

**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»

Обучающийся Архангельская Елизавета Павловна

Факультет прикладной информатики

Группа К3239

Направление подготовки 09.03.03 Прикладная информатика

Образовательная программа Мобильные и сетевые технологии 2023

Преподаватель Говорова Марина Михайловна

Санкт-Петербург
2025

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

Практическое задание:

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 2.2. 7 оригинальных триггеров - 7 баллов (max)

Задание 1. Создать хранимые процедуры:

- Для повышения цен в пригородные поезда на 20%.

```
create or replace procedure increase_suburban_prices_20()
```

```
language plpgsql
```

```
as $$
```

```
begin
```

```
    update flight
```

```
        set price = price * 1.2
```

```
        where id_train in (
```

```
            select train.id from train
```

```
            join station s1 on train.departure_station = s1.id
```

```
            join station s2 on train.arrival_station = s2.id
```

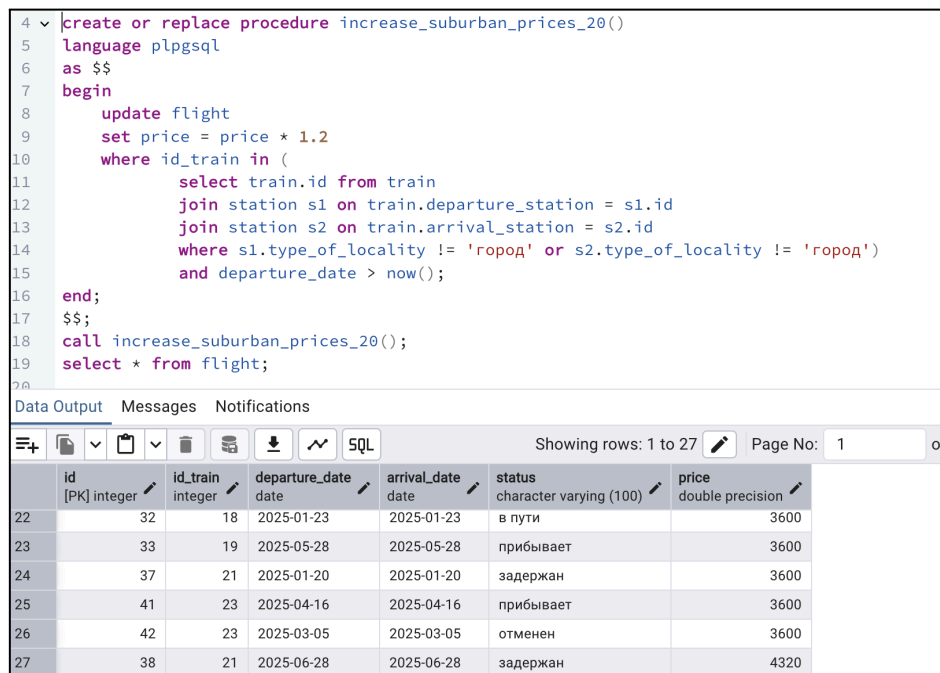
```
            where s1.type_of_locality != 'город' or s2.type_of_locality !=
```

```
'город')
```

```
            and departure_date > now();
```

```
end;
```

```
$$;
```



The screenshot displays a PostgreSQL database interface. At the top, a SQL query is shown in a text editor, which is the procedure `increase_suburban_prices_20()` defined in the previous blocks. Below the editor, the 'Data Output' tab is active, showing the results of a query. The table has 7 columns: `id` (integer, PK), `id_train` (integer), `departure_date` (date), `arrival_date` (date), `status` (character varying (100)), and `price` (double precision). The table contains 7 rows of data, with the last row showing a price of 4320.

	id [PK] integer	id_train integer	departure_date date	arrival_date date	status character varying (100)	price double precision
22	32	18	2025-01-23	2025-01-23	в пути	3600
23	33	19	2025-05-28	2025-05-28	прибывает	3600
24	37	21	2025-01-20	2025-01-20	задержан	3600
25	41	23	2025-04-16	2025-04-16	прибывает	3600
26	42	23	2025-03-05	2025-03-05	отменен	3600
27	38	21	2025-06-28	2025-06-28	задержан	4320

- Для создания нового рейса на поезд.

```
create or replace procedure create_new_flight(id_train integer,  
departure_date date, arrival_date date, status text)  
language plpgsql  
as $$  
begin  
    insert into flight (id_train, departure_date, arrival_date,status)  
        values (id_train, departure_date, arrival_date, status);  
end;  
$$;
```

Query	Query History
1	create or replace procedure create_new_flight(id_train integer,
2	departure_date date, arrival_date date, status text)
3	language plpgsql
4	as \$\$
5	begin
6	insert into flight (id_train, departure_date, arrival_date,status)
7	values (id_train, departure_date, arrival_date, status);
8	end;
9	\$\$;
10	call create_new_flight(10, '2025-05-30', '2025-05-30', 'отправлен');
11	
12	
13	

Data Output	Messages	Notifications
CALL		
Query returned successfully in 96 msec.		

