Министерство науки и высшего образования Российской Федерации
федеральное государственное автономное образовательное учреждение высшего образования
«НАЦИОНАЛЬНЫЙ ИССЛЕДОВАТЕЛЬСКИЙ УНИВЕРСИТЕТ ИТМО»

Отчет

по лабораторной работе №3

«Процедуры, функции, триггеры в PostgreSQL»

по дисциплине «**Проектирование и реализация баз данных**»

Автор: Макунина А.А.

Факультет: ИКТ

Группа: К32421

Преподаватель: Говорова М.М.

Санкт-Петербург 2023

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:

1. 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.

Индивидуальное задание:

Вариант 8. БД «Аэропорт»

Описание предметной области: необходимо обеспечить продажу билетов на нужный рейс, при отсутствии билетов (необходимого количества билетов) предложить билет на ближайший рейс.

Рейсы выполняются по расписанию. Но есть рейсы, назначаемые на определенный период или разовые.

Рейс может иметь несколько транзитных посадок.

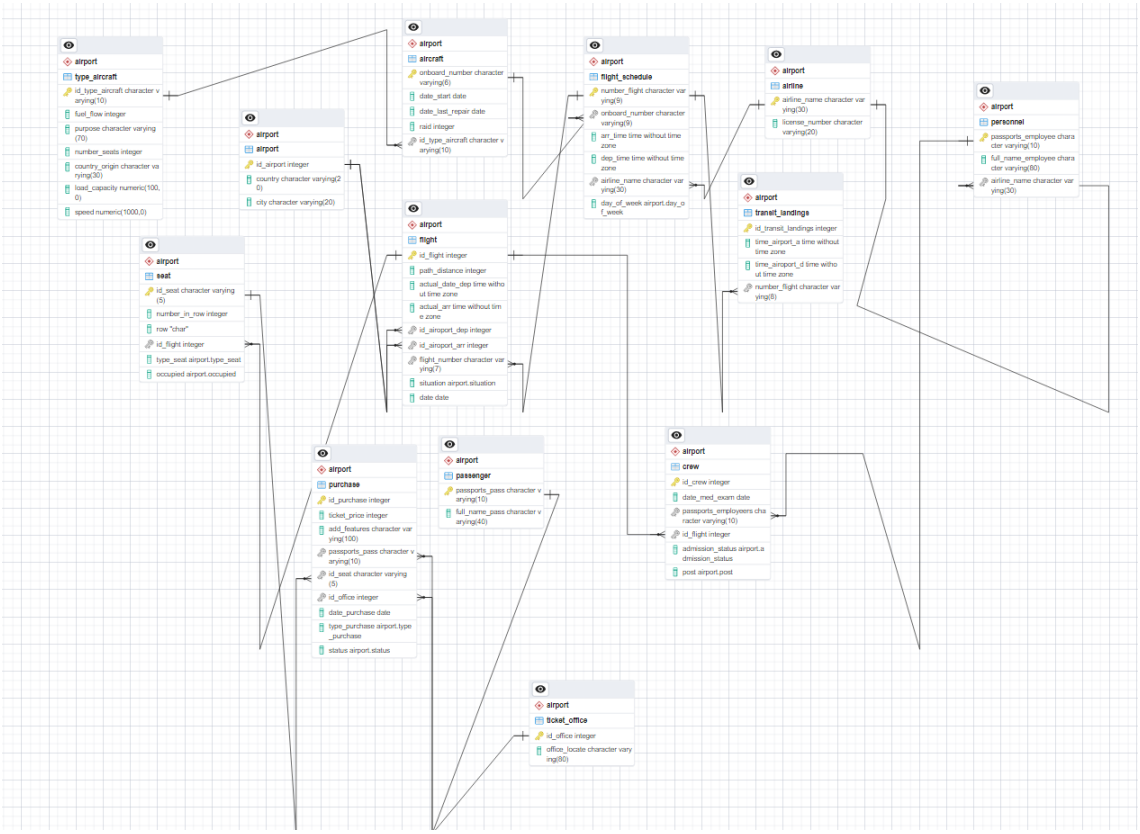
Билет может быть приобретен в кассе или онлайн. К базовой стоимости билета может быть дополнительная плата за выбор места, страховку багажа и т.п. Если билет приобретен в кассе, необходимо знать, в какой. Для каждой кассы известны номер и адрес. Кассы могут располагаться в различных населенных пунктах.

