

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»

Автор: Гуторова И.В.

Факультет: ИКТ

Группа: К3241

Преподаватель: Говорова М.М.



Санкт-Петербург 2023

Оглавление

1. Создать процедуры согласно индивидуальному заданию	3
2. Модифицировать триггер	5
3. Создать необходимый триггер.....	7
Вывод.....	8

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 (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.

1. Создать процедуры согласно индивидуальному заданию

Создать хранимые процедуры:

- Для поиска билетов в заданный пункт назначения.

```
CREATE OR REPLACE FUNCTION find_tickets_to_destination(destination_city
VARCHAR(20))
RETURNS TABLE (
    id_ticket INT,
    passenger_name TEXT,
    date_departure DATE,
    date_arrival DATE
)
LANGUAGE plpgsql AS $$
BEGIN
    RETURN QUERY
        SELECT t.id_ticket, CONCAT(p.name, ' ', p.surname) AS passenger_name,
        r.date_departure, r.date_arrival
        FROM schema.ticket t
        JOIN schema.passenger p ON t.id_passenger = p.id_passenger
        JOIN schema.route r ON t.id_route = r.id_route
        JOIN schema.schedule s ON r.id_schedule = s.id_schedule
        JOIN schema.airport a_arr ON s.id_airport_arrival = a_arr.id_airport
        WHERE a_arr.city = destination_city
        ORDER BY r.date_departure;

END;
$$;
```

Query

Query History

1

SELECT * from schema.find_tickets_to_destination('Moscow')

Data Output

Messages

Notifications

	id_ticket integer	passenger_name text	date_departure date	date_arrival date
1	2	Name 1 Surname 1	2023-10-28	2023-10-28
2	1	Name 1 Surname 1	2023-10-28	2023-10-28
3	3	Name 3 Surname 3	2023-10-30	2023-10-30
4	9	Name 1 Surname 1	2023-11-13	2023-11-13
5	8	Name 2 Surname 2	2023-11-16	2023-11-16

- Создания новой кассы продажи билетов.

```
CREATE OR REPLACE PROCEDURE create_ticket_office(
    city VARCHAR(20),
    address VARCHAR(50),
    country VARCHAR(20)
)
LANGUAGE SQL
AS $$
    INSERT INTO schema.ticket_office (city, address, country)
    VALUES (city, address, country);
$$;
```

7	8	city	address	country
---	---	------	---------	---------

- Определить расход топлива по всем маршрутам за истекший месяц.

```
CREATE OR REPLACE FUNCTION display_fuel_consumption_table()
RETURNS TABLE (
    id_route INT,
    fuel_consumption NUMERIC
)
LANGUAGE plpgsql AS $$
BEGIN
    RETURN QUERY
    SELECT r.id_route,
```

```

ROUND((EXTRACT(EPOCH FROM s.time_arrival) - EXTRACT(EPOCH FROM
s.time_departure))/3600 * p.fuel_rate, 0) AS fuel_consumption
FROM schema.route r
JOIN schema.schedule s ON r.id_schedule = s.id_schedule
JOIN schema.plane pl ON r.id_plane = pl.id_plane
JOIN schema.model p ON pl.id_model = p.id_model;
END;
$$;

```

Query Query History

1 SELECT * from schema.display_fuel_consumption_table()

Data Output Messages Notifications

	id_route integer	fuel_consumption numeric
1	5	750
2	1	750
3	4	1750
4	2	2100
5	6	1000
6	3	1200

2. Модифицировать триггер

create or replace function fn_check_time_punch() returns trigger as \$psql\$ begin

```

if
new.is_out_punch = (select tps.is_out_punch from time_punch tps
where tps.employee_id = new.employee_id order by tps.id desc limit 1 )
or
new.punch_time>now()
or
new.punch_time <= (select tps.punch_time from time_punch tps
where tps.employee_id = new.employee_id order by tps.id desc limit 1 )

```

```

then return null;
end if; return new;
end;