- Для формирования общей выручки по продаже билетов за сутки.

```
create or replace function get_total_sum_per_day(sale_date date)
returns double precision as $$
declare total_sum double precision;
begin
    select sum(price)::double precision into total_sum
    from ticket
    where purchase_date::date = sale_date;
    return total_sum;
end;
$$ language plpgsql;
```

The screenshot shows a SQL IDE interface. The top pane contains SQL code for creating a function and executing it. The bottom pane shows the results of the execution.

```
1 create or replace function get_total_sum_per_day(sale_date DATE)
2 returns double precision as $$
3 declare total_sum double precision;
4 begin
5     select sum(price)::double precision into total_sum
6     from ticket
7     where purchase_date::date = sale_date;
8     return total_sum;
9 end;
10 $$ language plpgsql;
11
12 select get_total_sum_per_day('2025-02-20');
13
14 |
15
```

Below the code editor, there are tabs for "Data Output", "Messages", and "Notifications". The "Data Output" tab is active, showing a table with one row of results.

get_total_sum_per_day	
double precision	
1	143613.133

Задание 2. Создать 7 оригинальных триггеров - 7 баллов (max)

- проверка, что место не куплено несколько раз
create or replace function check_seat_availability()
returns trigger as \$\$
begin
 if exists (
 select 1 from ticket
 where id_flight = new.id_flight
 and id_seat = new.id_seat
 and ticket.status in ('оплачен', 'забронирован')
) then
 raise exception 'seat is already taken.';
 end if;

 return new;
end;
\$\$ language plpgsql;

create or replace trigger prevent_double_booking
before insert on ticket
for each row
execute function check_seat_availability();

Query	Query History
1	create or replace function check_seat_availability()
2	returns trigger as \$\$
3	begin
4	if exists (
5	select 1 from ticket
6	where id_flight = new.id_flight
7	and id_seat = new.id_seat
8	and ticket.status in ('оплачен', 'забронирован')
9) then raise exception 'seat is already taken.';
10	end if;
11	return new;
12	end;
13	\$\$ language plpgsql;
14	
15	create trigger prevent_double_booking
16	before insert on ticket
17	for each row
18	execute function check_seat_availability();
19	
20	insert into ticket (id_flight, datetime_departure, datetime_arrival, dropoff_point,
21	status, id_seat, boarding_point, price, base_price, purchase_date, id_passport)
22	values(2, '2025-03-17 18:02:00', '2025-03-17 22:06:00',
23	27, 'забронирован', 1611, 18, 15937.25, 12250, '2025-03-17 22:06:00', 7);
24	
Data Output Messages Notifications	
ERROR: seat is already taken.	
CONTEXT: PL/pgSQL function check_seat_availability() line 9 at RAISE	

- *при покупке статус кассы открыт*

create or replace function ticket_office_is_open()

returns trigger as \$\$

begin

if exists (

select 1 from box_office

where id = new.id_box_office

and box_office.status != 'открыта'

) then raise exception 'box_office is closen';

end if;

return new;

end;

\$\$ language plpgsql;

create or replace trigger check_box_office_open_before_insert

before insert on ticket

for each row

execute function ticket_office_is_open();

Query	Query History
1	create or replace function ticket_office_is_open()
2	returns trigger as \$\$
3	begin
4	if exists (
5	select 1 from box_office
6	where id = new.id_box_office
7	and box_office.status != 'открыта'
8) then raise exception 'box_office is closen';
9	end if;
10	return new;
11	end;
12	\$\$ language plpgsql;
13	
14	create or replace trigger check_box_office_open_before_insert
15	before insert on ticket
16	for each row
17	execute function ticket_office_is_open();
18	
19	insert into ticket (id_flight, datetime_departure, datetime_arrival, dropoff_point,
20	status, id_seat, boarding_point, price, base_price, purchase_date, id_passport, id_box_office)
21	values(5, '2025-03-17 18:02:00', '2025-03-17 22:06:00',
22	27, 'забронирован', 1670, 18, 15937.25, 12250, '2025-03-17 22:06:00', 7, 317);
23	

Data Output	Messages	Notifications
ERROR: box_office is closen		
CONTEXT: PL/pgSQL function ticket_office_is_open() line 7 at RAISE		
SQL state: P0001		