БД должна содержать следующий минимальный набор сведений: Бортовой номер самолета. Тип самолета. Количество мест. Страна. Производитель. Грузоподъемность. Скорость. Дата выпуска. Налѐт в часах. Дата последнего ремонта. Назначение самолета. Расход топлива. Код экипажа. Паспортные данные членов экипажа. Номер рейса. Дата вылета. Время вылета. Аэропорт вылета. Аэропорт назначения. Расстояние. Транзитные посадки (прилет, вылет, аэропорт, время в аэропорту). ФИО пассажира. Паспортные данные. Номер места. Тип места. Цена билета. Касса продажи билета (возможен электронный билет) (номер и адрес).

Задание 4. Создать хранимые процедуры:

- Для поиска билетов в заданный пункт назначения.
- Создания новой кассы продажи билетов.
- Определить расход топлива по всем маршрутам за истекший месяц.

Задание 5. Создать необходимые триггеры.



Выполнение:

1. Процедуры

- Для поиска билетов в заданный пункт назначения.

```
CREATE OR REPLACE FUNCTION FIND_TICKETS(DESTINATION_CITY TEXT)
RETURNS TABLE (ID_PURCHASE INTEGER, ID_SEAT CHARACTER varying(5),
                                                    PASSPORTS_PASS
CHARACTER varying(10)) AS $$
BEGIN
   RETURN QUERY
   SELECT
      p.id_purchase,
      p.id_seat,
      p.passports_pass
   FROM
      airport.purchase p
    JOIN airport.seat s ON p.id_seat = s.id_seat
   JOIN airport.flight r ON s.id_flight = r.id_flight
   JOIN airport.airport a ON r.id_airport_dep = a.id_airport
   WHERE
      a.city = destination_city;
END;
$$ LANGUAGE PLPGSQL;


SELECT *
FROM find_tickets('Novopolis');
```

```
Airport=# CREATE OR REPLACE FUNCTION FIND_TICKETS(DESTINATION_CITY TEXT) RETURNS TABLE (ID_PURCHASE INTEGER, ID_SEAT character varying(5), PASSPORTS_PASS character varying(10)) AS $$
Airport$# BEGIN
Airport$#     RETURN QUERY
Airport$#     SELECT
Airport$#         p.id_purchase,
Airport$#         p.id_seat,
Airport$#         p.passports_pass
Airport$#     FROM
Airport$#         airport.purchase p
Airport$# JOIN airport.seat s ON p.id_seat = s.id_seat
Airport$#     JOIN airport.flight r ON s.id_flight = r.id_flight
Airport$#     JOIN airport.airport a ON r.id_airport_dep = a.id_airport
Airport$#     WHERE
Airport$#         a.city = destination_city;
Airport$# END;
Airport$# $$ LANGUAGE PLPGSQL;
CREATE FUNCTION
Airport=# SELECT * FROM find_tickets('Novopolis');
 id_purchase | id_seat | passports_pass
-------------+---------+---------------
          50 | 5F39    | 2098011750
          66 | 5D39    | 2412782815
          75 | 3B39    | 2548269591
         118 | 5A39    | 3639657478
         131 | 1A39    | 3853526187
         150 | 5C39    | 4064500768
         175 | 2D39    | 4546013181
         179 | 1C39    | 4637671392
         190 | 3A39    | 4770521288
         223 | 3C39    | 5473529444
         234 | 6E39    | 5722238407
         280 | 1D39    | 6811396001
         281 | 5B39    | 6812672756
         285 | 4D39    | 6884709753
         291 | 6D39    | 7027621979
         304 | 6B39    | 7455051548
         305 | 2A39    | 7474331792
         321 | 4B39    | 7788207837
         358 | 1F39    | 8649323912
(19 строк)


Airport=#
```

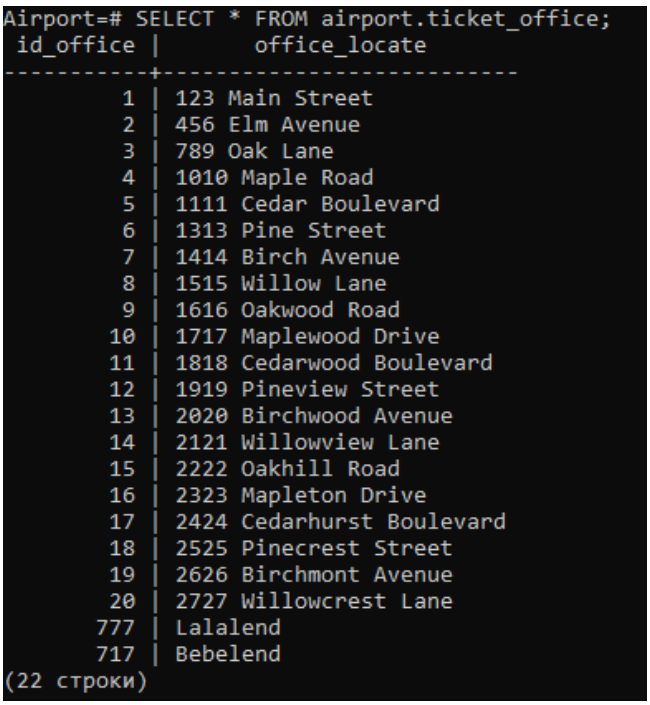
- Создания новой кассы продажи билетов.

CREATE OR REPLACE FUNCTION add_ticket (id_office INTEGER, office_locate VARCHAR(80)) RETURNS VOID AS $$
BEGIN
    INSERT INTO airport.ticket_office(id_office, office_locate)
    VALUES (id_office, office_locate);
END;
$$ LANGUAGE PLPGSQL;

SELECT add_ticket(777, 'Lalalend');

```
Airport=# CREATE OR REPLACE FUNCTION add_ticket (
Airport(#     id_office INTEGER,
Airport(#     office_locate VARCHAR(80)
Airport(# )
Airport-# RETURNS VOID
Airport-# AS $$
Airport$# BEGIN
Airport$#     INSERT INTO airport.ticket_office(id_office, office_locate)
Airport$#     VALUES (id_office, office_locate);
Airport$# END;
Airport$# $$ LANGUAGE plpgsql;
CREATE FUNCTION
Airport=# SELECT add_ticket(777, 'Lalalend');
 add_ticket
------------

(1 строка)
```



```
Airport=# SELECT * FROM airport.ticket_office;
 id_office |       office_locate
-----------+---------------------------
         1 | 123 Main Street
         2 | 456 Elm Avenue
         3 | 789 Oak Lane
         4 | 1010 Maple Road
         5 | 1111 Cedar Boulevard
         6 | 1313 Pine Street
         7 | 1414 Birch Avenue
         8 | 1515 Willow Lane
         9 | 1616 Oakwood Road
        10 | 1717 Maplewood Drive
        11 | 1818 Cedarwood Boulevard
        12 | 1919 Pineview Street
        13 | 2020 Birchwood Avenue
        14 | 2121 Willowview Lane
        15 | 2222 Oakhill Road
        16 | 2323 Mapleton Drive
        17 | 2424 Cedarhurst Boulevard
        18 | 2525 Pinecrest Street
        19 | 2626 Birchmont Avenue
        20 | 2727 Willowcrest Lane
       777 | Lalalend
       717 | Bebelend
(22 строки)
```

- Определить расход топлива по всем маршрутам за истекший месяц.

```
CREATE OR REPLACE FUNCTION fuel_consumption_last_month() RETURNS TABLE(flight_number VARCHAR(10), fuel_consumption DOUBLE PRECISION) AS $$
DECLARE
start_date DATE;
end_date DATE;
BEGIN
-- Определяем начало и конец искомого месяца
end_date = DATE_TRUNC('month', CURRENT_DATE)::DATE;
start_date = (end_date - INTERVAL '1 MONTH')::DATE + 1;
-- Выполняем запрос для расхода топлива за искомый период
RETURN QUERY SELECT
    flight.flight_number,
    CAST(ROUND(type_aircraft.fuel_flow * (EXTRACT(epoch FROM (flight_schedule.dep_time - flight_schedule.arr_time)) / 3600)::numeric, 2) AS double precision) AS fuel_consumption
FROM
    airport.flight
JOIN
    airport.flight_schedule ON flight.flight_number = flight_schedule.number_flight
JOIN
    airport.aircraft ON flight_schedule.onboard_number = aircraft.onboard_number
JOIN
    airport.type_aircraft ON aircraft.id_type_aircraft = type_aircraft.id_type_aircraft
WHERE
    flight.date >= start_date AND flight.date < end_date + INTERVAL '1 DAY'
GROUP BY
    flight.flight_number,
    type_aircraft.fuel_flow,
    flight_schedule.dep_time,
    flight_schedule.arr_time;
END;
```

$$ LANGUAGE PLPGSQL;

SELECT *
FROM fuel_consumption_last_month();

```
Airport=# CREATE OR REPLACE FUNCTION fuel_consumption_last_month()
Airport-# RETURNS TABLE(flight_number VARCHAR(10), fuel_consumption DOUBLE PRECISION) AS $$
Airport$# DECLARE
Airport$# start_date DATE;
Airport$# end_date DATE;
Airport$# BEGIN
Airport$# -- Определяем начало и конец искомого месяца
Airport$# end_date = DATE_TRUNC('month', CURRENT_DATE)::DATE;
Airport$# start_date = (end_date - INTERVAL '1 MONTH')::DATE + 1;
Airport$# -- Выполняем запрос для расхода топлива за искомый период
Airport$# RETURN QUERY SELECT
Airport$#     flight.flight_number,
Airport$#     CAST(ROUND(type_aircraft.fuel_flow * (EXTRACT(epoch FROM (flight_schedule.dep_time - flight_schedule.arr_time)) / 3600)::numeric, 2) AS double precision) AS fuel_consumption
Airport$# FROM
Airport$#     airport.flight
Airport$# JOIN
Airport$#     airport.flight_schedule ON flight.flight_number = flight_schedule.number_flight
Airport$# JOIN
Airport$#     airport.aircraft ON flight_schedule.onboard_number = aircraft.onboard_number
Airport$# JOIN
Airport$#     airport.type_aircraft ON aircraft.id_type_aircraft = type_aircraft.id_type_aircraft
Airport$# WHERE
Airport$#     flight.date >= start_date AND flight.date < end_date + INTERVAL '1 DAY'
Airport$# GROUP BY
Airport$#     flight.flight_number,
Airport$#     type_aircraft.fuel_flow,
Airport$#     flight_schedule.dep_time,
Airport$#     flight_schedule.arr_time;
Airport$# END;
Airport$# $$ LANGUAGE plpgsql;
CREATE FUNCTION
Airport=# SELECT * FROM fuel_consumption_last_month();
 flight_number | fuel_consumption
---------------+------------------
 KLM567        |          2100.83
(1 строка)


Airport=#
```

2. Триггеры

1. Создаем триггер log_changes_trigger, который срабатывает после каждого изменения (вставки, обновления или удаления) в таблицах airport.passenger, airport.purchase, airport.seat, airport.flight и airport.airport. Триггер вызывает функцию log_changes(), которая записывает информацию о событии изменения в таблицу airport.log_changes.

CREATE TABLE airport.log_changes (id SERIAL PRIMARY KEY,

TABLE_NAME TEXT NOT NULL,

change_type TEXT NOT NULL,

change_time TIMESTAMP NOT NULL);

CREATE OR REPLACE FUNCTION log_changes() RETURNS TRIGGER AS $$

BEGIN

IF TG_OP = 'DELETE' THEN

INSERT INTO airport.log_changes (table_name, change_type, change_time)

VALUES (TG_TABLE_NAME, 'DELETE', NOW());

ELSIF TG_OP = 'INSERT' THEN

INSERT INTO airport.log_changes (table_name, change_type, change_time)

VALUES (TG_TABLE_NAME, 'INSERT', NOW());

```
    ELSIF TG_OP = 'UPDATE' THEN
        INSERT INTO airport.log_changes (table_name, change_type, change_time)
        VALUES (TG_TABLE_NAME, 'UPDATE', NOW());
    END IF;
    RETURN NEW;
END;
$$ LANGUAGE PLPGSQL;


CREATE TRIGGER log_changes_passenger_trigger AFTER
INSERT
OR
UPDATE
OR
DELETE ON airport.passenger
FOR EACH ROW EXECUTE FUNCTION log_changes();


CREATE TRIGGER log_changes_purchase_trigger AFTER
INSERT
OR
UPDATE
OR
DELETE ON airport.purchase
FOR EACH ROW EXECUTE FUNCTION log_changes();


CREATE TRIGGER log_changes_seat_trigger AFTER
INSERT
OR
UPDATE
OR
```

DELETE ON airport.seat

FOR EACH ROW EXECUTE FUNCTION log_changes();

CREATE TRIGGER log_changes_flight_trigger AFTER

INSERT

OR

UPDATE

OR

DELETE ON airport.flight

FOR EACH ROW EXECUTE FUNCTION log_changes();

CREATE TRIGGER log_changes_airport_trigger AFTER

INSERT

OR

UPDATE

OR

DELETE ON airport.airport

FOR EACH ROW EXECUTE FUNCTION log_changes();

```
Airport=# CREATE TABLE airport.log_changes (
Airport(#     id SERIAL PRIMARY KEY,
Airport(#     table_name TEXT NOT NULL,
Airport(#     change_type TEXT NOT NULL,
Airport(#     change_time TIMESTAMP NOT NULL
Airport(# );
CREATE TABLE
```

```
Airport=# CREATE OR REPLACE FUNCTION log_changes()
Airport-# RETURNS TRIGGER AS $$
Airport$# BEGIN
Airport$#     IF TG_OP = 'DELETE' THEN
Airport$#         INSERT INTO airport.log_changes (table_name, change_type, change_time)
Airport$#         VALUES (TG_TABLE_NAME, 'DELETE', NOW());
Airport$#     ELSIF TG_OP = 'INSERT' THEN
Airport$#         INSERT INTO airport.log_changes (table_name, change_type, change_time)
Airport$#         VALUES (TG_TABLE_NAME, 'INSERT', NOW());
Airport$#     ELSIF TG_OP = 'UPDATE' THEN
Airport$#         INSERT INTO airport.log_changes (table_name, change_type, change_time)
Airport$#         VALUES (TG_TABLE_NAME, 'UPDATE', NOW());
Airport$#     END IF;
Airport$#     RETURN NEW;
Airport$# END;
Airport$# $$ LANGUAGE PLPGSQL;
CREATE FUNCTION
```

```
Airport=# CREATE TRIGGER log_changes_passenger_trigger
Airport-# AFTER INSERT OR UPDATE OR DELETE ON airport.passenger
Airport-# FOR EACH ROW
Airport-# EXECUTE FUNCTION log_changes();
CREATE TRIGGER
Airport=#
Airport=# CREATE TRIGGER log_changes_purchase_trigger
Airport-# AFTER INSERT OR UPDATE OR DELETE ON airport.purchase
Airport-# FOR EACH ROW
Airport-# EXECUTE FUNCTION log_changes();
CREATE TRIGGER
Airport=#
Airport=# CREATE TRIGGER log_changes_seat_trigger
Airport-# AFTER INSERT OR UPDATE OR DELETE ON airport.seat
Airport-# FOR EACH ROW
Airport-# EXECUTE FUNCTION log_changes();
CREATE TRIGGER
Airport=#
Airport=# CREATE TRIGGER log_changes_flight_trigger
Airport-# AFTER INSERT OR UPDATE OR DELETE ON airport.flight
Airport-# FOR EACH ROW
Airport-# EXECUTE FUNCTION log_changes();
CREATE TRIGGER
Airport=#
Airport=# CREATE TRIGGER log_changes_airport_trigger
Airport-# AFTER INSERT OR UPDATE OR DELETE ON airport.airport
Airport-# FOR EACH ROW
Airport-# EXECUTE FUNCTION log_changes();
CREATE TRIGGER
Airport=#
```

```
Airport=#  UPDATE airport.purchase SET add_features = 'With a baby' WHERE id_purchase = 364;
UPDATE 1
```

```
Airport=# SELECT * FROM airport.log_changes;
 id | table_name | change_type |        change_time
----+------------+-------------+---------------------------
  1 | purchase   | UPDATE      | 2023-04-23 11:26:32.284261
(1 строка)
```

2. При выполнении попытки удаления данных пассажира, если он приобретал билет менее 60 дней назад, будет выдана ошибка "Cannot delete passenger with recent ticket purchase".

```
CREATE OR REPLACE FUNCTION prevent_passenger_delete() RETURNS TRIGGER AS
$$
BEGIN
   IF EXISTS (
      SELECT *
      FROM airport.purchase p
      WHERE p.passports_pass = OLD.passports_pass
      AND p.date_purchase >= CURRENT_DATE - INTERVAL '60 days'
   ) THEN
      RAISE EXCEPTION 'Cannot delete passenger with recent ticket purchase';
   END IF;
   RETURN OLD;
END;
$$ LANGUAGE PLPGSQL;

CREATE TRIGGER prevent_passenger_delete_trigger
BEFORE
DELETE ON airport.passenger
FOR EACH ROW EXECUTE FUNCTION prevent_passenger_delete();
```

```
Airport=# CREATE OR REPLACE FUNCTION prevent_passenger_delete() RETURNS TRIGGER AS $$
Airport$# BEGIN
Airport$#     IF EXISTS (
Airport$#         SELECT *
Airport$#         FROM airport.purchase p
Airport$#         WHERE p.passports_pass = OLD.passports_pass
Airport$#         AND p.date_purchase >= CURRENT_DATE - INTERVAL '60 days'
Airport$#     ) THEN
Airport$#         RAISE EXCEPTION 'Cannot delete passenger with recent ticket purchase';
Airport$#     END IF;
Airport$#     RETURN OLD;
Airport$# END;
Airport$# $$ LANGUAGE PLPGSQL;
CREATE FUNCTION
Airport=#
Airport=# CREATE TRIGGER prevent_passenger_delete_trigger
Airport-# BEFORE DELETE ON airport.passenger
Airport-# FOR EACH ROW
Airport-# EXECUTE FUNCTION prevent_passenger_delete();
CREATE TRIGGER
```

```
Airport=# DELETE FROM airport.passenger
Airport-# WHERE passports_pass = '8814743491';
ОШИБКА:  Cannot delete passenger with recent ticket purchase
```

Вывод: в рамках данной лабораторной работы были приобретены практические навыки создания и использования процедур, функций и триггеров в базе данных PostgreSQL с использованием SQL Shell (psql).

В ходе выполнения практического задания были созданы процедуры и функции, соответствующие индивидуальному заданию, а также написан триггер для запрета удаления записей в таблице, если удаляемый элемент связан с другой таблицей.

Кроме того, был создан триггер для логирования событий вставки, удаления и редактирования данных в базе данных PostgreSQL. Для этого была создана отдельная таблица "log_changes", в которую добавляются записи о всех событиях, производимых в базе данных.

В результате выполнения лабораторной работы мы получили практические навыки работы с процедурами, функциями и триггерами в PostgreSQL, что позволит эффективнее работать с данными в базах данных.