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: Богданов Максим Александрович
Факультет прикладной информатики
Группа K3240
Направление подготовки 09.03.03 Прикладная информатика
Образовательная программа Мобильные и сетевые технологии 2023
Преподаватель Говорова Марина Михайловна

Санкт-Петербург
2025

Цель работы:

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

Программное обеспечение:

СУБД PostgreSQL 1X, SQL Shell (psql)

Практическое задание

1. Создать 3 процедуры для индивидуальной БД согласно варианту (часть 4 ЛР 2). Допустимо использование IN/OUT параметров. Допустимо создать авторские процедуры.
2. Создать триггеры для индивидуальной БД согласно варианту:

   7 оригинальных триггеров - 7 баллов (max).Изучить графическое представление запросов и просмотреть историю запросов.

Схема базы данных

Модель описывает систему управления такси или автопарком с возможностью учета сотрудников, машин, заказов и клиентов. Ознакомиться с моделью можно на рисунке 1.

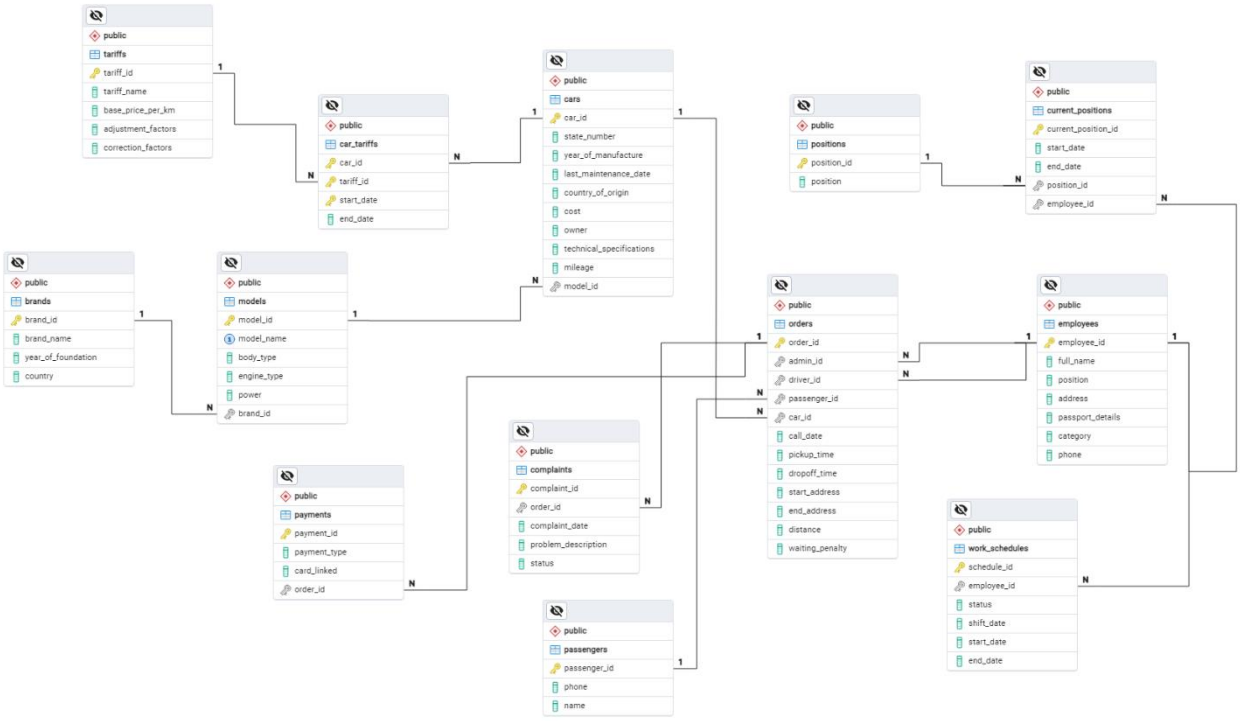

Рисунок 1 – Схема базы данных

Ход работы

Задание 1. Создать 3 процедуры для индивидуальной БД согласно варианту.