- *предыдущая остановка не раньше следующей по времени*

create or replace function check_time()

returns trigger as \$\$

begin

if exists (

select arrival_time from flight_stops

where id_train = new.id_train and number_of_stops_in_turn - 1 =
new.number_of_stops_in_turn) and (select arrival_time from flight_stops

where id_train = new.id_train and number_of_stops_in_turn - 1 =
new.number_of_stops_in_turn) > new.departure_time

then raise exception 'departure time must be after the previous stop arrival
time' ;

end if;

return new;

end;

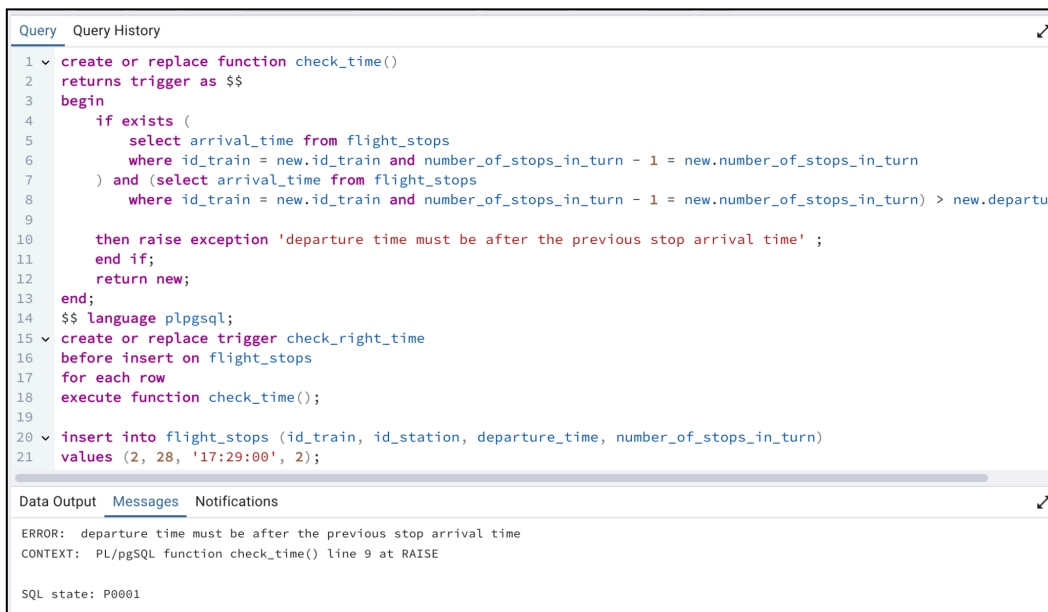
\$\$ language plpgsql;

create or replace trigger check_right_time

before insert on flight_stops

for each row

execute function check_time();



The screenshot shows a database IDE with a query editor and a messages pane. The query editor contains the following SQL code:

```
1 create or replace function check_time()
2 returns trigger as $$
3 begin
4     if exists (
5         select arrival_time from flight_stops
6         where id_train = new.id_train and number_of_stops_in_turn - 1 = new.number_of_stops_in_turn
7     ) and (select arrival_time from flight_stops
8         where id_train = new.id_train and number_of_stops_in_turn - 1 = new.number_of_stops_in_turn) > new.departu
9
10    then raise exception 'departure time must be after the previous stop arrival time' ;
11    end if;
12    return new;
13 end;
14 $$ language plpgsql;
15 create or replace trigger check_right_time
16 before insert on flight_stops
17 for each row
18 execute function check_time();
19
20 insert into flight_stops (id_train, id_station, departure_time, number_of_stops_in_turn)
21 values (2, 28, '17:29:00', 2);
```

The messages pane shows an error message:

```
ERROR: departure time must be after the previous stop arrival time
CONTEXT: PL/pgSQL function check_time() line 9 at RAISE
SQL state: P0001
```

- *запрет на продажу билетов после отправления поезда*

```

create or replace function valid_buying_date()
returns trigger as $$
declare
    train_departure_timestamp timestamp;
begin
    select flight.departure_date + flight_stops.departure_time
    into train_departure_timestamp
    from flight
    join flight_stops on flight.id_train = flight_stops.id_train
    where flight.id = new.id_flight
    and flight_stops.id_station = new.boarding_point;
    if train_departure_timestamp < new.purchase_date then
        raise exception 'incorrect purchase_date';
    end if;
    return new;
end;
$$ language plpgsql;

create or replace trigger check_valid_buying_date
before insert on ticket
for each row
execute function valid_buying_date();

```

```

16 create or replace function valid_buying_date()
17 returns trigger as $$
18 declare
19     train_departure_timestamp timestamp;
20 begin
21     select flight.departure_date + flight_stops.departure_time
22     into train_departure_timestamp
23     from flight
24     join flight_stops on flight.id_train = flight_stops.id_train
25     where flight.id = new.id_flight
26     and flight_stops.id_station = new.boarding_point;
27     if train_departure_timestamp < new.purchase_date then
28         raise exception 'incorrect purchase_date';
29     end if;
30     return new;
31 end;
32 $$ language plpgsql;
33
34 create or replace trigger check_valid_buying_date
35 before insert on ticket
36 for each row
37 execute function valid_buying_date();
38 insert into ticket (id_flight, dropoff_point, status,
39                     id_seat, boarding_point, price, base_price, purchase_date, id_passport)
40 values (2, 18, 'отменен', 1681, 27, 10000, 10000, now(), 7 );

```

Data Output [Messages](#) Notifications

ERROR: incorrect purchase_date
CONTEXT: PL/pgSQL function valid_buying_date() line 12 at RAISE

- *запрет на продажу билета на рейс, который отменили*

```
create or replace function valid_status()
returns trigger as $$
begin
    if new.status = 'отменен' then
        raise exception 'этот рейс отменен';
    end if;
    return new;
end;
$$ language plpgsql;

create or replace trigger check_valid_status
before insert on ticket
for each row
execute function valid_status();
```

```
2  ✓ create or replace function valid_status()
3    returns trigger as $$
4    begin
5        if new.status = 'отменен' then
6            raise exception 'этот рейс отменен';
7        end if;
8        return new;
9    end;
10   $$ language plpgsql;
11
12
13
14  ✓ create or replace trigger check_valid_status
15    before insert on ticket
16    for each row
17    execute function valid_status();
18  ✓ insert into ticket (id_flight, dropoff_point, status,
19                        id_seat, boarding_point, price, base_price, purchase_date, id_passport)
```

Data Output Messages Notifications

ERROR: этот рейс отменен
CONTEXT: PL/pgSQL function valid_status() line 4 at RAISE

SQL state: P0001

- не допускать удаление станции, если через неё проходят рейсы

```
create or replace function used_station()
returns trigger as $$
begin
    if exists (select 1 from flight_stops where id_station = old.id)
    then raise exception 'эту станцию удалить нельзя, через нее проходят поезда';
    end if;
    return old;
end;
$$ language plpgsql;

create or replace trigger before_delete_station_check
before delete on station
for each row
execute function used_station();
```

Query	Query History
1	create or replace function used_station()
2	returns trigger as \$\$
3	begin
4	if exists (select 1 from flight_stops where id_station = old.id)
5	then raise exception 'эту станцию удалить нельзя, через нее проходят поезда';
6	end if;
7	return old;
8	end;
9	\$\$ language plpgsql;
10	
11	
12	create or replace trigger before_delete_station_check
13	before delete on station
14	for each row
15	execute function used_station();
16	
17	delete from station
18	where id = 28;
19	
Data Output Messages Notifications	
ERROR: эту станцию удалить нельзя, через нее проходят поезда	
CONTEXT: PL/pgSQL function used_station() line 4 at RAISE	

- станция прибытия и отправления не могут быть одинаковыми

```
create or replace function check_departure_arrival_different()
returns trigger as $$
begin
    if new.departure_station = new.arrival_station
    then raise exception 'станция прибытия и отправления не могут быть
одинаковыми';
    end if;
    return new;
end;
$$ language plpgsql;
```

```
create or replace trigger before_insert_or_update_flight_check_stations
before insert or update on train
for each row
execute function check_departure_arrival_different();
```

Query	Query History
<pre> 1 create or replace function check_departure_arrival_different() 2 returns trigger as \$\$ 3 begin 4 if new.departure_station = new.arrival_station 5 then raise exception 'станция прибытия и отправления не могут быть одинаковыми'; 6 end if; 7 return new; 8 end; 9 \$\$ language plpgsql; 10 11 create or replace trigger before_insert_or_update_flight_check_stations 12 before insert or update on train 13 for each row 14 execute function check_departure_arrival_different(); 15 16 update train 17 set arrival_station = 28 18 where id = 2;</pre>	
Data Output	Messages
<pre> ERROR: станция прибытия и отправления не могут быть одинаковыми CONTEXT: PL/pgSQL function check_departure_arrival_different() line 4 at RAISE SQL state: P0001</pre>	

Выводы:

В ходе выполнения лабораторной работы были созданы и протестированы процедуры, функции и триггеры для индивидуальной БД. Были реализованы 3 процедуры с использованием входных параметров, также разработано 7 оригинальных триггеров, которые обеспечивают проверку целостности данных. В результате лабораторной работы были получены практические навыки по созданию и использованию процедур, функций и триггеров в СУБД PostgreSQL, что является важным элементом проектирования надежных и устойчивых к ошибкам баз данных.