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»

Вариант 10. БД «Автовокзал»

Автор: Чан Дык Минь

Факультет: ИКТ

Группа: К32392

Преподаватель: Говорова М. М.

Санкт-Петербург, 2023

1 Описание работы

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

Оборудование: компьютерный класс.

Программное обеспечение: СУБД PostgreSQL , pgAdmin 4.

Практическое задание:

Вариант 1

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.

Вариант 2

1. Создать процедуры/функции согласно индивидуальному заданию и (согласно индивидуальному заданию, часть 4).

2.1. Модифицировать триггер (триггерную функцию) на проверку корректности входа и выхода сотрудника (см. Практическое задание 1 Лабораторного практикума (Приложение)) с максимальным учетом «узких» мест некорректных данных по входу и выходу.

2.2. Создать авторский триггер по варианту индивидуального задания.

Указание. Работа выполняется в консоли SQL Shell (psql).

2 Описание предметной области

Вариант 10. БД «Автовокзал»

Описание предметной области: С автовокзала ежедневно отправляется несколько междугородных/международных автобусных рейсов. Номер рейса определяется маршрутом и временем отправления. По всем промежуточным остановкам на маршруте известны название, тип населенного пункта, время прибытия, отправления, время стоянки. Автобусы курсируют по расписанию, но могут назначаться дополнительные рейсы на заданный период или определенные даты. Билеты могут продаваться предварительно, но не ранее чем за 10 суток. В билете указывается номер места в автобусе. На каждый рейс может продаваться не более 10 билетов без места, цена на которые снижается на 10%. Пунктами отправления и назначения, согласно билету, могут быть промежуточные остановки. Билеты могут продаваться в кассе автовокзала или онлайн. На каждый рейс формируется экипаж из двух водителей. БД должна содержать следующий минимальный набор сведений: Номер рейса. Номер водителя. Номер автобуса. Паспортные данные водителя. Пункт отправления. Пункт назначения. Промежуточные остановки. Дата отправления. Время отправления. Время в пути. Тип автобуса. Количество мест в автобусе. Страна. Производитель. Год выпуска. Номер билета. Номер места в автобусе (при наличии). Цена билета. ФИО пассажира. Паспортные данные пассажира.

Задание 4. Создать хранимые процедуры:

- Продажи билета.
- Возврата билета.
- Добавления нового рейса.

Задание 5. Создать необходимые триггеры.

Вариант 1

I. Создать процедуры.

1. Продажи билета.

Таблица "билеты" сначала:

```
LAB1=# select * from tickets;
 passport  | seat_code | trip_id | ticket_price | landing_point | drop_point | payment_type | status  |   sale_type   | ticket_code
-----------+-----------+---------+--------------+---------------+------------+--------------+---------+---------------+-------------
 EF3456789 |         1 |       1 |          100 | City B        | City A     | card         | payed   | sell directly |           1
 GH5678901 |         2 |       2 |          150 | City C        | City B     | cash         | waiting | sell by phone |           2
(2 rows)
```

Создать процедуру:

```
ALTER SEQUENCE ticket_code_seq RESTART WITH 3;
CREATE OR REPLACE PROCEDURE public.sell_ticket(
  IN p_passport varchar(12),
  IN p_seat_code integer,
  IN p_trip_id integer,
  IN p_ticket_price integer,
  IN p_landing_point varchar(200),
  IN p_drop_point varchar(200),
  IN p_payment_type varchar(10),
  IN p_sale_type varchar(30)
)
LANGUAGE plpgsql
AS $$
DECLARE
  v_seat_status varchar(10);
BEGIN
  SELECT status INTO v_seat_status
  FROM public."tickets"
  WHERE seat_code = p_seat_code AND trip_id = p_trip_id;
  IF v_seat_status = 'booked' THEN
    RAISE EXCEPTION 'Seat already booked!';
  ELSE
      INSERT INTO public."tickets" (
         "passport",
         "seat_code",
         "trip_id",
         "ticket_price",
```

```
          "landing_point",
          "drop_point",
          "payment_type",
          "status",
          "sale_type",
          "ticket_code"
       ) VALUES (
          p_passport,
          p_seat_code,
          p_trip_id,
          p_ticket_price,
          p_landing_point,
          p_drop_point,
          p_payment_type,
          'payed',
          p_sale_type,
          nextval('ticket_code_seq')
       );
       UPDATE public.seats
       SET book_status = 'booked'
       WHERE seat_code = p_seat_code AND trip_id = p_trip_id;
     END IF;
END;
$$;
```

```
CALL public.sell_ticket(
     'C1223445',
     4,
     3,
     200,
     'City A',
     'City C',
     'cash',
     'sell directly'
);
```

Таблица "tickets" после:

```
LAB1=# select * from tickets;
 passport  | seat_code | trip_id | ticket_price | landing_point | drop_point | payment_type | status  |   sale_type    | ticket_code
-----------+-----------+---------+--------------+---------------+------------+--------------+---------+----------------+-------------
 EF3456789 |         1 |       1 |          100 | City B        | City A     | card         | payed   | sell directly  |           1
 GH5678901 |         2 |       2 |          150 | City C        | City B     | cash         | waiting | sell by phone  |           2
 C1223445  |         4 |       3 |          200 | City A        | City C     | cash         | payed   | sell directly  |           3
(3 rows)
```

2. **Возврата билета.**

Таблица "tickets" сначала:

```
LAB1=# select * from tickets;
 passport  | seat_code | trip_id | ticket_price | landing_point | drop_point | payment_type | status  |   sale_type    | ticket_code
-----------+-----------+---------+--------------+---------------+------------+--------------+---------+----------------+-------------
 EF3456789 |         1 |       1 |          100 | City B        | City A     | card         | payed   | sell directly  |           1
 GH5678901 |         2 |       2 |          150 | City C        | City B     | cash         | waiting | sell by phone  |           2
 C1223445  |         4 |       3 |          200 | City A        | City C     | cash         | payed   | sell directly  |           3
(3 rows)
```

Создать процедуру:

```
CREATE OR REPLACE PROCEDURE public.refund_ticket(
  IN p_ticket_code integer
)
LANGUAGE plpgsql
AS $$
BEGIN
  UPDATE public."tickets"
  SET status = 'refund'
  WHERE ticket_code = p_ticket_code;
  UPDATE public.seats
  SET book_status = 'canceled'
  WHERE seat_code IN(
       SELECT seat_code
       FROM public.tickets
       WHERE ticket_code = p_ticket_code
  );
END;
$$;
```

```
CALL public.refund_ticket(3);
```

Таблица "tickets" и "seats" после:

```
LAB1=# select * from tickets;
 passport  | seat_code | trip_id | ticket_price | landing_point | drop_point | payment_type | status  |   sale_type   | ticket_code
-----------+-----------+---------+--------------+---------------+------------+--------------+---------+---------------+-------------
 EF3456789 |         1 |       1 |          100 | City B        | City A     | card         | payed   | sell directly |           1
 GH5678901 |         2 |       2 |          150 | City C        | City B     | cash         | waiting | sell by phone |           2
 C1223445  |         4 |       3 |          200 | City A        | City C     | cash         | refund  | sell directly |           3
(3 rows)


LAB1=# select * from seats;
 seat_code | seat_id | book_status | trip_id
-----------+---------+-------------+---------
         1 |      10 | empty       |       1
         2 |      11 | booked      |       2
         3 |      12 | booked      |       1
         4 |      13 | canceled    |       3
(4 rows)
```

3. Добавления нового рейса.

Таблица "trips" сначала:

```
LAB1=# select * from trips;
 trip_id | arrival_point | departure_point | bus_id | actual_departure_time | actual_arrival_time |   status
---------+---------------+-----------------+--------+-----------------------+---------------------+-------------
       1 | City B        | City A          |      1 | 2023-05-18 10:15:00   | 2023-05-18 11:45:00 | arrived
       2 | City C        | City B          |      2 | 2023-05-18 13:15:00   | 2023-05-18 14:45:00 | arrived
       3 | City A        | City C          |      3 | 2023-05-18 16:15:00   | 2023-05-18 17:45:00 | not arrived
(3 rows)
```

Создать процедуру:

```
CREATE OR REPLACE PROCEDURE public.add_trip(
  IN p_departure_point varchar(200),
  IN p_arrival_point varchar(200),
  IN p_departure_time timestamp,
  IN p_arrival_time timestamp,
  IN p_bus_id integer
)
LANGUAGE plpgsql
AS $$
BEGIN
  INSERT INTO public.trips (
     "trip_id",
    "departure_point",
    "arrival_point",
    "actual_departure_time",
    "actual_arrival_time",
    "bus_id",
    "status"
  ) VALUES (
```

```
     nextval('trip_id_seq'),
    p_departure_point,
    p_arrival_point,
    p_departure_time,
    p_arrival_time,
    p_bus_id,
    'not arrived'
  ) ;
END;
$$;
```

```
CALL public.add_trip(
  'City A',
  'City C',
  '2023-05-20 08:00:00',
  '2023-05-20 12:00:00',
  1
);
```

Таблица "trips" после:

```
LAB1=# select * from trips;
 trip_id | arrival_point | departure_point | bus_id | actual_departure_time | actual_arrival_time |   status
---------+---------------+-----------------+--------+-----------------------+---------------------+-------------

       1 | City B        | City A          |      1 | 2023-05-18 10:15:00   | 2023-05-18 11:45:00 | arrived
       2 | City C        | City B          |      2 | 2023-05-18 13:15:00   | 2023-05-18 14:45:00 | arrived
       3 | City A        | City C          |      3 | 2023-05-18 16:15:00   | 2023-05-18 17:45:00 | not arrived
      10 | City C        | City A          |      1 | 2023-05-20 08:00:00   | 2023-05-20 12:00:00 | not arrived
(4 rows)
```

II. Создать триггер для логирования событий вставки, удаления, редактирования данных в базе данных PostgreSQL.

1) **Создать триггерную функцию**

```
CREATE OR REPLACE FUNCTION public.log_changes() RETURNS trigger
AS $$
BEGIN
  IF (TG_OP = 'DELETE') THEN
    INSERT INTO public.changelog (table_name, action, timestamp,
old_data)
    VALUES (jsonb_build_object('table_name',
quote_ident(TG_TABLE_NAME)), 'DELETE', NOW(), to_jsonb(OLD));
    RETURN OLD;
  ELSIF (TG_OP = 'UPDATE') THEN
    INSERT INTO public.changelog (table_name, action, timestamp,
old_data, new_data)
    VALUES (jsonb_build_object('table_name',
quote_ident(TG_TABLE_NAME)), 'UPDATE', NOW(), to_jsonb(OLD),
to_jsonb(NEW));
    RETURN NEW;
  ELSIF (TG_OP = 'INSERT') THEN
    INSERT INTO public.changelog (table_name, action, timestamp,
new_data)
    VALUES (jsonb_build_object('table_name',
quote_ident(TG_TABLE_NAME)), 'INSERT', NOW(), to_jsonb(NEW));
    RETURN NEW;
  END IF;
END;
$$ LANGUAGE plpgsql;
```

2) **Создание триггеров для таблиц «Поездки» и «Билеты»**

```
CREATE TRIGGER log_changes
AFTER INSERT OR UPDATE OR DELETE
ON public.tickets
FOR EACH ROW
EXECUTE FUNCTION public.log_changes();
```

```sql
CREATE TRIGGER log_changes
AFTER INSERT OR UPDATE OR DELETE
ON public.trips
FOR EACH ROW
EXECUTE FUNCTION public.log_changes();
```

3) **Вызывать функции «INSERT», «UPDATE», «DELETE», чтобы проверить, работает ли триггер.**

```sql
CALL public.add_trip(
  'City B',
  'City A',
  '2023-05-20 08:00:00',
  '2023-05-20 12:00:00',
  1
);
```

```sql
CALL public.refund_ticket(2);
```

```sql
DELETE from trips WHERE trip_id=10;
```

4) **Таблица «changelog» записывает историю срабатывания**

```
LAB1=# select * from public.changelog;
 id |        table_name        | action |         timestamp          |
                                                                             old_data
                                                                                         |
                                                                                              new_data

----+--------------------------+--------+----------------------------+------------------------------------------------------------------------------------------------------------------------------------
------------------------------------------------------------------------------+------------------------------------------------------------------------------------------------------------------------
-------------------------------------------------------------------------------------------
  1 | {"table_name": "trips"}   | INSERT | 2023-05-17 21:02:03.592231 |

                                                                                         | {"bus_id": 1, "status": "not arrived", "trip_id": 15, "arrival_point": "City A", "departure_point": "City B", "actual_a
rrival_time": "2023-05-20T12:00:00", "actual_departure_time": "2023-05-20T08:00:00"}
  2 | {"table_name": "tickets"} | UPDATE | 2023-05-17 21:07:20.609316 | {"status": "refund", "trip_id": 2, "passport": "GH5678901", "sale_type": "sell by phone", "seat_code": 2, "drop_point": "City B", "ticket
_code": 2, "payment_type": "cash", "ticket_price": 150, "landing_point": "City C"} | {"status": "refund", "trip_id": 2, "passport": "GH5678901", "sale_type": "sell by phone", "seat_code": 2, "drop_point": "Cit
y B", "ticket_code": 2, "payment_type": "cash", "ticket_price": 150, "landing_point": "City C"}
  3 | {"table_name": "trips"}   | DELETE | 2023-05-17 21:09:18.479828 | {"bus_id": 1, "status": "not arrived", "trip_id": 10, "arrival_point": "City C", "departure_point": "City A", "actual_arrival_time": "202
3-05-20T12:00:00", "actual_departure_time": "2023-05-20T08:00:00"}                    |
(3 rows)
```

4 Вывод

Во время лабораторной работы я уже знаю использование функции, процедуры и триггера. Узнайте, как создавать функции, процедуры и триггеры, и просмотрите историю записи данных в измененную таблицу.

На мой взгляд, эти возможности PostgresQL очень важны, и их следует изучить, их можно много применять.