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 Малахов Алексей Витальевич
Факультет прикладной информатики
Группа K3239
Направление подготовки 09.03.03 Прикладная информатика
Образовательная программа Мобильные и сетевые технологии 2023
Преподаватель Говорова Марина Михайловна

Санкт-Петербург
2025/2026

1. **Цель работы:** овладеть практическими создания и использования процедур, функций и триггеров в базе данных PostgreSQL.
2. **Практическое задание:**

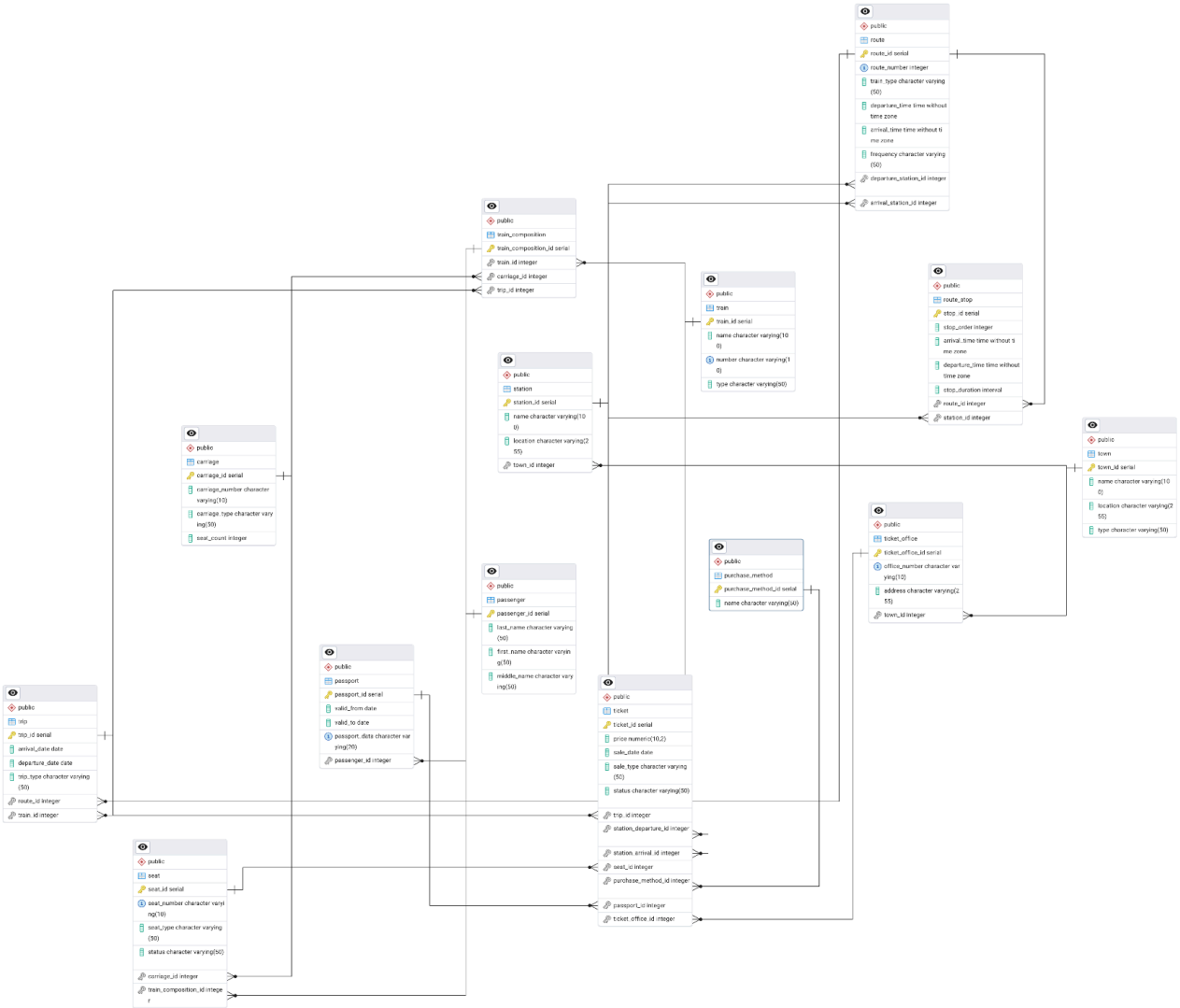
   Создать 3 процедуры для индивидуальной БД согласно варианту (часть 4 ЛР 2). Допустимо использование IN/OUT параметров. Допустимо создать авторские процедуры. (3 балла)

   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).

