444444444	желтый	JTDBR32E520123451	1	2024	SMT	2222

(4 rows)

Рисунок 3 - таблица после выполнения процедуры.

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

SQL-скрипт который создает процедуру:

```

CREATE OR REPLACE PROCEDURE issue_car_with_discount(
    p_client_id INT,
    p_car_id INT,
    p_manager_id INT,
    p_begin DATE,
    p_end DATE
)
LANGUAGE plpgsql
AS $$
DECLARE
    base_price MONEY;
    discount_factor REAL := 1.0;
    days_rented INT;
BEGIN
    SELECT brp.Daily_Price INTO base_price
    FROM Car c
    JOIN Brand_Rental_Price brp ON c.Brand_id = brp.Brand_ID
    WHERE c.ID = p_car_id
    ORDER BY brp.Effective_From DESC
    LIMIT 1;

```

```

        IF (SELECT COUNT(*) FROM Contract WHERE Client_id =
p_client_id AND End_date < CURRENT_DATE) > 2 THEN
            discount_factor := 0.9;
        END IF;

        days_rented := p_end - p_begin;

        INSERT INTO Contract (Begin_date, End_date, Rental_cost,
Client_id, Car_id, Manager_id)
        VALUES (
            p_begin,
            p_end,
            base_price * days_rented * discount_factor,
            p_client_id,
            p_car_id,
            p_manager_id
        );
    END;
$$;

```

SQL- скрипт который вызывает процедуру:

```

CALL issue_car_with_discount(100, 2, 1, '2025-06-01',
'2025-06-10');

```

Таблица Contract до выполнения процедуры изображена на рисунке 4.

id	begin_date	end_date	rental_cost	client_id	car_id	manager_id	extensions_count
1	2025-05-06	2025-05-11	\$15,000.00	1	1	1	0
3	2025-05-23	2025-05-25	\$25,000.00	1	3	1	0
2	2025-04-03	2025-04-17	\$20,000.00	2	2	1	0
1000	2025-05-09	2025-05-14	\$10,000.00	100	1	1	0
1001	2025-05-15	2025-05-19	\$12,000.00	100	1	1	0

Рисунок 4 - таблица до процедуры

Таблица contract после выполнения процедуры изображена на рисунке 5.

id	begin_date	end_date	rental_cost	client_id	car_id	manager_id	extensions_count
1	2025-05-06	2025-05-11	\$15,000.00	1	1	1	0
3	2025-05-23	2025-05-25	\$25,000.00	1	3	1	0
2	2025-04-03	2025-04-17	\$20,000.00	2	2	1	0
1000	2025-05-09	2025-05-14	\$10,000.00	100	1	1	0
1001	2025-05-15	2025-05-19	\$12,000.00	100	1	1	0
1002	2025-05-30	2025-06-03	\$12,000.00	100	1	1	0

Рисунок 5 - таблица после процедуры.

Задание 3 - Для вычисления количества автомобилей заданной марки.

SQL-скрипт который создает процедуру:

```
CREATE OR REPLACE PROCEDURE count_cars_by_brand(brand_name
TEXT)
LANGUAGE plpgsql
AS $$
DECLARE
    car_count INT;
BEGIN
    SELECT COUNT(*) INTO car_count
    FROM Car c
    JOIN Brand b ON c.Brand_id = b.ID
    WHERE b.name = brand_name;

    RAISE NOTICE 'Количество автомобилей марки %: %',
brand_name, car_count;
END;
$$;
```

SQL- скрипт который вызывает процедуру:

```
CALL count_cars_by_brand('TOYOTA');
```

Результат работы процедуры изображен на рисунке 6.

```
current=# CALL count_cars_by_brand('TOYOTA');
NOTICE:  Количество автомобилей марки TOYOTA: 3
CALL
```

Рисунок 6 - Результат процедуры.

