

Министерство науки и высшего образования Российской Федерации  
федеральное государственное автономное образовательное  
учреждение высшего образования  
“Национальный исследовательский университет ИТМО”

Факультет инфокоммуникационных технологий

## **ЛАБОРАТОРНАЯ РАБОТА №2**

**Запросы на выборку и модификацию данных,  
представления и индексы в PostgreSQL  
по дисциплине:  
«Проектирование и реализация баз данных»**

**Выполнил студент:**

Тюмин Никита Сергеевич

Группа №K32402

**Преподаватель:**

Говорова Марина Михайловна

Санкт-Петербург  
2023

## Цель работы:

Овладеть практическими создания и использования процедур, функций и триггеров в базе данных PostgreSQL.

## Программное обеспечение:

СУБД PostgreSQL 14, pgAdmin 4.

## Практическое задание:

1. Создать процедуры/функции согласно индивидуальному заданию
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.

## Ход работы:

Работа проводилась в созданной ранее базе данных, ER диаграмма представлена на Рисунке 1.

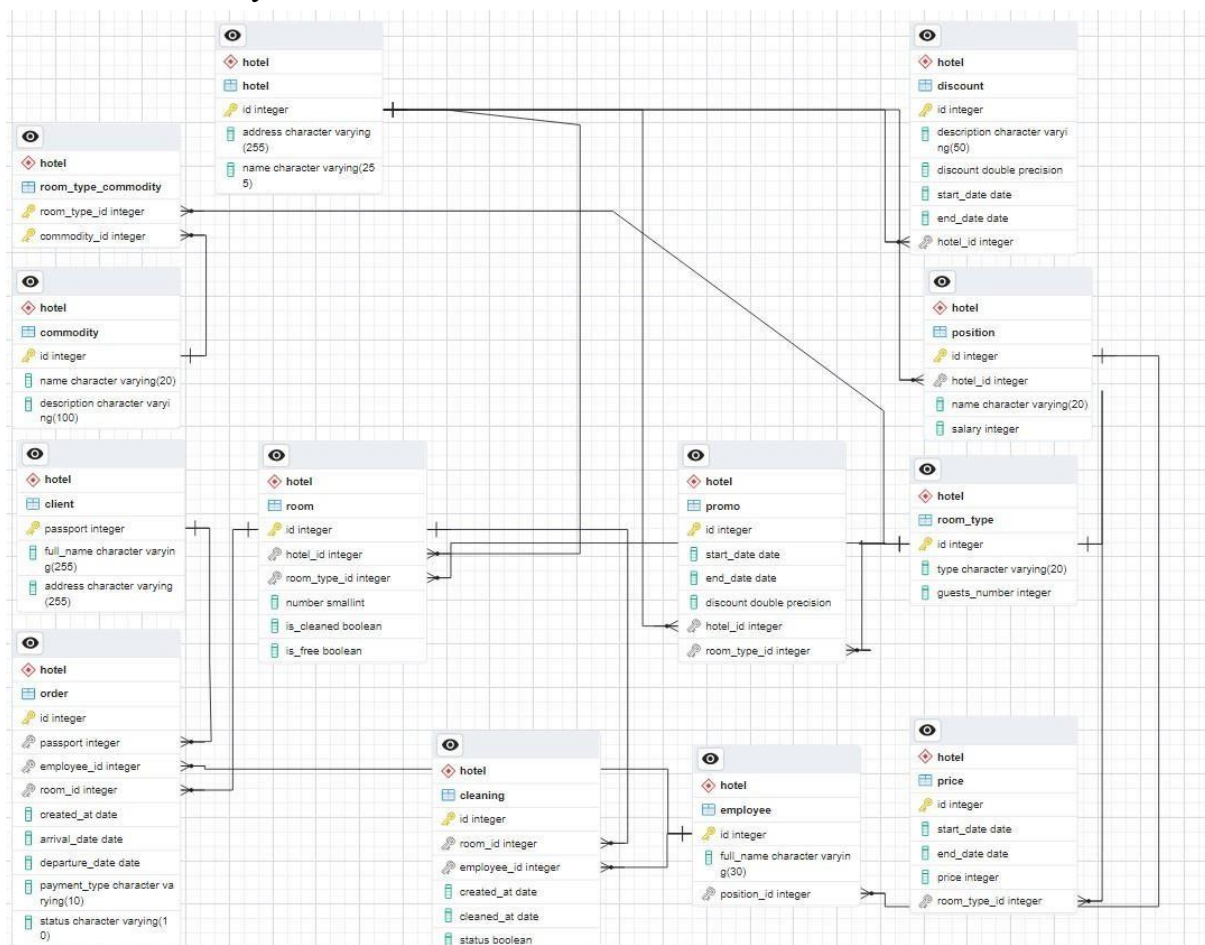


Рисунок 1 – ER диаграмма базы данных

## Создание процедур/функций:

Увеличение цены всех номеров на 5 %, если в отеле нет свободных номеров

Query Query History

```
1
2 CREATE OR REPLACE PROCEDURE
3 increase_price_if_all_occupied(hotelId INTEGER)
4 LANGUAGE plpgsql
5 AS $$
6 DECLARE
7     rooms_free INTEGER;
8 BEGIN
9     SELECT count(*) INTO rooms_free from (
10         SELECT h.id, r.id
11         FROM hotel.hotel h
12         JOIN hotel.room r
13         ON r.hotel_id = h.id
14         WHERE r.is_free IS true
15         AND h.id = hotelId
16     ) as foo;
17 if rooms_free = 0 THEN
18     UPDATE hotel.price
19     SET price = price * 1.05
20     WHERE price.id in (
```

Data Output Messages Notifications

CREATE PROCEDURE

Query returned successfully in 63 msec.

Total rows: 3 of 3 Query complete 00:00:00.063

✓ Query returned successfully in 63 msec. ✕

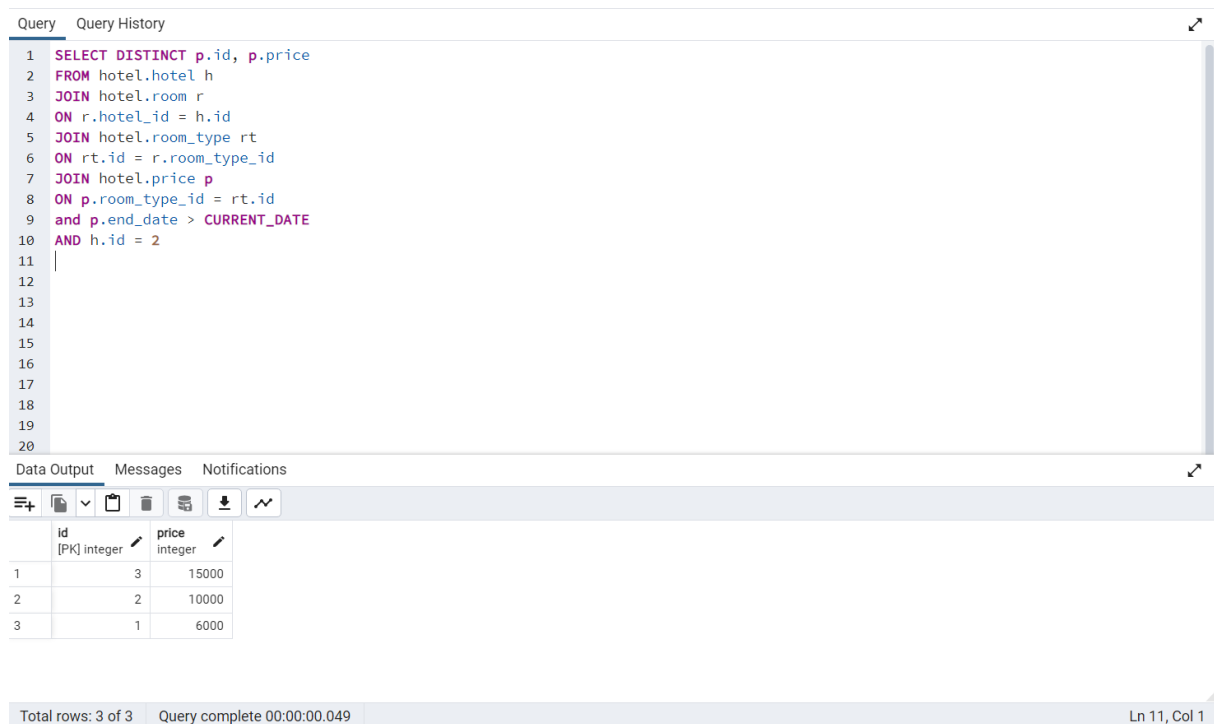
```
CREATE OR REPLACE PROCEDURE
increase_price_if_all_occupied(hotelId INTEGER)
LANGUAGE plpgsql
AS $$
DECLARE
    rooms_free INTEGER;
BEGIN
    SELECT count(*) INTO rooms_free from (
        SELECT h.id, r.id
        FROM hotel.hotel h
        JOIN hotel.room r
        ON r.hotel_id = h.id
        WHERE r.is_free IS true
        AND h.id = hotelId
    ) as foo;
    if rooms_free = 0 THEN
        UPDATE hotel.price
        SET price = price * 1.05
        WHERE price.id in (
            SELECT distinct pr.id
            FROM hotel.hotel h
            JOIN hotel.room r
            ON r.hotel_id = h.id
            JOIN hotel.room_type rt
            ON rt.id = r.room_type_id
            JOIN hotel.price pr
```

```

        ON pr.room_type_id = rt.id
        and pr.end_date > CURRENT_DATE
        AND h.id = 2
    );
END IF;
end;
$$;

```

До запроса:



The screenshot shows a database query editor with a query window and a results window. The query window contains a SQL query that selects distinct hotel IDs and prices, filtered by room type and end date. The results window shows three rows of data.

```

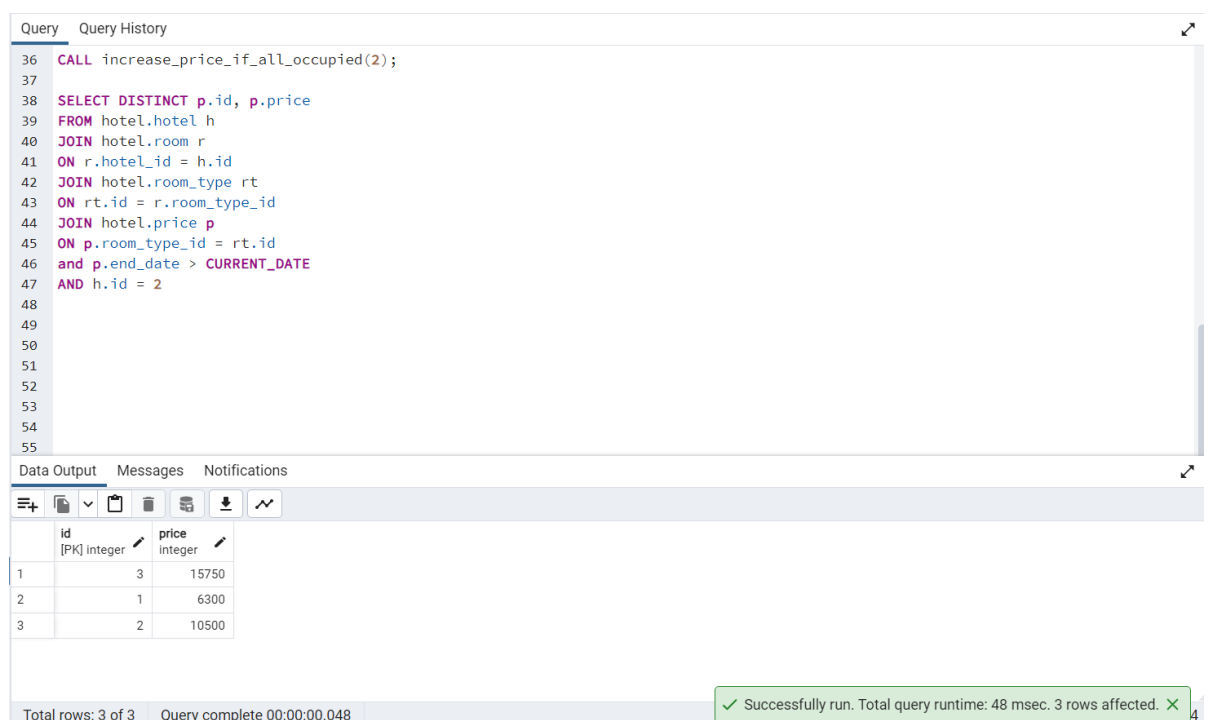
1  SELECT DISTINCT p.id, p.price
2  FROM hotel.hotel h
3  JOIN hotel.room r
4  ON r.hotel_id = h.id
5  JOIN hotel.room_type rt
6  ON rt.id = r.room_type_id
7  JOIN hotel.price p
8  ON p.room_type_id = rt.id
9  and p.end_date > CURRENT_DATE
10 AND h.id = 2
11
12
13
14
15
16
17
18
19
20

```

	id [PK] integer	price integer
1	3	15000
2	2	10000
3	1	6000

Total rows: 3 of 3    Query complete 00:00:00.049    Ln 11, Col 1

После:



The screenshot shows a database query editor with a query window and a results window. The query window contains a SQL query that first calls a stored procedure to increase prices and then selects distinct hotel IDs and prices, filtered by room type and end date. The results window shows three rows of data.

```

36 CALL increase_price_if_all_occupied(2);
37
38 SELECT DISTINCT p.id, p.price
39 FROM hotel.hotel h
40 JOIN hotel.room r
41 ON r.hotel_id = h.id
42 JOIN hotel.room_type rt
43 ON rt.id = r.room_type_id
44 JOIN hotel.price p
45 ON p.room_type_id = rt.id
46 and p.end_date > CURRENT_DATE
47 AND h.id = 2
48
49
50
51
52
53
54
55

```

	id [PK] integer	price integer
1	3	15750
2	1	6300
3	2	10500

Total rows: 3 of 3    Query complete 00:00:00.048    Successfully run. Total query runtime: 48 msec. 3 rows affected.    Ln 47, Col 1

Получение информации о свободных одноместных номерах отеля на завтрашний день;

Query
Query History

```

1 CREATE OR REPLACE FUNCTION
2 get_free_onerooms_for_tommorow(hotelId INTEGER) RETURNS TABLE (id integer)
3 LANGUAGE plpgsql
4 AS $$
5 DECLARE
6 BEGIN
7     RETURN QUERY SELECT r.id
8     FROM hotel.room r
9     JOIN hotel.room_type rt
10    ON r.room_type_id = rt.id
11    LEFT JOIN hotel.order o
12    ON o.room_id = r.id
13    WHERE r.room_type_id IN (

```

Data Output
Messages
Notifications

	id integer
1	2
2	10
3	5
4	8
5	6
6	4
7	3
8	9
9	7

Total rows: 9 of 9
Query complete 00:00:00.081
Ln 1, Col 1

```

CREATE OR REPLACE FUNCTION
get_free_onerooms_for_tommorow(hotelId INTEGER) RETURNS TABLE (id integer)
LANGUAGE plpgsql
AS $$
DECLARE
BEGIN
    RETURN QUERY SELECT r.id
    FROM hotel.room r
    JOIN hotel.room_type rt
    ON r.room_type_id = rt.id
    LEFT JOIN hotel.order o
    ON o.room_id = r.id
    WHERE r.room_type_id IN (
        SELECT DISTINCT rt.id
        FROM hotel.room r
        JOIN hotel.room_type rt
        ON r.room_type_id = rt.id
        WHERE rt.guests_number = 1
    )
    AND r.id NOT IN (
        SELECT r.id
        FROM hotel.room r
        JOIN hotel.room_type rt
        ON r.room_type_id = rt.id
        LEFT JOIN hotel.order o
        ON o.room_id = r.id
        WHERE r.room_type_id IN (
            SELECT DISTINCT rt.id

