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 «**Базы данных**»

Автор: Акулов Алексей

Факультет: ФИКТ

Группа: К32391

Преподаватель: Говорова М.М.

Санкт-Петербург 2022

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

Оборудование: компьютерный класс.

Программное обеспечение: СУБД PostgreSQL, SQL Shell (psql).

Практическое задание:

Вариант 1

1. 2.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.

Вариант 6. БД «Пассажир»

Описание предметной области: Информационная система служит для продажи железнодорожных билетов. Билеты могут продаваться на текущие сутки или предварительно (не более чем за 45 суток). Цена билета при предварительной продаже снижается на 5%. Билет может быть приобретен в кассе или онлайн. Если билет приобретен в кассе, необходимо знать, в какой. Для каждой кассы известны номер и адрес. Кассы могут располагаться в различных населенных пунктах.

Поезда курсируют по расписанию, но могут назначаться дополнительные поезда на заданный период или определенные даты.

По всем промежуточным остановкам на маршруте известны название, тип населенного пункта, время прибытия, отправления, время стоянки.

БД должна содержать следующий минимальный набор сведений: Номер поезда. Название поезда. Тип поезда. Пункт назначения. Пункт назначения для проданного билета. Номер вагона. Тип вагона. Количество мест в вагоне. Цена билета. Дата отправления. Дата прибытия. Дата прибытия для пункта назначения проданного билета. Время отправления. Номер вагона в поезде. Номер билета. Место. Тип места. Фамилия пассажира. Имя пассажира. Отчество пассажира. Паспортные данные.

I. Название модели – «База пассажиров»
II. Состав реквизитов сущностей
    a. Пассажир (Паспортные данные, Фамилия, Имя, Отчество, Телефон, Электронная почта)
    b. Билет (Номер, Пассажир, Пункт назначения, Пункт отправления, Место, Цена, Статус оплаты, Статус возврата, В кассе или онлайн)
    c. Место (Номер места, Вагон, Статус занятости)
    d. Вагон (Номер вагона, Номер в поезде, Поезд, Количество мест, Тип)

e. Поезд (Номер поезда, Расписание, Дата отправления, Дата прибытия, Выполнение, Название)
f. Расписание поездов (Номер маршрута, Время отправления, Время прибытия, Тип, Пункт отправления, Пункт прибытия, Периодичность)
g. Остановка (Номер, Название, Тип, Маршрут)
h. Остановка поезда (Время прибытия, Время стоянки, Время отправления)
i. Касса (Номер кассы, Адрес, Населенный пункт)

Задание 2. Создать запросы:

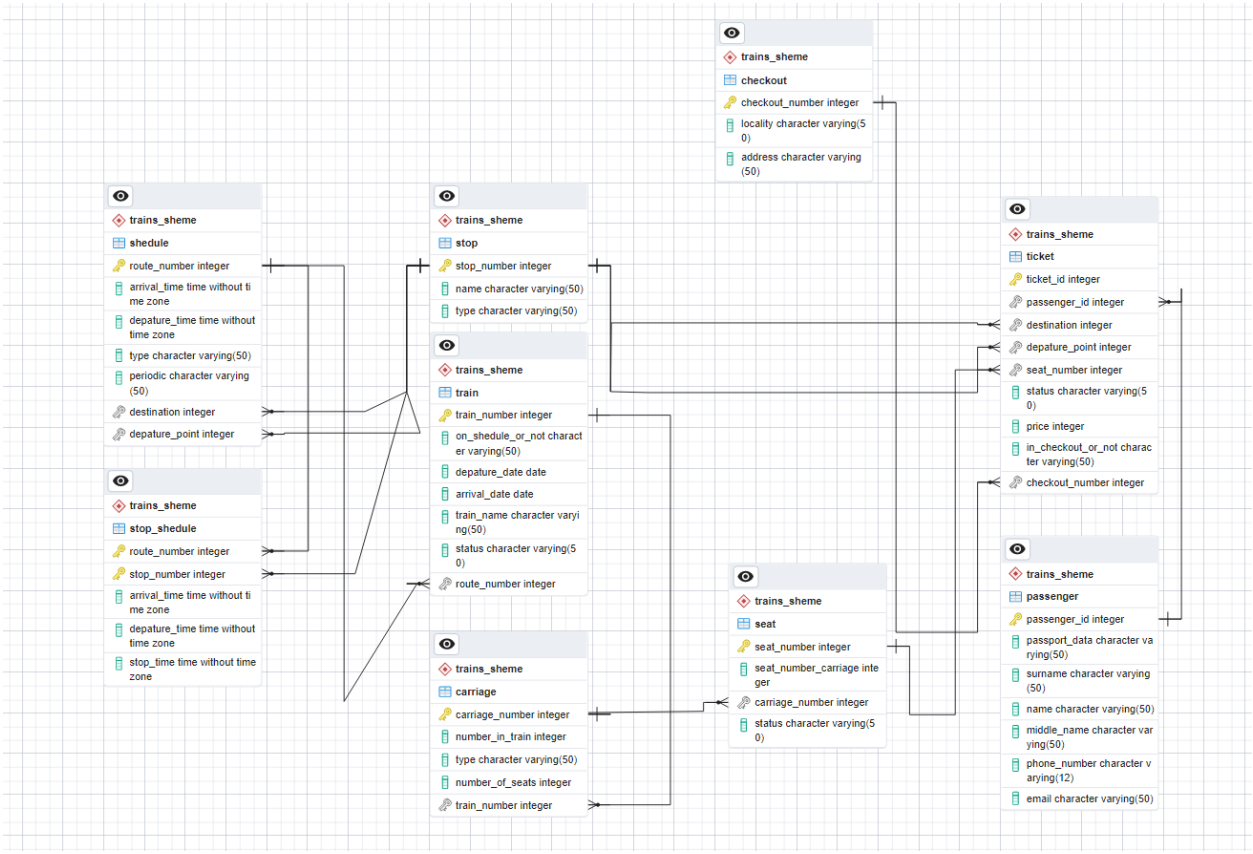
a) Свободные места на все поезда, отправляющиеся с вокзала в течение следующих суток.
b) Список пассажиров, отправившихся в Москву всеми рейсами за прошедшие сутки.
c) Номера поездов, на которые проданы все билеты на следующие сутки.
d) Свободные места в купейные вагоны всех рейсов до Москвы на текущие сутки.
e) Выручка от продажи билетов на все поезда за прошедшие сутки.
f) Общее количество билетов, проданных по всем направлениям в вагоны типа “СВ”.
g) Номера и названия поездов, все вагоны которых были заполнены менее чем наполовину за прошедшие сутки.

Задание 3. Создать представление:

- для пассажиров о наличии свободных мест на заданный рейс;
- количество непроданных билетов на все поезда, формирующиеся за прошедшие сутки (номер поезда, тип вагона, количество).

Выполнение:

1. Схема базы данных



2. Процедура для повышения цен в купейные поезда на 20%

```
CREATE OR REPLACE PROCEDURE increase_suburban_prices()
LANGUAGE plpgsql
AS $$
BEGIN
    UPDATE "trains_sheme"."ticket"
    SET "price" = "price" * 1.2
    WHERE "train_number" IN (
        SELECT "train_number"
        FROM "trains_sheme"."train" t
        JOIN "trains_sheme"."shedule" s ON t."route_number" = s."route_number"
        WHERE s."type" = 'coupe'
    );
END; $$;
```

Результаты:

	ticket_id [PK] integer	passenger_id integer	train_number integer	destination integer	depature_point integer	seat_number integer	status character varying (50)	price integer	in_checkout_or_not character varying (50)	checkout_number integer
1	1	1	1	2	5	1	paid	1000	yes	1
2	2	3	9	5	3	2	back	1500	no	2
3	3	3	10	6	3	3	changed	2000	yes	3
4	4	4	4	2	6	4	paid	2500	no	4
5	5	5	21	7	2	5	back	3000	yes	5
6	6	6	16	2	7	6	changed	3500	no	6
7	7	7	7	2	3	7	paid	4000	yes	7

	ticket_id [PK] integer	passenger_id integer	train_number integer	destination integer	depature_point integer	seat_number integer	status character varying (50)	price integer	in_checkout_or_not character varying (50)	checkout_number integer
1	1	1	1	2	5	1	paid	1000	yes	1
2	2	3	9	5	3	2	back	1500	no	2
3	3	3	10	6	3	3	changed	2000	yes	3
4	4	4	4	2	6	4	paid	2500	no	4
5	7	7	7	2	3	7	paid	4000	yes	7
6	5	5	21	7	2	5	back	3600	yes	5
7	6	6	16	2	7	6	changed	4200	no	6

3. Процедура для создания нового рейса на поезд.

```
CREATE OR REPLACE PROCEDURE create_new_train_route(
    _arrival_time TIME, _depature_time TIME, _type VARCHAR(50), _periodic VARCHAR(50),
    _destination VARCHAR(50), _depature_point VARCHAR(50), _train_name VARCHAR(50), _status VARCHAR(50),
    _depature_date DATE, _arrival_date DATE)
LANGUAGE plpgsql
AS $$
BEGIN
    INSERT INTO "trains_sheme"."shedule"("arrival_time", "depature_time", "type",
                                         "periodic", "destination", "depature_point")
    VALUES (_arrival_time, _depature_time, _type, _periodic,
            (SELECT stop_number FROM "trains_sheme"."stop" WHERE "name" = _destination),
            (SELECT stop_number FROM "trains_sheme"."stop" WHERE "name" = _depature_point));

    INSERT INTO "trains_sheme"."train"("on_shedule_or_not", "depature_date",
                                       "arrival_date", "train_name", "status", "route_number")
    VALUES ('YES', _depature_date, _arrival_date, _train_name, _status,
    (SELECT MAX("route_number") FROM "trains_sheme"."shedule"));
END; $$;
```

CALL create_new_train_route('10:00:00', '15:00:00', 'express', 'daily', 'MSC', 'EKB', 'Score train', 'success', '2023-06-01', '2023-06-02');

7	1	yes	2023-05-11	2023-05-11	Train 1	success	1
8	9	yes	2023-05-11	2023-05-11	Train 2	success	2
9	10	yes	2023-05-09	2023-05-09	Train 3	success	3
10	4	yes	2023-05-09	2023-05-09	Train 4	success	4
11	21	yes	2023-05-10	2023-05-10	Train 5	success	5
12	16	yes	2023-05-10	2023-05-10	Train 6	success	6
13	7	yes	2023-05-10	2023-05-10	Train 7	success	7

4. Процедура для формирования общей выручки по продаже билетов за сутки.

```
CREATE OR REPLACE PROCEDURE total_daily_revenue(_date DATE)
LANGUAGE plpgsql
AS $$
DECLARE
    total_revenue INTEGER;
BEGIN
    SELECT SUM("price") INTO total_revenue
    FROM "trains_sheme"."ticket"
    WHERE train_number IN (SELECT train_number FROM "trains_sheme"."train" WHERE "depature_date" = _date) AND "status"

    RAISE NOTICE 'Total revenue for the date %: %', _date, total_revenue;
END; $$
```

Результат запроса для третьей процедуры:

```
ЗАМЕЧАНИЕ:  Total revenue for the date 2023-05-11: 1000
CALL

Query returned successfully in 84 msec.
```

5. Дальше создаем таблицу и сам триггер, который записывает данные в журнал

```
CREATE TABLE "trains_sheme"."audit_log"
(
    "log_id" SERIAL PRIMARY KEY,
    "table_name" VARCHAR(128) NOT NULL,
    "operation" VARCHAR(50) NOT NULL,
    "old_data" TEXT,
    "new_data" TEXT,
    "timestamp" TIMESTAMP NOT NULL DEFAULT CURRENT_TIMESTAMP
);

CREATE OR REPLACE FUNCTION audit_trigger_func() RETURNS TRIGGER AS $audit_trigger$
BEGIN
    IF (TG_OP = 'DELETE') THEN
        INSERT INTO "trains_sheme"."audit_log" ("table_name", "operation", "old_data", "new_data")
        VALUES (TG_RELNAME, TG_OP, row_to_json(OLD)::text, NULL);
        RETURN OLD;
    ELSIF (TG_OP = 'UPDATE') THEN
        INSERT INTO "trains_sheme"."audit_log" ("table_name", "operation", "old_data", "new_data")
        VALUES (TG_RELNAME, TG_OP, row_to_json(OLD)::text, row_to_json(NEW)::text);
        RETURN NEW;
    ELSIF (TG_OP = 'INSERT') THEN
        INSERT INTO "trains_sheme"."audit_log" ("table_name", "operation", "old_data", "new_data")
        VALUES (TG_RELNAME, TG_OP, NULL, row_to_json(NEW)::text);
        RETURN NEW;
    END IF;
    RETURN NULL;
END;



CREATE TRIGGER audit_trigger
AFTER INSERT OR UPDATE OR DELETE ON "trains_sheme"."train"
FOR EACH ROW EXECUTE PROCEDURE audit_trigger_func();

CREATE TRIGGER audit_trigger1
AFTER INSERT OR UPDATE OR DELETE ON "trains_sheme"."ticket"
FOR EACH ROW EXECUTE PROCEDURE audit_trigger_func();
```

6. Результат работы триггера для таблицы ticket

	log_id [PK] integer	table_name character varying (128)	operation character varying (50)	old_data text
1	1	ticket	DELETE	{"ticket_id":8,"passenger_id":1,"train_number":4,"destination":2,"depature_point":1,"seat_number":7,"status":"paid","price":2000,"in_checkout_or_not":"yes",'

Вывод:
PGadmin позволяет создавать процедуры и триггеры и работать с ними. Также это можно делать используя консоль shell