Было решено добавить 2 новых таблицы в БД для записи логов:

```
CREATE TABLE order_log (
    log_id SERIAL PRIMARY KEY,
    order_id INT,
    action VARCHAR(50),
    log_timestamp TIMESTAMP DEFAULT CURRENT_TIMESTAMP,
    details TEXT
);

CREATE TABLE complaint_status_log (
    log_id SERIAL PRIMARY KEY,
    complaint_id INT,
    old_status VARCHAR(20),
    new_status VARCHAR(20),
    log_timestamp TIMESTAMP DEFAULT CURRENT_TIMESTAMP
);
```

```
taxi_service=# \dt
                  Список отношений
 Схема  |        Им\xFF         |  Ти  |   Владелец
--------+-----------------------+------+----------
 public | brands                | таблица | postgres
 public | car_tariffs           | таблица | postgres
 public | cars                  | таблица | postgres
 public | complaint_status_log  | таблица | postgres
 public | complaints            | таблица | postgres
 public | current_positions     | таблица | postgres
 public | employees             | таблица | postgres
 public | models                | таблица | postgres
 public | order_log             | таблица | postgres
 public | orders                | таблица | postgres
 public | passengers            | таблица | postgres
 public | payments              | таблица | postgres
 public | positions             | таблица | postgres
 public | tariffs               | таблица | postgres
 public | work_schedules        | таблица | postgres
(15 строк)
```

Были разработаны три хранимые процедуры на языке PL/pgSQL, соответствующие заданию. Процедуры используют параметры IN и OUT, включают обработку ошибок и возвращают результаты через курсоры (где применимо).

Процедура 1: Добавление нового пассажира

```
CREATE OR REPLACE PROCEDURE add_passenger(
    p_phone VARCHAR(11),
    p_name VARCHAR(100)
)
LANGUAGE plpgsql
AS $$
BEGIN
    INSERT INTO passengers (phone, name)
    VALUES (p_phone, p_name);

    IF NOT FOUND THEN
        RAISE EXCEPTION 'Ошибка при добавлении пассажира';
    END IF;
END;
$$;
```

Эта процедура добавляет нового пассажира в таблицу passengers.

Результат:

```
CREATE PROCEDURE
taxi_service=# CALL add_passenger('79991234567', 'Иван Иванов');
CALL
taxi_service=#
```

```
taxi_service=# SELECT * FROM passengers;
 passenger_id |    phone    |              name
--------------+-------------+--------------------------------
            1 | 79001234567 | Смирнова Анна Ивановна
            2 | 79009876543 | Кузнецов Михаил Сергеевич
            3 | 79005551122 | Петрова Елена Викторовна
            4 | 79007778899 | Иванов Сергей Александрович
            5 | 79003332211 | Соколова Мария Дмитриевна
            6 | 79004443322 | Лебедев Антон Павлович
            7 | 79006667788 | Морозова Ольга Николаевна
            8 | 79008889900 | Васильев Олег Игоревич
            9 | 79001122334 | Фёдорова Светлана Андреевна
           10 | 79009988776 | Николаев Павел Константинович
           11 | 79002233445 | Ковалёв Дмитрий Викторович
           12 | 79003344556 | Зайцева Екатерина Павловна
           13 | 12345678901 | Иван Иванов
           14 | 98765432102 | Мария Петрова
           15 | 55555555555 | Алексей Сидоров
           16 | 11122233344 | Елена Смирнова
           17 | 44455566677 | Дмитрий Кузнецов
           18 | 79991234567 | €ў - €ў -®ў
           19 | 79991234567 | Иван Иванов
(19 строк)
```

Процедура 2: Завершение поездки и расчёт стоимости

```
CREATE OR REPLACE PROCEDURE complete_order(
    p_order_id INT,
    p_dropoff_time TIMESTAMP
)
LANGUAGE plpgsql
AS $$
DECLARE
    v_distance DECIMAL(10,2);
    v_base_price DECIMAL(10,2);
    v_cost DECIMAL(10,2);
BEGIN
    SELECT distance INTO v_distance
    FROM orders
    WHERE order_id = p_order_id;

    SELECT t.base_price_per_km INTO v_base_price
    FROM orders o
    JOIN car_tariffs ct ON o.car_id = ct.car_id
    JOIN tariffs t ON ct.tariff_id = t.tariff_id
    WHERE o.order_id = p_order_id
    AND ct.end_date IS NULL
    LIMIT 1;

    IF v_distance IS NULL THEN
        RAISE EXCEPTION 'Заказ с ID % не найден', p_order_id;
    END IF;
    IF v_base_price IS NULL THEN
        v_base_price = 5.00; -- по умолчанию
    END IF;

    v_cost = v_distance * v_base_price;

    UPDATE orders
    SET dropoff_time = p_dropoff_time,
        waiting_penalty = v_
    WHERE order_id = p_order_id;

    IF NOT FOUND THEN
        RAISE EXCEPTION 'Ошибка при завершении заказа с ID %', p_order_id;
    END IF;
END;
$$;
```

Эта процедура завершает поездку, обновляя dropoff_time в таблице orders, и рассчитывает стоимость поездки на основе тарифа и расстояния.