```

\$psql\$ language plpgsql;

drop trigger if exists check_time_punch on time_punch; create trigger check_time_punch

before insert on time_punch for each row

execute procedure fn_check_time_punch();

```
emp_time=# INSERT INTO time_punch (employee_id, is_out_punch, punch_time)
emp_time=# VALUES
emp_time=#      (2, false, '2021-01-01 10:00:00'),
emp_time=#      (2, true, '2021-01-01 11:30:00');
INSERT 0 2
emp_time=# INSERT INTO time_punch (employee_id, is_out_punch, punch_time)
emp_time=# VALUES
emp_time=#      (2, false, '2021-01-02 14:00:00'),
emp_time=#      (2, false, '2021-01-02 15:00:00');
INSERT 0 1
```

```
emp_time=# INSERT INTO time_punch (employee_id, is_out_punch, punch_time)
emp_time=# VALUES
emp_time=#      (1, false, '2023-10-15 17:00:00'),
emp_time=#      (1, true, '2023-10-15 10:30:00');
INSERT 0 0
```

```
emp_time=# INSERT INTO time_punch (employee_id, is_out_punch, punch_time)
emp_time=# VALUES
emp_time=#      (2, false, '2023-12-15 10:30:00');
INSERT 0 0
emp_time=#
```

```
emp_time=# select * from time_punch;
 id | employee_id | is_out_punch |      punch_time
-----+-----+-----+-----
  1 |           1 | f            | 2021-01-01 10:00:00
  2 |           1 | t            | 2021-01-01 11:30:00
  3 |           1 | f            | 2023-10-15 10:00:00
  4 |           1 | t            | 2023-10-15 17:30:00
  5 |           1 | f            | 2021-01-02 14:00:00
  7 |           1 | t            | 2021-01-02 16:00:00
  8 |           1 | f            | 2023-10-15 17:00:00
  9 |           1 | t            | 2023-10-15 10:30:00
 10 |           1 | f            | 2023-10-15 17:00:00
 11 |           1 | t            | 2023-10-15 17:00:00
 12 |           3 | f            | 2023-03-04 10:00:00
 13 |           3 | t            | 2023-03-04 15:00:00
 20 |           2 | f            | 2021-01-01 10:00:00
 21 |           2 | t            | 2021-01-01 11:30:00
 22 |           2 | f            | 2021-01-02 14:00:00
 24 |           2 | t            | 2023-10-15 10:30:00
(16 строк)
```

3. Создать необходимый триггер

```
CREATE OR REPLACE FUNCTION update_ticket_status()
RETURNS TRIGGER AS $$
BEGIN
    IF NEW.payment_status = 'Paid' AND OLD.payment_status <> 'Paid' THEN
        UPDATE schema.ticket
        SET status = 'Purchased'
        WHERE id_ticket = NEW.id_ticket;
    END IF;
    RETURN NEW;
END;
$$ LANGUAGE plpgsql;
```

```
CREATE TRIGGER ticket_payment_status_trigger
AFTER UPDATE ON schema.ticket
FOR EACH ROW
EXECUTE FUNCTION update_ticket_status();
```

	id_ticket [PK] integer	status character varying (20)	payment_status character varying (20)	service_class character varying (20)	baggage_insurance character varying (20)
1	1	Purchased	Paid	Business	Yes
2	2	Purchased	Paid	Economy	No
3	3	Available	Not Paid	Business	Yes
4	4	Booked	Not Paid	Business	No

Copy Copy to Query Editor

```
UPDATE schema.ticket
SET payment_status = 'Paid'
Where id_ticket = 4
```

Messages

Query returned successfully in 182 msec.

	id_ticket [PK] integer	status character varying (20)	payment_status character varying (20)	service_class character varying (20)	baggage_insurance character varying (20)
1	1	Purchased	Paid	Business	Yes
2	2	Purchased	Paid	Economy	No
3	3	Available	Not Paid	Business	Yes
4	4	Purchased	Paid	Business	No

Вывод

В ходе лабораторной работы были успешно созданы и протестированы процедуры и триггеры в системе управления базами данных PostgreSQL. Работа позволила овладеть практическими навыками программирования в SQL и пониманием механизма триггеров.