Далее приступим к созданию триггеров:

Первый триггер - Логирование удаления автомобилей.

sql - скрипт создающий триггер и функцию:

```
CREATE OR REPLACE FUNCTION log_car_deletion()
RETURNS TRIGGER AS $$
BEGIN
    INSERT INTO Car_Delete_Log(car_id, reason)
    VALUES (OLD.ID, 'Удаление автомобиля из системы');
    RETURN OLD;
END;
$$ LANGUAGE plpgsql;

CREATE TRIGGER trg_log_car_delete
BEFORE DELETE ON Car
FOR EACH ROW
EXECUTE FUNCTION log_car_deletion();
```

```

CREATE OR REPLACE FUNCTION set_signed_at_if_null()
RETURNS TRIGGER AS $$
BEGIN
    IF NEW.Signed_At IS NULL THEN
        NEW.Signed_At := NOW();
    END IF;
    RETURN NEW;
END;
$$ LANGUAGE plpgsql;

CREATE TRIGGER trg_set_signed_at
BEFORE INSERT ON Act
FOR EACH ROW
EXECUTE FUNCTION set_signed_at_if_null();

```

Таблица car_delete_log до срабатывания триггера изображена на рисунке 7.

```

[carrent=# select * from car_delete_log;
 car_id | deleted_at | reason
-----+-----+-----
(0 rows)

```

Рисунок 7 - таблица логов до срабатывания триггера.

Таблица car_delete_log после срабатывания триггера изображена на рисунке 8.

car_id	deleted_at	reason
4	2025-05-29 10:57:30.570993	Удаление автомобиля из системы

Рисунок 8 - таблица логов после срабатывания триггера.

Триггер 2 - Автоматическая установка даты, если не задана.

sql - скрипт создающий триггер и функцию:

```

CREATE OR REPLACE FUNCTION set_signed_at_if_null()
RETURNS TRIGGER AS $$
BEGIN
    IF NEW.Signed_At IS NULL THEN
        NEW.Signed_At := NOW();
    END IF;
    RETURN NEW;
END;
$$ LANGUAGE plpgsql;

CREATE TRIGGER trg_set_signed_at
BEFORE INSERT ON Act
FOR EACH ROW

```

```
EXECUTE FUNCTION set_signed_at_if_null();
```

Результат работы триггера изображена на рисунке 9.

```
[current=# SELECT * FROM ACT;
 id | contract_id | type  | condition | signed_at
-----+-----+-----+-----+-----
  1 |           1 | Прием | Хорошее  | 2025-05-12 00:00:00
(1 row)

[current=# INSERT INTO ACT (ID, Contract_id, Type,Condition)
[current=# values
[current=# (2,2, 'Прием', 'хорошее')
[current=# ;
INSERT 0 1
[current=# SELECT * FROM ACT;
 id | contract_id | type  | condition | signed_at
-----+-----+-----+-----+-----
  1 |           1 | Прием | Хорошее  | 2025-05-12 00:00:00
  2 |           2 | Прием | хорошее  | 2025-05-29 11:14:39.461515
(2 rows)

current=# █
```

Рисунок 9 - результат срабатывания триггера.

Триггер 3 - Запрет удаления клиента с активным контрактом.

sql - скрипт создающий триггер и функцию:

```
CREATE OR REPLACE FUNCTION prevent_client_delete_if_active()
RETURNS TRIGGER AS $$
BEGIN
    IF EXISTS (SELECT 1 FROM Contract WHERE Client_id = OLD.ID)
    THEN
        RAISE EXCEPTION 'Нельзя удалить клиента с действующими
договорами';
    END IF;
    RETURN OLD;
END;
$$ LANGUAGE plpgsql;

CREATE TRIGGER trg_prevent_client_delete
BEFORE DELETE ON Client
FOR EACH ROW
EXECUTE FUNCTION prevent_client_delete_if_active();
```

Результат работы триггера изображен на рисунке 10.


```

current=# SELECT * FROM client;
 id | fullname | passport_data | driving_license_data | address | phone_number
-----+-----+-----+-----+-----+-----
  1 | Иванов Иван | 1234567890 | 111222333 | Москва | 89161234567
  2 | Петров Петр | 0987654321 | 444555666 | Санкт Петербург | 89161234568
100 | Тестов Тест | 1234567899 | 9876543210 | Тестград | 89161234599
(3 rows)

current=# DELETE FROM client WHERE ID = 100;
ERROR:  не удалил клиента с действующими договорами
CONTEXT:  plpgsql function prevent_client_delete_if_active() line 4 at RAISE
current=#

```

Рисунок 10 - результат работы триггера.