4. **Выполнение:**
   1. Наименование создаваемой БД: **Rail Ticket Database** (Вариант 6)
   2. Схема логической модели базы данных, сгенерированная в pgadmin:



3. Процедуры:

Повышение цен в пригородные поезда на 20%:

```
CREATE PROCEDURE increase_ticket_price(
    train_type TEXT,
    ticket_status TEXT,
    percent NUMERIC
)
LANGUAGE plpgsql
AS $$
BEGIN
    UPDATE ticket t
    SET price = price * (1 + percent / 100)
    FROM trip tr
    JOIN train trn ON trn.train_id = tr.train_id
    WHERE tr.trip_id = t.trip_id
      AND trn.type = train_type
      AND t.status = ticket_status;
END;
$$;
```

Использование: CALL increase_ticket_price('Пригородный', 'Продается', 20);

До:

	ticket_id [PK] integer	price numeric (10,2)	sale_date date	sale_type character varying (50)	status character varying (50)	trip_id integer
26	26	1900.00	[null]	Не продан	Продается	2
27	27	2000.00	[null]	Не продан	Продается	2
28	28	2100.00	[null]	Не продан	Продается	2
29	29	2200.00	[null]	Не продан	Продается	2
30	30	2300.00	[null]	Не продан	Продается	2
31	31	2500.00	2025-05-24	Онлайн	Куплен	10

После:

36	26	2280.00	[null]	Не продан	Продается	2
37	27	2400.00	[null]	Не продан	Продается	2
38	28	2520.00	[null]	Не продан	Продается	2
39	29	2640.00	[null]	Не продан	Продается	2
40	30	2760.00	[null]	Не продан	Продается	2

Создание нового рейса на поезд:

```
CREATE OR REPLACE PROCEDURE create_trip_with_carriages(
    p_arrival_date DATE,
    p_departure_date DATE,
    p_trip_type VARCHAR,
    p_route_id INTEGER,
    p_train_id INTEGER,
    p_carriage_ids INTEGER[]
)
LANGUAGE plpgsql
AS $$
DECLARE
    new_trip_id INTEGER;
    cid INTEGER;
BEGIN
    INSERT INTO trip(arrival_date, departure_date, trip_type, route_id, train_id)
    VALUES (p_arrival_date, p_departure_date, p_trip_type, p_route_id, p_train_id)
    RETURNING trip_id INTO new_trip_id;

    FOREACH cid IN ARRAY p_carriage_ids LOOP
        INSERT INTO train_composition(train_id, carriage_id, trip_id)
        VALUES (p_train_id, cid, new_trip_id);
    END LOOP;
END;
$$;
```

Использование: CALL create_trip_with_carriages('2025-05-22', '2025-05-21', 'Скоростной', 1, 1, ARRAY[8,9]);

Пояснение: процедура принимает даты прибытия и отбытия, тип рейса, номер маршрута и поезда, а затем массив номеров вагонов. После создается состав поезда с указанными вагонами и привязывается к новому рейсу.

До:

	trip_id [PK] integer	arrival_date date	departure_date date	trip_type character varying (50)	route_id integer	train_id integer
1	1	2025-05-17	2025-05-16	Скоростной	1	1
2	2	2025-05-18	2025-05-17	Пассажирский	2	2
3	3	2025-05-18	2025-05-17	Пассажирский	3	3
4	4	2025-05-17	2025-05-16	Обычный	4	4
5	5	2025-05-17	2025-05-16	Обычный	5	5
6	6	2025-05-17	2025-05-16	Скоростной	6	6
7	7	2025-05-18	2025-05-17	Пассажирский	7	7
8	8	2025-05-18	2025-05-17	Скоростной	8	8
9	10	2025-05-19	2025-05-18	Скоростной	1	1

	train_composition_id [PK] integer	train_id integer	carriage_id integer	trip_id integer
1	1	1	1	1
2	2	1	2	1
3	3	2	3	2
4	4	3	4	3
5	5	4	5	4
6	6	5	6	5
7	7	6	7	6
8	8	7	8	7
9	9	8	9	8
10	10	1	10	10

После:

10	10	1	10	10
11	11	1	8	11
12	12	1	9	11

9	10	2025-05-19	2025-05-18	Скоростной	1	1
10	11	2025-05-22	2025-05-21	Скоростной	1	1

Формирование общей выручки по продаже билетов за сутки:

```
CREATE OR REPLACE PROCEDURE calculate_daily_revenue(
    IN p_day DATE,
    OUT total NUMERIC
)
LANGUAGE plpgsql
AS $$
BEGIN
    SELECT COALESCE(SUM(price), 0)
    INTO total
    FROM ticket
    WHERE sale_date = p_day AND status = 'Куплен';
END;
$$;
```

Использование: CALL calculate_daily_revenue('2025-05-16', NULL);

Выполнение:
CALL calculate_daily_revenue('2025-05-16', NULL);

	total numeric
1	19700.00

CALL calculate_daily_revenue('2025-05-17', NULL);

	total numeric
1	28200.00

4. Триггеры:

Автоматическая установка текущей даты при продаже (если она не указана):