Результат:

```
taxi_service=# INSERT INTO orders (order_id, car_id, distance, call_date, pickup_time) VALUES
INSERT 0 1
taxi_service=# CALL complete_order(44, '2025-05-15 13:30:00');
CALL
taxi_service=# SELECT dropoff_time, waiting_penalty FROM orders WHERE order_id = 44;
    dropoff_time     | waiting_penalty
---------------------+-----------------
 2025-05-15 13:30:00 |           27.50
(1 строка)
```

Процедура 3: Регистрация оплаты за заказ

```
CREATE OR REPLACE PROCEDURE register_payment(
    p_order_id INT,
    p_payment_type VARCHAR(20),
    p_card_linked BOOLEAN
)
LANGUAGE plpgsql
AS $$
BEGIN
    INSERT INTO payments (payment_type, card_linked, order_id)
    VALUES (p_payment_type, p_card_linked, p_order_id);

    IF NOT FOUND THEN
        RAISE EXCEPTION 'Ошибка при регистрации оплаты для заказа с ID %',
p_order_id;
    END IF;

    INSERT INTO order_log (order_id, action, details)
    VALUES (p_order_id, 'PAYMENT', 'Оплата заказа произведена через ' ||
p_payment_type);
END;
$$;
```

Эта процедура добавляет запись об оплате заказа в таблицу payments, фиксируя тип оплаты (например, "cash" или "card") и информацию о привязке карты.

Результат:

```
taxi_service=# CALL register_payment(1, 'card', TRUE);
CALL
taxi_service=# SELECT * FROM payments WHERE order_id = 1;
 payment_id | payment_type | card_linked | order_id
------------+--------------+-------------+----------
          1 | Наличные     | f           |        1
         35 | card         | t           |        1
(2 строки)


taxi_service=# SELECT * FROM order_log WHERE order_id = 1 AND action = 'PAYMENT';
 log_id | order_id | action  |       log_timestamp        |               details
--------+----------+---------+----------------------------+--------------------------------------
      5 |        1 | PAYMENT | 2025-05-15 14:31:24.779214 | Оплата заказа произведена через card
(1 строка)


taxi_service=#
```

Задание 2. Создание триггеров.

Необходимо придумать и создать 7 триггеров для базы данных.

Триггер 1: Логирование создания заказа

```
CREATE OR REPLACE FUNCTION log_order_insertion()
RETURNS TRIGGER AS $$
BEGIN
    INSERT INTO order_log (order_id, action, details)
    VALUES (NEW.order_id, 'INSERT', 'Создан новый заказ для пассажира ' ||
NEW.passenger_id);
    RETURN NEW;
END;
$$ LANGUAGE plpgsql;

CREATE TRIGGER order_insert_trigger
AFTER INSERT ON orders
FOR EACH ROW
EXECUTE FUNCTION log_order_insertion();
```

Результат:

```
taxi_service=# INSERT INTO orders (admin_id, driver_id, passenger_id, car_id, call_date, pickup_ti
t_address, end_address, distance)
taxi_service-# VALUES (1, 2, 3, 2, '2025-05-14 20:00:00', '2025-05-14 20:15:00', '2025-05-14 20:45
етровская 10', 6.0);
INSERT 0 1
taxi_service=# SELECT * FROM order_log;
 log_id | order_id | action |       log_timestamp        |              details
--------+----------+--------+----------------------------+-----------------------------------
      1 |       40 | INSERT | 2025-05-15 01:09:15.531931 | Создан новый заказ для пассажира 3
(1 строка)
```

Триггер 2: Проверка дат заказа

```
CREATE OR REPLACE FUNCTION check_order_dates()
RETURNS TRIGGER AS $$
BEGIN
    IF NEW.pickup_time < NEW.call_date THEN
        RAISE EXCEPTION 'Время посадки не может быть раньше времени вызова';
    END IF;
    RETURN NEW;
END;
$$ LANGUAGE plpgsql;

CREATE TRIGGER order_dates_trigger
BEFORE INSERT OR UPDATE ON orders
FOR EACH ROW
EXECUTE FUNCTION check_order_dates();
```