Триггер 4 - Автоматическое обновление пробега после завершения аренды.

sql - скрипт создающий триггер и функцию:

```

CREATE OR REPLACE FUNCTION update_mileage_on_return()
RETURNS TRIGGER AS $$
DECLARE
    rented_days INT;
BEGIN
    IF NEW.Type = 'Прием' THEN
        SELECT End_date - Begin_Date INTO rented_days
        FROM Contract WHERE ID = NEW.Contract_ID;

        UPDATE Car
        SET mileage = mileage + rented_days * 50 -- условный
прирост
        WHERE ID = (SELECT Car_id FROM Contract WHERE ID =
NEW.Contract_ID);
    END IF;
    RETURN NEW;
END;
$$ LANGUAGE plpgsql;

CREATE TRIGGER trg_update_mileage
AFTER INSERT ON Act
FOR EACH ROW
EXECUTE FUNCTION update_mileage_on_return();

```

Таблицу car до срабатывания триггера можно увидеть на рисунке 11.

```

current=# select * from car;
 id | license_plate | insurance_data | passport_data | color | vin | brand_id | release_year | model | mileage
-----+-----+-----+-----+-----+-----+-----+-----+-----+-----
  1 | A123AA77 | 1234567890 | 1111111111 | черный | JTDDBR32E520123456 | 1 | 2020 | Corolla | 20000
  3 | A789AA77 | 3456789012 | 3333333333 | белый | JTDDBR32E520123457 | 1 | 2025 | Corolla | 1500
  2 | A456AA77 | 2345678901 | 2222222222 | черный | 1FAFP34P01W123456 | 2 | 2019 | Focus | 293721
(3 rows)

```

Рисунок 11 - таблица до срабатывания триггера

Таблицу car после срабатывания работы триггера можно увидеть на рисунке 12.

```
INSERT 0 1
[current=# select * from car;
 id | license_plate | insurance_data | passport_data | color | vin | brand_id | release_year | model | mileage
-----+-----+-----+-----+-----+-----+-----+-----+-----+-----
 3 | A789AA77 | 3456789012 | 3333333333 | белый | JTDBR32E520123457 | 1 | 2025 | Corolla | 1500
 2 | A456AA77 | 2345678901 | 2222222222 | черный | 1FAFP34P01W123456 | 2 | 2019 | Focus | 293721
 1 | A123AA77 | 1234567890 | 1111111111 | черный | JTDBR32E520123456 | 1 | 2020 | Corolla | 20250
(3 rows)
```

Рисунок 12 - таблица после срабатывания триггера.

Триггер 5 - Автоматическое добавление акта “Передача” после создания контракта
sql - скрипт создающий триггер и функцию:

```
CREATE OR REPLACE FUNCTION auto_create_transfer_act()
RETURNS TRIGGER AS $$
BEGIN
    INSERT INTO Act (Contract_ID, Type, Condition)
    VALUES (NEW.ID, 'Передача', 'Без повреждений');
    RETURN NEW;
END;
$$ LANGUAGE plpgsql;

CREATE TRIGGER trg_auto_act_on_contract
AFTER INSERT ON Contract
FOR EACH ROW
EXECUTE FUNCTION auto_create_transfer_act();
```

Таблицу act до срабатывания триггера можно увидеть на рисунке 14.

```
[current=# SELECT * FROM ACT
[current=# ;
 id | contract_id | type | condition | signed_at
-----+-----+-----+-----+-----
 1 |          1 | Прием | Хорошее | 2025-05-12 00:00:00
 2 |          2 | Прием | хорошее | 2025-05-29 11:14:39.461515
 3 |         1000 | Прием | хорошее | 2025-05-29 11:36:06.231707
(3 rows)
```

Рисунок 14 - таблица до срабатывания триггера

Таблицу act после срабатывания работы триггера можно увидеть на рисунке 15.

```
[current=# select * from act;
 id | contract_id | type | condition | signed_at
-----+-----+-----+-----+-----
 1 |          1 | Прием | Хорошее | 2025-05-12 00:00:00
 2 |          2 | Прием | хорошее | 2025-05-29 11:14:39.461515
 3 |         1000 | Прием | хорошее | 2025-05-29 11:36:06.231707
 4 |         1123 | Передача | Без повреждений | 2025-05-29 11:51:19.597818
(4 rows)
```

Рисунок 15 - таблица после срабатывания триггера

Триггер 6 - Обновление количества продлений у договора

SQL - скрипт создающий триггер и функцию:

```
CREATE OR REPLACE FUNCTION update_extensions_counter()
RETURNS TRIGGER AS $$
```