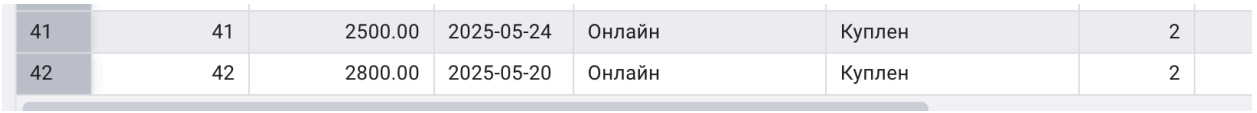
```
CREATE FUNCTION set_default_sale_date() RETURNS TRIGGER AS $$
BEGIN
  IF NEW.sale_date IS NULL THEN
    NEW.sale_date := CURRENT_DATE;
  END IF;
  RETURN NEW;
END;
$$ LANGUAGE plpgsql;

CREATE TRIGGER trg_default_sale_date
BEFORE INSERT ON ticket
FOR EACH ROW
EXECUTE FUNCTION set_default_sale_date();
```

Проверка:

```
INSERT INTO ticket(price, sale_type, status, trip_id,
                   station_departure_id, station_arrival_id,
                   seat_id, purchase_method_id, passport_id, ticket_office_id)
VALUES (2500, 'Онлайн', 'Куплен', 2, 2, 1, 10, 1, 1, 1);

INSERT INTO ticket(price, sale_date, sale_type, status, trip_id,
                   station_departure_id, station_arrival_id,
                   seat_id, purchase_method_id, passport_id, ticket_office_id)
VALUES (2800, '2025-05-20', 'Онлайн', 'Куплен', 2, 2, 1, 11, 1, 2, 1);
```



41	41	2500.00	2025-05-24	Онлайн	Куплен	2
42	42	2800.00	2025-05-20	Онлайн	Куплен	2

Автоматическая смена статуса места при покупке билета:

```
CREATE FUNCTION update_seat_on_purchase() RETURNS TRIGGER AS $$
BEGIN
  IF NEW.status = 'Куплен' THEN
    UPDATE seat SET status = 'Занято' WHERE seat_id = NEW.seat_id;
  END IF;
  RETURN NEW;
END;
$$ LANGUAGE plpgsql;

CREATE TRIGGER trg_seat_update_on_ticket
AFTER INSERT ON ticket
FOR EACH ROW
EXECUTE FUNCTION update_seat_on_purchase();
```

Проверка:

```
SELECT seat_id, status FROM seat WHERE seat_id = 12;
```

	seat_id [PK] integer	status character varying (50)
1	12	Свободно

```
INSERT INTO ticket(price, sale_date, sale_type, status, trip_id, station_departure_id, station_arrival_id, seat_id, purchase_method_id, passport_id, ticket_office_id)

VALUES (2600, CURRENT_DATE, 'Онлайн', 'Куплен', 2, 2, 1, 12, 1, 1, 1);
```

42	42	2800.00	2025-05-20	Онл...	Куп...	2	2	1	11
43	43	2600.00	2025-05-25	Онл...	Куп...	2	2	1	12

	seat_id [PK] integer	status character varying (50)
1	12	Занято

Логирование покупок билетов:

```
CREATE TABLE ticket_log (
    log_id SERIAL PRIMARY KEY,
    ticket_id INTEGER,
    operation TEXT,
    log_time TIMESTAMP DEFAULT CURRENT_TIMESTAMP
);

CREATE FUNCTION log_ticket_insert() RETURNS TRIGGER AS $$
BEGIN
  INSERT INTO ticket_log(ticket_id, operation)
  VALUES (NEW.ticket_id, 'INSERT');
  RETURN NEW;
END;
$$ LANGUAGE plpgsql;

CREATE TRIGGER trg_log_ticket_insert
AFTER INSERT ON ticket
FOR EACH ROW
EXECUTE FUNCTION log_ticket_insert();
```

Проверка:

```
INSERT INTO ticket(price, sale_date, sale_type, status, trip_id,
                   station_departure_id, station_arrival_id,
                   seat_id, purchase_method_id, passport_id, ticket_office_id)
VALUES (2500, CURRENT_DATE, 'Онлайн', 'Куплен', 2, 2, 1, 14, 1, 1, 1);

SELECT * FROM ticket_log ORDER BY log_id DESC LIMIT 5;
```

	log_id [PK] integer	ticket_id integer	operation text	log_time timestamp without time zone
1	4	49	INSERT	2025-05-25 00:15:02.205755
2	3	48	INSERT	2025-05-25 00:15:01.131673
3	2	47	INSERT	2025-05-25 00:15:00.323182
4	1	46	INSERT	2025-05-25 00:14:59.300334

Выводы: В ходе этой лабораторной работы я изучил и научился применять триггеры, процедуры и функции в СУБД PostgreSQL.