Результат:

```
taxi_service=# INSERT INTO orders (admin_id, driver_id, passenger_id, car_id, call_dat
t_address, end_address, distance)
taxi_service-# VALUES (1, 1, 1, 1, '2025-05-14 22:00:00', '2025-05-14 21:00:00', '2025
ра 50', 5.0);
ОШИБКА:  Время посадки не может быть раньше времени вызова
КОНТЕКСТ:  функция PL/pgSQL check_order_dates(), строка 4, оператор RAISE
taxi_service=#
```

Триггер 3: Обновление пробега автомобиля

```sql
CREATE OR REPLACE FUNCTION update_car_mileage()
RETURNS TRIGGER AS $$
BEGIN
    UPDATE cars
    SET mileage = mileage + NEW.distance
    WHERE car_id = NEW.car_id;
    RETURN NEW;
END;
$$ LANGUAGE plpgsql;

CREATE TRIGGER car_mileage_trigger
AFTER INSERT ON orders
FOR EACH ROW
WHEN (NEW.distance IS NOT NULL)
EXECUTE FUNCTION update_car_mileage();
```

Результат:

```
taxi_service=# SELECT mileage FROM cars WHERE car_id = 1;
 mileage
---------
  300000
(1 строка)


taxi_service=# INSERT INTO orders (admin_id, driver_id, passenger_id, car_id, call_date,
t_address, end_address, distance)
taxi_service-# VALUES (1, 1, 1, 1, '2025-05-14 22:00:00', '2025-05-14 22:15:00', '2025-0
ра 50', 5.0);
INSERT 0 1
taxi_service=# SELECT mileage FROM cars WHERE car_id = 1;
 mileage
---------
  300005
(1 строка)
```

Триггер 4: Проверка пересечения тарифов

```
CREATE OR REPLACE FUNCTION check_tariff_overlap()
RETURNS TRIGGER AS $$
BEGIN
    IF EXISTS (
        SELECT 1
        FROM car_tariffs
        WHERE car_id = NEW.car_id
        AND start_date <= COALESCE(NEW.end_date, CURRENT_TIMESTAMP)
        AND (end_date IS NULL OR end_date >= NEW.start_date)
        AND (NEW.end_date IS NULL OR NEW.start_date <= end_date)
        AND (start_date != NEW.start_date OR tariff_id != NEW.tariff_id)
    ) THEN
        RAISE EXCEPTION 'Пересечение периодов тарифов для автомобиля %',
NEW.car_id;
    END IF;
    RETURN NEW;
END;
$$ LANGUAGE plpgsql;

CREATE TRIGGER tariff_overlap_trigger
BEFORE INSERT OR UPDATE ON car_tariffs
FOR EACH ROW
EXECUTE FUNCTION check_tariff_overlap();
```

Результат:

```
taxi_service=# INSERT INTO car_tariffs (car_id, tariff_id, start_date, end_date) VALUES (1, 2,
');
ОШИБКА:  Пересечение периодов тарифов для автомобиля 1
КОНТЕКСТ:  функция PL/pgSQL check_tariff_overlap(), строка 12, оператор RAISE
taxi_service=#
```

Триггер 5: Логирование изменения статуса жалобы

```
CREATE OR REPLACE FUNCTION log_complaint_status()
RETURNS TRIGGER AS $$
BEGIN
    IF OLD.status != NEW.status THEN
        INSERT INTO complaint_status_log (complaint_id, old_status, new_status)
        VALUES (NEW.complaint_id, OLD.status, NEW.status);
    END IF;
    RETURN NEW;
END;
$$ LANGUAGE plpgsql;

CREATE TRIGGER complaint_status_trigger
AFTER UPDATE ON complaints
FOR EACH ROW
EXECUTE FUNCTION log_complaint_status();
```

Результат:

```
taxi_service=#
taxi_service=# UPDATE complaints SET status = 'Закрыта' WHERE complaint_id = 1;
UPDATE 1
taxi_service=# SELECT * FROM complaint_status_log;
 log_id | complaint_id | old_status | new_status |       log_timestamp
--------+--------------+------------+------------+----------------------------
      1 |            1 | Открыта    | Закрыта    | 2025-05-15 01:17:39.183099
(1 строка)
```

Триггер 6: Проверка категории водителя

```
CREATE OR REPLACE FUNCTION check_driver_category()
RETURNS TRIGGER AS $$
BEGIN
    IF NOT EXISTS (
        SELECT 1
        FROM employees
        WHERE employee_id = NEW.driver_id
        AND category = 'B'
    ) THEN
        RAISE EXCEPTION 'Водитель % не имеет категорию B', NEW.driver_id;
    END IF;
    RETURN NEW;
END;
$$ LANGUAGE plpgsql;

CREATE TRIGGER driver_category_trigger
BEFORE INSERT OR UPDATE ON orders
FOR EACH ROW
WHEN (NEW.driver_id IS NOT NULL)
EXECUTE FUNCTION check_driver_category();
```

Результат:

```
taxi_service=# INSERT INTO orders (admin_id, driver_id, passenger_id, car_id, call_dat
t_address, end_address, distance)
taxi_service-# VALUES (1, 3, 1, 1, '2025-05-14 23:00:00', '2025-05-14 23:15:00', '2025
pa 60', 4.0);
ОШИБКА:  Водитель 3 не имеет категорию B
КОНТЕКСТ:  функция PL/pgSQL check_driver_category(), строка 9, оператор RAISE
taxi_service=#
```

Триггер 7: Запрет обновления прошедших смен

```
CREATE OR REPLACE FUNCTION prevent_past_schedule_update()
RETURNS TRIGGER AS $$
BEGIN
    IF NEW.shift_date < CURRENT_DATE THEN
        RAISE EXCEPTION 'Нельзя обновлять расписание для прошедшей даты %',
NEW.shift_date;
    END IF;
    RETURN NEW;
END;
$$ LANGUAGE plpgsql;

CREATE TRIGGER schedule_update_trigger
BEFORE UPDATE ON work_schedules
FOR EACH ROW
EXECUTE FUNCTION prevent_past_schedule_update();
```

Результат:

```
taxi_service=#
taxi_service=# UPDATE work_schedules SET status = 'Неактивна' WHERE shift_date = '2025-05-15';
UPDATE 1
taxi_service=# UPDATE work_schedules SET status = 'Неактивна' WHERE shift_date = '2025-05-10';
ОШИБКА:  Нельзя обновлять расписание для прошедшей даты 2025-05-10
КОНТЕКСТ:  функция PL/pgSQL prevent_past_schedule_update(), строка 4, оператор RAISE
taxi_service=#
```

Задание 3 Доп задание.

Требовалось исправить триггер, который был дан во время выполнения практической работы

```
CREATE OR REPLACE FUNCTION fn_check_time_punch()
RETURNS TRIGGER AS $$
DECLARE
    last_punch RECORD;
BEGIN
    -- Поиск последней записи для сотрудника
    SELECT is_out_punch, punch_time
    INTO last_punch
    FROM time_punch
    WHERE employee_id = NEW.employee_id
    ORDER BY id DESC
    LIMIT 1;

    -- Если это первая запись для сотрудника, разрешаем вставку
    IF last_punch IS NULL THEN
        RETURN NEW;
    END IF;
```

```
    -- Проверка, что новое время больше предыдущего
    IF NEW.punch_time <= last_punch.punch_time THEN
        RETURN NULL;
    END IF;

    -- Проверка, что действия вход/выход не повторяются
    IF NEW.is_out_punch = last_punch.is_out_punch THEN
        RETURN NULL;
    END IF;

    RETURN NEW;
END;
$$ LANGUAGE plpgsql;
```

1. **Проверка времени**:

   Добавлена проверка NEW.punch_time <= last_punch.punch_time. Если новое время меньше или равно предыдущему, триггер возвращает NULL, и вставка отменяется.
2. **Проверка повторяющихся действий**:

   Сохранена исходная логика: если NEW.is_out_punch совпадает с последним is_out_punch для того же сотрудника, вставка отменяется (RETURN NULL).
3. **Обработка первой записи**:

   Если для сотрудника нет предыдущих записей (last_punch IS NULL), вставка разрешается (RETURN NEW).

Вывод

В процессе выполнения лабораторной работы были освоены навыки разработки и использования функций и триггеров в СУБД PostgreSQL. Созданы пользовательские функции для обработки данных и триггеры, обеспечивающие контроль целостности информации. Также был доработан триггер учёта входа/выхода сотрудников, в котором добавлены проверки на последовательность действий и корректность времени, что улучшило точность фиксации рабочего времени.