```

```

        FROM hotel.room r
        JOIN hotel.room_type rt
        ON r.room_type_id = rt.id
        WHERE rt.guests_number = 1
    )
    AND r.hotel_id = hotelId
    AND CURRENT_DATE + interval '1 day' >= o.arrival_date
    AND CURRENT_DATE + interval '1 day' <= o.departure_date
)
AND r.hotel_id = hotelId;

END;
$$;

select * from get_free_onerooms_for_tomorrow(2);

```

Бронирование двухместного номера в гостинице на заданную дату и количество дней проживания

Query
Query History

```

1 CREATE OR REPLACE PROCEDURE
2 book_a_number(daysNumber INTEGER, arrivalDate DATE, hotelId INTEGER, guestId INTEGER, empId INTEGER)
3 LANGUAGE plpgsql
4 AS $$
5 DECLARE
6     roomId INTEGER;
7     departureDate DATE;
8 BEGIN
9     CREATE TEMP TABLE bookedRooms ON COMMIT DROP AS
10    SELECT DISTINCT o.room_id
11    FROM hotel.order o
12    WHERE o.arrival_date BETWEEN '2024-12-01' AND ('2024-12-01'::date + interval '1 day')
13    OR o.departure_date BETWEEN '2024-12-01' AND ('2024-12-01'::date + interval '1 day')
14    OR ('2024-12-01' BETWEEN o.arrival_date AND o.departure_date AND ('2024-12-01'::date + interval '1 day') BETWEEN o.arrival_date
15

```

Data Output
Messages
Notifications

	id [PK] integer	passport integer	employee_id integer	room_id integer	created_at date	arrival_date date	departure_date date	payment_type character varying (10)	status character varying (10)
6	7	1167417325	1	1	2023-03-11	2023-03-11	2023-06-15	card	processed
7	8	1167417325	1	2	2023-03-11	2023-03-11	2023-04-24	card	processed
8	9	1167417325	1	11	2023-04-24	2024-11-30	2024-12-05	card	processed
9	10	1167417325	1	12	2023-04-24	2024-11-30	2024-12-05	card	processed
10	11	1167417325	1	13	2023-04-24	2024-11-30	2024-12-05	card	processed
11	12	1167417325	1	14	2023-04-24	2024-11-30	2024-12-05	card	processed
12	13	1167417325	1	15	2023-04-24	2024-11-30	2024-12-05	card	processed
13	14	1167417325	1	16	2023-04-24	2024-11-30	2024-12-05	card	processed
14	15	1167417325	1	17	2023-04-24	2024-11-30	2024-12-05	card	processed

Total rows: 14 of 14
Query complete 00:00:00.092
Rows selected: 7
Ln 1, Col 1

```

CREATE OR REPLACE PROCEDURE
book_a_number(daysNumber INTEGER, arrivalDate DATE, hotelId INTEGER, guestId
INTEGER, empId INTEGER)
LANGUAGE plpgsql
AS $$
DECLARE
    roomId INTEGER;
    departureDate DATE;
BEGIN

```

```

CREATE TEMP TABLE bookedRooms ON COMMIT DROP AS
SELECT DISTINCT o.room_id
FROM hotel.order o
WHERE o.arrival_date BETWEEN '2024-12-01' AND ('2024-12-01'::date +
interval '1 day')
OR o.departure_date BETWEEN '2024-12-01' AND ('2024-12-01'::date +
interval '1 day')
OR ('2024-12-01' BETWEEN o.arrival_date AND o.departure_date AND ('2024-
12-01'::date + interval '1 day') BETWEEN o.arrival_date AND o.departure_date);

SELECT r.id into roomId
FROM hotel.hotel h
JOIN hotel.room r
ON h.id = r.hotel_id
JOIN hotel.room_type rt
ON r.room_type_id = rt.id
WHERE rt.guests_number = 2
AND r.hotel_id = hotelId
AND r.id NOT IN (SELECT * FROM bookedRooms)
LIMIT 1;

IF roomId IS NOT null THEN
    INSERT INTO hotel.order (passport, employee_id, room_id, created_at,
arrival_date, departure_date, payment_type, status)
VALUES
    (guestId, empId, roomId, CURRENT_DATE, arrivalDate, arrivalDate +
(daysNumber::text || ' day')::INTERVAL, 'card', 'processed');
END IF;

END;
$$;

CALL book_a_number(5, '30-11-2024', 2, 1167417325, 1);
select * from hotel.order;