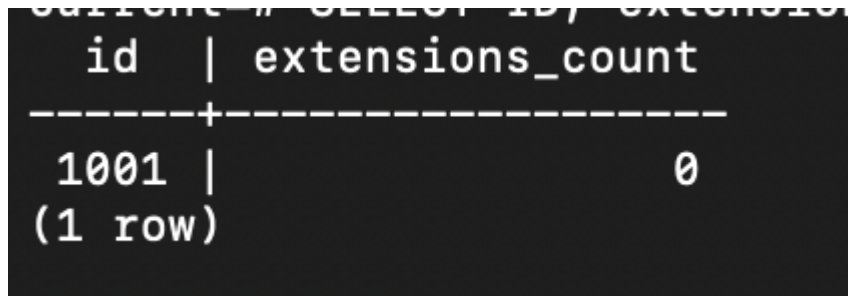
```

BEGIN
    UPDATE Contract
    SET extensions_count = COALESCE(extensions_count, 0) + 1
    WHERE ID = NEW.Contract_ID;
    RETURN NEW;
END;
$$ LANGUAGE plpgsql;

CREATE TRIGGER trg_count_extensions
AFTER INSERT ON Contract_Extension
FOR EACH ROW
EXECUTE FUNCTION update_extensions_counter();

```

таблицу contract до срабатывания триггера можно увидеть на рисунке 14.

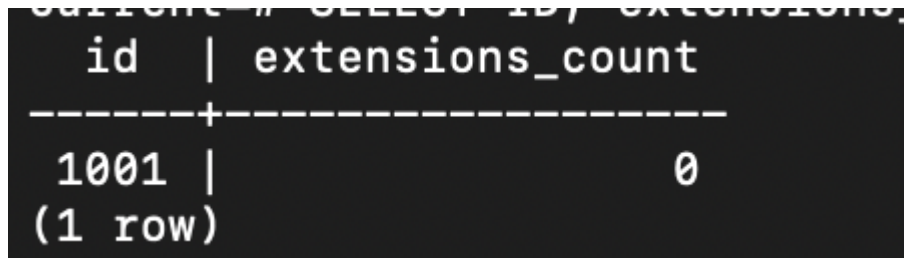


id	extensions_count
1001	0

(1 row)

Рисунок 16 - таблица до срабатывания триггера

Таблицу contract после срабатывания работы триггера можно увидеть на рисунке 16.



id	extensions_count
1001	0

(1 row)

Рисунок 16 - таблица после срабатывания триггера

Триггер 7 - Автоматическая проверка пробега авто;

SQL - скрипт создающий триггер и функцию:

```

CREATE OR REPLACE FUNCTION check_mileage_reasonable()
RETURNS TRIGGER AS $$
DECLARE
    car_age INT;
    mileage_limit INT;
BEGIN
    IF NEW.release_year IS NULL OR NEW.mileage IS NULL THEN
        RETURN NEW;
    END IF;

```

```

    car_age := EXTRACT(YEAR FROM CURRENT_DATE)::INT -
NEW.release_year::INT + 1;
    mileage_limit := car_age * 100000;

    IF NEW.mileage > mileage_limit THEN
        RAISE EXCEPTION 'Пробег (% км) превышает допустимый для
% лет эксплуатации: максимум % км',
        NEW.mileage, car_age, mileage_limit;
    END IF;

    RETURN NEW;
END;
$$ LANGUAGE plpgsql;

DROP TRIGGER IF EXISTS trg_check_mileage ON Car;

CREATE TRIGGER trg_check_mileage
BEFORE INSERT OR UPDATE ON Car
FOR EACH ROW
EXECUTE FUNCTION check_mileage_reasonable();

```

Результат работы триггера можно увидеть на рисунке 17.

```

current=# INSERT INTO Car (ID, License_plate, Insurance_data, Passport_data, Color, VIN,Brand_id, release_year, mileage, model) VALUES (997,'X222XX77','5100','5555555556','Красный','VINTEST123456787',1,2023,4)
000000,'Yaris');
ERROR:  Пробег (400000 км) превышает допустимый для 3 лет эксплуатации: максимум 300000 км
CONTEXT:  PL/pgSQL function check_mileage_reasonable() line 14 at RAISE

```

Рисунок 17 - результат работы триггера

Дополнительное задание: