МИНИСТЕРСТВО НАУКИ И ВЫСШЕГО ОБРАЗОВАНИЯ
РОССИЙСКОЙ ФЕДЕРАЦИИ

Федеральное государственное автономное
образовательное учреждение высшего образования
«Национальный исследовательский университет ИТМО»
(Университет ИТМО)

ОТЧЕТ
по лабораторной работе № 3.2
Проектирование и реализация баз данных

СОЗДАНИЕ ТАБЛИЦ БАЗЫ ДАННЫХ POSTGRESQL. ЗАПОЛНЕНИЕ ТАБЛИЦ РАБОЧИМИ ДАННЫМИ

Студент:
Группа № 436209 *А.А. Цырульников*

Предподаватель:
Преподаватель практики *М.М. Говорова*

Санкт-Петербург 2025

СОДЕРЖАНИЕ

Стр.

ВВЕДЕНИЕ

Цель работы: овладеть практическими навыками создания и использования процедур, функций и триггеров в базе данных PostgreSQL.

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. Создать триггеры для индивидуальной БД согласно варианту:
   - Вариант 2.1. 3 триггера - 3 балла (min). Допустимо использовать триггеры логирования из практического занятия по функциям и триггерам.
   - Вариант 2.2. 7 оригинальных триггеров - 7 баллов (max).

Индивидуальное задание:

Вариант 14. База данных «Служба заказа такси»

Описание предметной области: Система предназначена для регистрации всех вызовов такси и их распределения между водителями. Каждый водитель ежедневно получает заработную плату, зависящую от количества выполненных вызовов и их тарифов

Заказы принимает дежурный администратор и передает их водителю. В заказе указывается способ оплаты — наличными или онлайн. Если оплата 3 производится онлайн, система хранит информацию о карте, с которой была совершена транзакция. Необходимо хранить график работы водителей в системе.

Ежедневно действуют базовые тарифы на тип предоставляемых автомобилей, но стоимость может корректироваться в зависимости от времени суток и дорожной ситуации.

База данных должна включать следующий минимальный набор информации:

- Идентификатор сотрудника, ФИО, адрес, номер телефона, паспортные данные, должность, категория

- Модель и марка автомобиля, технические характеристики, страна-производитель, стоимость
- Код тарифа, название тарифа, цена за километр
- Идентификатор автомобиля, государственный номер, год выпуска, пробег, дата последнего ТО
- Дата вызова, время посадки/высадки пассажира, номер телефона пассажира
- Место отправления/назначения, расстояние, штраф за ожидание (мин)
- Способ оплаты (онлайн/наличные), жалобы клиента

Дополните список атрибутов на основе детального анализа предметной области.

Выполните инфологическое моделирование базы данных системы (самостоятельно определите ограничения).

Создайте логическую модель базы данных, используя инфологическую модель (задание 1.1). Примените необходимые средства обеспечения целостности данных в СУБД.

Дополните атрибуты и сущности на основе анализа предметной области и требований к базе данных.

1 Ход работы

Инфологическая модель представлена на рисунке 1.1.

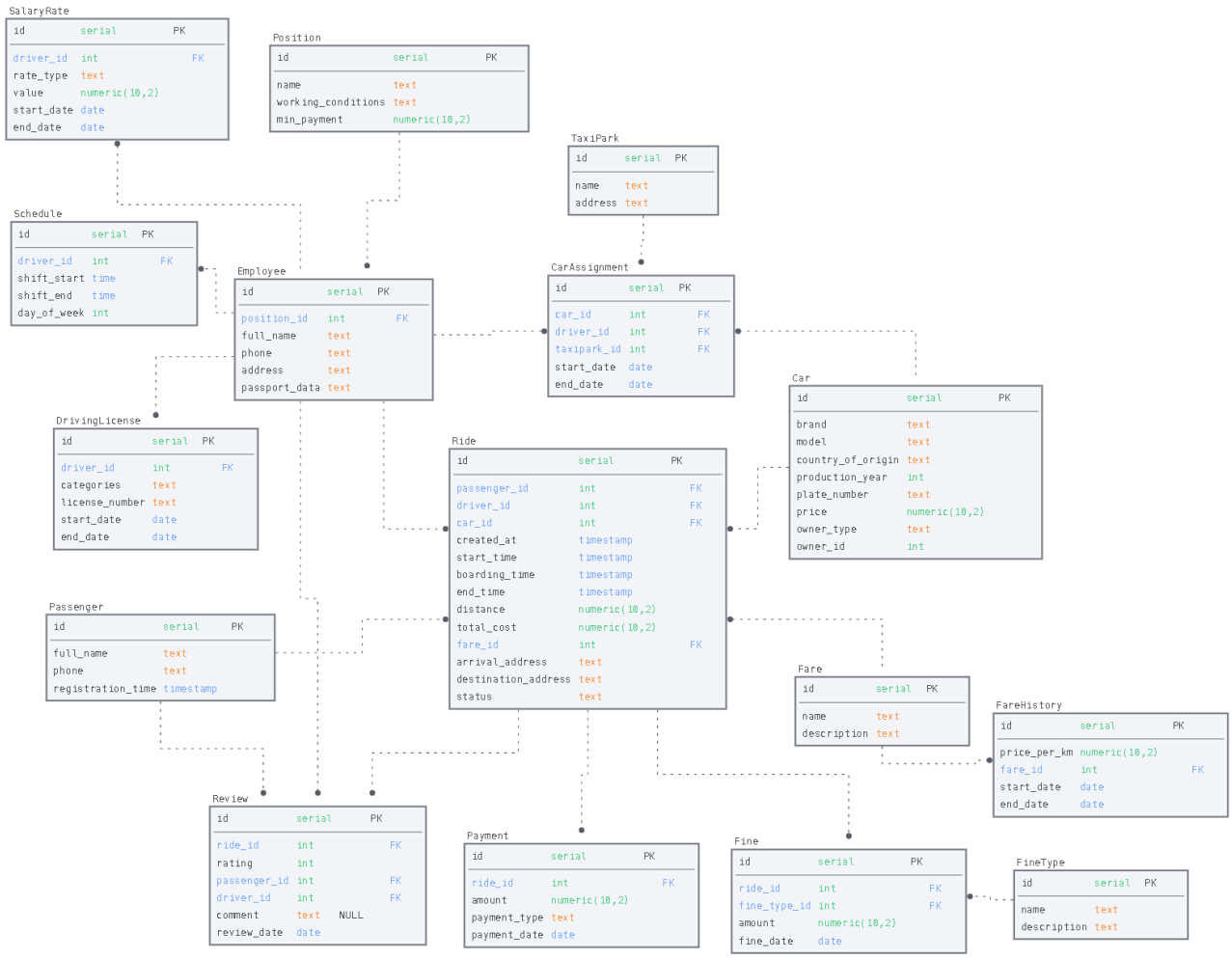

Рисунок 1.1 — Инфологическая модель

ERD схема представлена на рисунке 1.2.

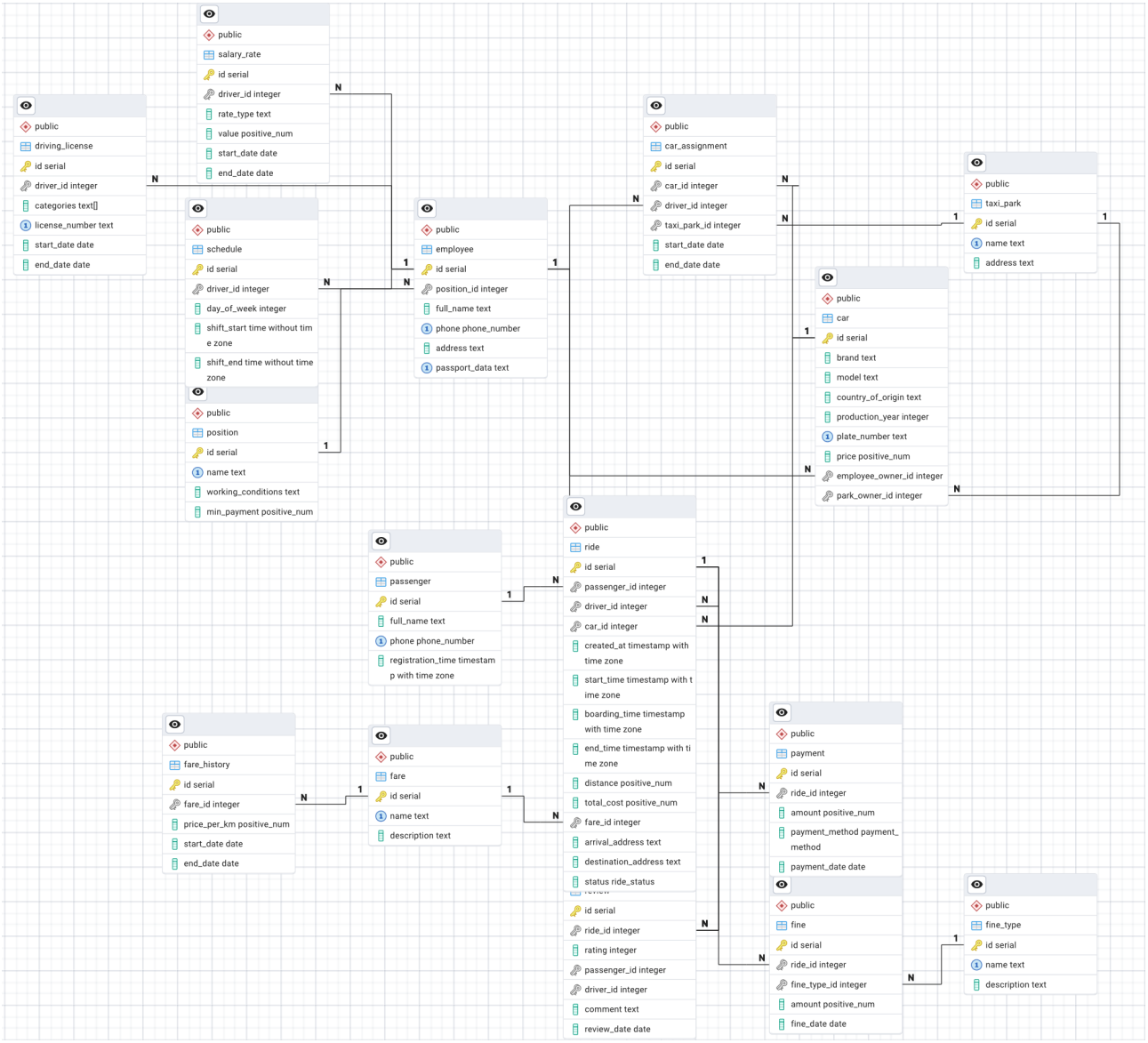

Рисунок 1.2 — ERD схема

1.1 Процедуры

1.1.1 Процедура для вывода данных о пассажирах, заказывавших такси в заданном временном интервале

```
CREATE OR REPLACE FUNCTION get_passengers_by_period(
    start_date TIMESTAMP,
    end_date   TIMESTAMP
)
RETURNS TABLE(
    passenger_name TEXT,
```

```
7         phone           TEXT,
8         ride_count      BIGINT,
9         total_amount    NUMERIC
10    )
11    LANGUAGE sql
12    AS $$
13        SELECT
14            p.full_name    AS passenger_name,
15            p.phone,
16            COUNT(r.id)    AS ride_count,
17            COALESCE(SUM(r.total_cost), 0) AS total_amount
18        FROM passenger p
19        JOIN ride r
20          ON r.passenger_id = p.id
21         AND r.start_time BETWEEN $1 AND $2
22        GROUP BY p.full_name, p.phone
23        ORDER BY p.full_name;
24    $$;
```

Результат выполнения процедуры:

```
lab5=# SELECT * FROM get_passengers_by_period(
  '2023-01-01'::date,
  '2025-05-29'::date
);
       passenger_name       |    phone     | ride_count | total_amount
----------------------------+--------------+------------+--------------
 Бабушкина Татьяна Юрьевна  | +79002225566 |          1 |       252.00
 Воробьёв Иван Сергеевич    | +79001114455 |          2 |      1661.50
 Киселёв Олег Николаевич    | +79003336677 |          1 |        20.00
 Кузнецова Ольга Николаевна | +79002223344 |          1 |       122.40
 Морозов Дмитрий Сергеевич  | +79003334455 |          1 |       156.00
 Новиков Алексей Николаевич | +79005556677 |          1 |       502.00
 Петров Петр Петрович       | +79001112233 |          1 |       168.00
 Смирнова Анна Владимировна | +79004445566 |          1 |        84.00
(8 rows)
```

Рисунок 1.3 — Результат выполнения процедуры 1

1.1.2 Процедура для вывода данных о поездках пассажира по номеру телефона

```
1     CREATE OR REPLACE FUNCTION get_rides_by_phone(
2         passenger_phone TEXT
```

```
    )
    RETURNS TABLE(
        ride_date       TIMESTAMPTZ,
        from_address    TEXT,
        to_address      TEXT,
        distance        NUMERIC,
        cost            NUMERIC,
        driver_name     TEXT,
        status          TEXT
    )
    LANGUAGE sql
    AS $$
        SELECT
            r.start_time        AS ride_date,
            r.arrival_address   AS from_address,
            r.destination_address AS to_address,
            r.distance,
            r.total_cost        AS cost,
            e.full_name         AS driver_name,
            r.status::TEXT      AS status
        FROM passenger p
        JOIN ride     r ON r.passenger_id = p.id
        JOIN employee e ON r.driver_id    = e.id
        WHERE p.phone = $1
        ORDER BY r.start_time DESC;
    $$;
```

Результат выполнения процедуры:

```
lab5=# SELECT * FROM get_rides_by_phone('+79001112233');
          ride_date          |  from_address  |     to_address      | distance |  cost  |      driver_name      |  status
-----------------------------+----------------+---------------------+----------+--------+-----------------------+-----------
 2025-05-29 04:51:55.431603+00 | ул. Ленина, 10 | ул. Гагарина, 5     |    10.00 | 200.00 | Иванов Иван Иванович  | completed
 2025-05-28 20:51:55.431603+00 | пр. Ленина, 14 | ул. Маяковского, 23 |     8.40 | 168.00 | Смирнов Иван Петрович | completed
(2 rows)
```

Рисунок 1.4 — Результат выполнения процедуры 2

1.1.3 Процедура для вычисления суммарного дохода таксопарка за истекший месяц

```
    CREATE OR REPLACE FUNCTION get_taxi_park_revenue()
    RETURNS TABLE(
        park_name TEXT,
        revenue   NUMERIC
    )
```

```
    LANGUAGE sql
    AS $$
        SELECT
            tp.name         AS park_name,
            COALESCE(SUM(pay.amount), 0) AS revenue
        FROM taxi_park tp
        LEFT JOIN car_assignment ca ON ca.taxi_park_id = tp.id
        LEFT JOIN ride           r    ON r.car_id = ca.car_id
                                     AND r.start_time::DATE BETWEEN ca.start_
    date AND ca.end_date
        LEFT JOIN payment        pay ON pay.ride_id = r.id
        WHERE pay.payment_date >= DATE_TRUNC('month', CURRENT_DATE -
    INTERVAL '1 month')
          AND pay.payment_date <  DATE_TRUNC('month', CURRENT_DATE)
        GROUP BY tp.name
        ORDER BY revenue DESC;
    $$;
```

Результат выполнения процедуры:

```
lab5=# SELECT * FROM get_taxi_park_revenue();
   park_name   | revenue
---------------+---------
 Такси Лиса    | 1753.50
 Такси Медведь | 1029.40
 Такси Волк    |  537.00
 Такси Гепард  |  374.40
(4 rows)
```

Рисунок 1.5 — Результат выполнения процедуры 3

1.2 Триггеры

1.2.1 Автоматический расчёт total_cost при вставке в ride

```
    CREATE OR REPLACE FUNCTION calc_total_cost() RETURNS TRIGGER AS $$
BEGIN
  NEW.total_cost := (
```

```
    SELECT price_per_km
      FROM fare_history
     WHERE fare_id = NEW.fare_id
       AND NEW.start_time::DATE BETWEEN start_date AND end_date
  ) * NEW.distance;
  RETURN NEW;
END;
$$ LANGUAGE plpgsql;

CREATE TRIGGER trg_calc_total_cost
BEFORE INSERT ON ride
FOR EACH ROW
EXECUTE PROCEDURE calc_total_cost();
```

Проверка триггера:

```
lab5=# INSERT INTO ride (
lab5(#   passenger_id, driver_id, car_id,
lab5(#   created_at, start_time, boarding_time, end_time,
lab5(#   distance, fare_id, arrival_address, destination_address, status
lab5(# ) VALUES (
lab5(#   1,1,1,
lab5(#   now(), now(), now()+interval '5 min', now()+interval '20 min',
lab5(#   7.2, 1, 'Точка А','Точка Б','completed'
lab5(# );
INSERT 0 1
```

Рисунок 1.6 — Пример вставки в таблицу ride

```
lab5=# SELECT
lab5-#   r.id,
lab5-#   r.distance,
lab5-#   r.total_cost,
lab5-#   (r.total_cost / r.distance)::NUMERIC(10,2) AS price_per_km
lab5-# FROM ride r
lab5-# WHERE r.id = currval('ride_id_seq');
 id | distance | total_cost | price_per_km
----+----------+------------+--------------
 17 |     7.20 |      57.60 |         8.00
(1 row)
```

Рисунок 1.7 — Результат выполнения триггера

1.2.2 Проверка совпадения payment.amount с ride.total_cost

```
CREATE OR REPLACE FUNCTION chk_payment_amount() RETURNS TRIGGER AS $$
BEGIN
  IF NEW.amount <> (
    SELECT total_cost
      FROM ride
     WHERE id = NEW.ride_id
  ) THEN
    RAISE EXCEPTION 'Sum of payment % != expected %', NEW.amount,
      (SELECT total_cost FROM ride WHERE id = NEW.ride_id);
  END IF;
  RETURN NEW;
END;
$$ LANGUAGE plpgsql;

CREATE TRIGGER trg_chk_payment
BEFORE INSERT OR UPDATE ON payment
FOR EACH ROW
EXECUTE PROCEDURE chk_payment_amount();
```

Проверка триггера:

```
lab5=# INSERT INTO payment(ride_id, amount, payment_method)
lab5-# VALUES (currval('ride_id_seq'), 3.5 * 8.00, 'cash');
ERROR:  Сумма оплаты 28.00 ≠ ожидаемой 57.60
CONTEXT:  PL/pgSQL function chk_payment_amount() line 8 at RAISE
lab5=# INSERT INTO payment(ride_id, amount, payment_method)
lab5-# VALUES (currval('ride_id_seq'), 100.00, 'cash');
ERROR:  Сумма оплаты 100.00 ≠ ожидаемой 57.60
CONTEXT:  PL/pgSQL function chk_payment_amount() line 8 at RAISE
```

Рисунок 1.8 — Результат выполнения триггера

1.2.3 Логирование отменённых поездок

```
CREATE TABLE IF NOT EXISTS cancelled_rides_log (
    ride_id      INT,
    cancelled_at TIMESTAMPTZ DEFAULT now()
  );

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

```
7       BEGIN
8         INSERT INTO cancelled_rides_log(ride_id) VALUES (NEW.id);
9         RETURN NEW;
10      END;
11      $$ LANGUAGE plpgsql;
12
13      CREATE TRIGGER trg_log_cancel
14      AFTER UPDATE OF status ON ride
15      FOR EACH ROW
16      WHEN (NEW.status = 'cancelled' AND OLD.status <> 'cancelled')
17      EXECUTE PROCEDURE log_ride_cancel();
```

Проверка триггера:

```
lab5=# UPDATE ride SET status = 'cancelled' WHERE id = 1;
UPDATE 1
lab5=# SELECT * FROM cancelled_rides_log WHERE ride_id = 1;
 ride_id |         cancelled_at
---------+-------------------------------
       1 | 2025-05-29 09:20:30.864098+00
(1 row)
```

Рисунок 1.9 — Результат выполнения триггера

ЗАКЛЮЧЕНИЕ

В рамках данной работы я получил практические навыки использования процедур, функций и триггеров в базе данных PostgreSQL, создав согласно заданию три процедуры и три триггера для своей базы данных.