```

**Создание триггера для логирования вставки/изменения/удаления записей в таблице заказов.**

Создание триггера:

```
Query  Query History
19      msg := 'Deleted order (' || NEW.id || ')';
20      INSERT INTO order_logs(created_at, "message") values (CURRENT_DATE, msg);
21      RETURN OLD;
22  END IF;
23  END;
24  $$ LANGUAGE plpgsql;
25
26
27  CREATE TRIGGER t_order AFTER INSERT OR UPDATE OR DELETE ON hotel.order FOR EACH ROW EXECUTE PROCEDURE add_to_log();
28
29

Data Output  Messages  Notifications
CREATE TRIGGER
Query returned successfully in 51 msec.

Total rows: 14 of 14  Query complete 00:00:00.051  Ln 19, Col 45
```

```
CREATE TABLE IF NOT EXISTS order_logs (
    created_at DATE,
    "message" TEXT
);
```

```
CREATE OR REPLACE FUNCTION add_to_log() RETURNS TRIGGER AS $$
DECLARE
    msg varchar(254);
BEGIN
    IF TG_OP = 'INSERT' THEN
        msg := 'New order (' || NEW.id || ') from ' || NEW.passport;
        INSERT INTO order_logs(created_at, "message") values (CURRENT_DATE,
msg);
        RETURN NEW;
    ELSIF TG_OP = 'UPDATE' THEN
        msg := 'Order (' || NEW.id || ') updated';
        INSERT INTO order_logs(created_at, "message") values (CURRENT_DATE,
msg);
        RETURN NEW;
    ELSIF TG_OP = 'DELETE' THEN
        msg := 'Deleted order (' || NEW.id || ')';
        INSERT INTO order_logs(created_at, "message") values (CURRENT_DATE,
msg);
        RETURN OLD;
    END IF;
END;
$$ LANGUAGE plpgsql;
```



```
CREATE TRIGGER t_order AFTER INSERT OR UPDATE OR DELETE ON hotel.order FOR EACH ROW EXECUTE PROCEDURE add_to_log();
```

Таблица с логами после вставки:

The screenshot shows a database query editor with a SQL script and its results. The SQL script consists of four statements: an INSERT into hotel.order, an UPDATE of status to 'cancelled' for id=20, a DELETE from hotel.order where id=20, and a SELECT from public.order\_logs. The results are shown in a table with two columns: 'created\_at' and 'message'. The table contains three rows of log data. The status bar at the bottom indicates 'Total rows: 3 of 3' and 'Query complete 00:00:00.046'.

```
28 INSERT INTO hotel.order (passport, employee_id, room_id, created_at, arrival_date, departure_date, payment_type, status) VALUES
29 (1167417325, 2, 16, CURRENT_DATE, CURRENT_DATE, CURRENT_DATE + interval '7 day', 'cash', 'processed');
30
31 UPDATE hotel.order SET status = 'cancelled' WHERE id = 20 ;
32
33 DELETE FROM hotel.order WHERE id = 20;
34
35 SELECT * FROM public.order_logs
36
37
38
39
40
41
42
43
44
45
46
```

	created_at date	message text
1	2023-04-24	New order (20) from 1167417325
2	2023-04-24	Order (20) updated
3	2023-04-24	Deleted order (20)

Total rows: 3 of 3    Query complete 00:00:00.046    Ln 29, Col 38

```
INSERT INTO hotel.order (passport, employee_id, room_id, created_at,
arrival_date, departure_date, payment_type, status) VALUES
(1167417325, 2, 16, CURRENT_DATE, CURRENT_DATE, CURRENT_DATE + interval '7
day', 'cash', 'processed');
```

```
UPDATE hotel.order SET status = 'cancelled' WHERE id = 20 ;
```

```
DELETE FROM hotel.order WHERE id = 20;
```

```
SELECT * FROM public.order_logs
```

## Выводы:

1. Были созданы процедуры и функции согласно индивидуальному заданию
2. Был создан триггер на различные операции со строками.