ЛАБОРАТОРНАЯ РАБОТА №5

ПРОЦЕДУРЫ, ФУНКЦИИ, ТРИГГЕРЫ В POSTGRESQL

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 (min - 6 баллов, max - 10 баллов, доп. баллы - 3):

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).

Дополнительные баллы - 3:

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).

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

Выполнил: Мищенко Максим K3239

Процедура 1. добавление нового рейса

```
CREATE OR REPLACE PROCEDURE sp_add_flight(
    in_plane_id           INT,
    in_route_id           INT,
    in_status             TEXT,
    in_distance           INT,
    in_departure_offset   INTERVAL,
    OUT out_flight_id     INT
)
LANGUAGE plpgsql
AS $$
BEGIN
  INSERT INTO public.flights (
    plane_id,
    route_id,
    status,
    distance,
    departure_datetime_real,
    arrival_datetime_real
  )
  VALUES (
    in_plane_id,
    in_route_id,
    in_status,
    in_distance,
    now() + in_departure_offset,
    now() + in_departure_offset + (in_distance::numeric / 500 || ' hours')::interval
  )
  RETURNING flight_id INTO out_flight_id;
END;
$$;
```

```
DO $$
DECLARE
  new_fid INT;
BEGIN
  CALL sp_add_flight(1, 2, 'scheduled', 1200, '1 day', new_fid);
  RAISE NOTICE 'New flight_id = %', new_fid;
END
$$;
```

Процедура 2. отмена рейса и освобождение мест

```
CREATE OR REPLACE PROCEDURE sp_cancel_flight(
    in_flight_id INT
)
LANGUAGE plpgsql
AS $$
BEGIN
  UPDATE public.flights
  SET status = 'cancelled'
  WHERE flight_id = in_flight_id;

  UPDATE public.seats s
    SET status = 'reserved'
  FROM public.tickets t
  WHERE t.flight_id = in_flight_id
    AND t.seat_id = s.seat_id
    AND t.status <> 'cancelled';
END;
$$;


CALL sp_cancel_flight(5);
```

Процедура 3. подсчёт загрузки самолёта

```
CREATE OR REPLACE PROCEDURE sp_plane_load(
    in_flight_id     INT,
    OUT load_percent NUMERIC
)
LANGUAGE plpgsql
AS $$
DECLARE
  total_seats    INT;
  occupied_count INT;
BEGIN
  SELECT COUNT(*) INTO total_seats
    FROM public.seats;

  SELECT COUNT(*) INTO occupied_count
```

```
    FROM public.tickets t
    WHERE t.flight_id = in_flight_id
      AND t.status = 'booked';

  IF total_seats = 0 THEN
    load_percent := 0;
  ELSE
    load_percent := (occupied_count::numeric / total_seats) * 100;
  END IF;
END;
$$;


DO $$
DECLARE
  load_pct NUMERIC;
BEGIN
  CALL sp_plane_load(1, load_pct);
  RAISE NOTICE 'New load_pct = %', load_pct;
END
$$;
```

2. Триггеры

2.1. Логирование изменений в таблице flights

```
CREATE TABLE IF NOT EXISTS flights_log (
  log_id      SERIAL PRIMARY KEY,
  action_time TIMESTAMP NOT NULL DEFAULT now(),
  op_type     TEXT        NOT NULL,
  flight_data JSONB       NOT NULL
);


CREATE OR REPLACE FUNCTION fn_log_flight_changes()
RETURNS trigger
LANGUAGE plpgsql
AS $$
BEGIN
  IF TG_OP = 'INSERT' THEN
    INSERT INTO flights_log(op_type, flight_data)
    VALUES ('INSERT', to_jsonb(NEW));
```

```
    RETURN NEW;
  ELSIF TG_OP = 'UPDATE' THEN
    INSERT INTO flights_log(op_type, flight_data)
    VALUES ('UPDATE', to_jsonb(NEW));
    RETURN NEW;
  ELSIF TG_OP = 'DELETE' THEN
    INSERT INTO flights_log(op_type, flight_data)
    VALUES ('DELETE', to_jsonb(OLD));
    RETURN OLD;
  END IF;
END;
$$;



CREATE TRIGGER trg_flights_log
AFTER INSERT OR UPDATE OR DELETE
ON public.flights
FOR EACH ROW
EXECUTE PROCEDURE fn_log_flight_changes();
```

2.2. Проверка корректности статуса кресла (BEFORE INSERT на tickets)

Нельзя продавать билет на несуществующее или уже занятое кресло.

```
CREATE OR REPLACE FUNCTION fn_check_seat_availability()
RETURNS trigger
LANGUAGE plpgsql
AS $$
DECLARE
  seat_stat TEXT;
BEGIN
  SELECT status INTO seat_stat
    FROM public.seats
    WHERE seat_id = NEW.seat_id;

  IF NOT FOUND THEN
    RAISE EXCEPTION 'Seat ID % does not exist', NEW.seat_id;
```

```
  ELSIF seat_stat <> ‘available’ THEN
    RAISE EXCEPTION 'Seat ID % is not available (status=%)', NEW.seat_id,
seat_stat;
  END IF;
  RETURN NEW;
END;
$$;



CREATE TRIGGER trg_check_seat
BEFORE INSERT ON public.tickets
FOR EACH ROW
EXECUTE PROCEDURE fn_check_seat_availability();
```

2.3. Автоматическое обновление статуса кресла (AFTER INSERT на tickets)

После успешной продажи билета помечаем кресло как occupied

```
CREATE OR REPLACE FUNCTION fn_mark_seat_occupied()
RETURNS trigger
LANGUAGE plpgsql
AS $$
BEGIN
  UPDATE public.seats
  SET status = 'occupied'
  WHERE seat_id = NEW.seat_id;
  RETURN NEW;
END;
$$;


CREATE TRIGGER trg_mark_seat
AFTER INSERT ON public.tickets
FOR EACH ROW
EXECUTE PROCEDURE fn_mark_seat_occupied();
```