

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

Факультет инфокоммуникационных технологий

ЛАБОРАТОРНАЯ РАБОТА №3

ПРОЦЕДУРЫ, ФУНКЦИИ, ТРИГГЕРЫ В PostgreSQL

по дисциплине:

«Проектирование и реализация баз данных»

Выполнил студент:

Старовойтова Елизавета Анатольевна

Группа №K32402

Преподаватель:

Говорова Марина Михайловна

Санкт-Петербург

2023

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

Программное обеспечение: СУБД PostgreSQL, SQL Shell (psql).

Практическое задание:

Вариант 1

1. Создать процедуры/функции 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.

Индивидуальное задание (вариант):

Вариант 1. БД «Отель»

Описание предметной области: Отели сети находятся в разных городах. Цены на номера одного типа во всех отелях одинаковы и зависят от типа номера и количества мест. Номер может быть забронирован, занят или свободен. При заезде в отель постояльцы проходят регистрацию. Информация о регистрации постояльцев отеля (выехавших из отеля) хранится в течение года и 1 января удаляется в архив.

Номера ежедневно убираются горничными, для чего составляется график уборки номеров. Ежедневно каждому номеру присваивается статус “убран”, “не убран”.

Цены на номера могут меняться.

БД должна содержать следующий минимальный набор сведений: Адрес отеля.

Название отеля. Номер комнаты. Тип комнаты. Количество мест. Удобства.

Цена комнаты за сутки проживания. Имя постояльца. Фамилия постояльца.

Отчество постояльца. Адрес постоянного проживания. Дата заезда. Дата отъезда. График уборки номеров.

Дополнить исходные данные информацией: по бронированию комнаты; по сотруднику, который регистрирует постояльца в отеле в день заезда; по оплате проживания; по составу удобств в комнате; по акциям, доступным при бронировании (скидки).

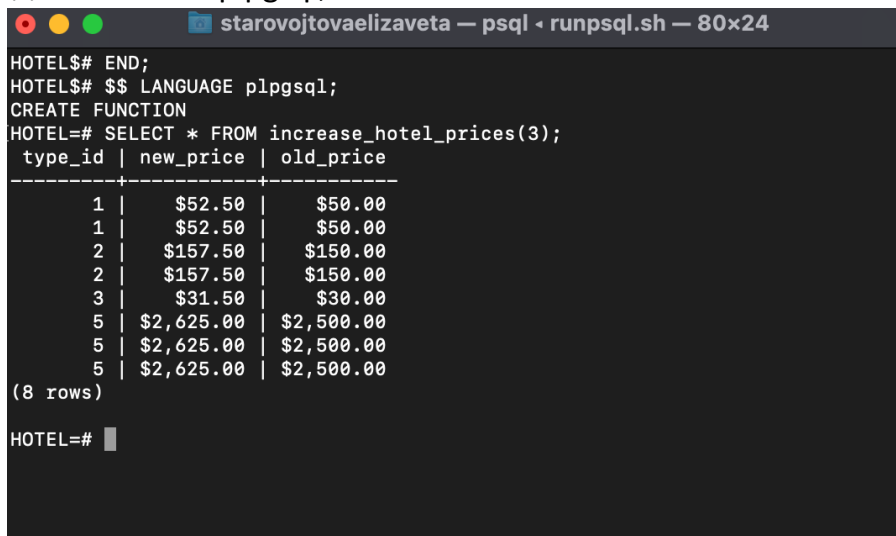
Выполнение:

Создайте хранимые процедуры/функции:

1. Для увеличения цены номеров на 5%, если в отеле нет свободных номеров.

```
CREATE OR REPLACE FUNCTION increase_hotel_prices(hotel_id INT)
RETURNS TABLE (type_id INT, new_price MONEY, old_price MONEY) AS $$
BEGIN
    UPDATE price
    SET sum = sum * 1.05
    WHERE room_type IN (
        SELECT type_room.id_type
        FROM room
        INNER JOIN type_room ON room.type = type_room.id_type
        LEFT JOIN "order" ON room.id_room = "order".room
        AND "order".status_order IN ('approved', 'settlement')
        WHERE room.hotel = hotel_id
        AND "order".id_order IS NULL
    );
    RETURN QUERY SELECT type_room.id_type, price.sum * 1.05, price.sum
    FROM room
    INNER JOIN type_room ON room.type = type_room.id_type
    INNER JOIN price ON room.type = price.room_type
    WHERE room.hotel = hotel_id AND room.type = type_room.id_type;
END;
```

```
$$ LANGUAGE plpgsql;
```



The screenshot shows a terminal window titled "starovojtovaelizaveta — psql • runpsql.sh — 80x24". The user has entered the following commands:

```
HOTEL=# END;
HOTEL=# $$ LANGUAGE plpgsql;
CREATE FUNCTION
HOTEL=# SELECT * FROM increase_hotel_prices(3);
```

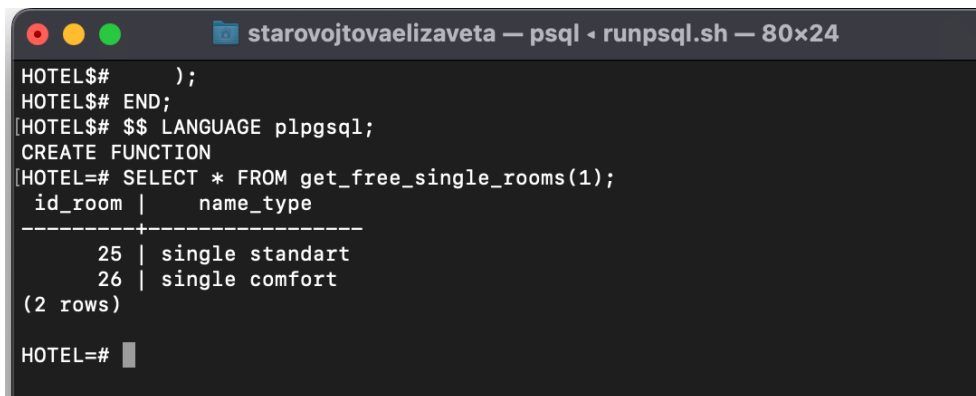
The result is a table with 8 rows:

type_id	new_price	old_price
1	\$52.50	\$50.00
1	\$52.50	\$50.00
2	\$157.50	\$150.00
2	\$157.50	\$150.00
3	\$31.50	\$30.00
5	\$2,625.00	\$2,500.00
5	\$2,625.00	\$2,500.00
5	\$2,625.00	\$2,500.00

The terminal also shows "(8 rows)" and "HOTEL=#" at the bottom.

2. Для получения информации о свободных одноместных номерах отеля на завтрашний день.

```
CREATE OR REPLACE FUNCTION get_free_single_rooms(hotel_id INTEGER)
RETURNS TABLE (id_room INTEGER, name_type VARCHAR(50))
AS $$
BEGIN
    RETURN QUERY
    SELECT room.id_room, type_room.name_type
    FROM room
    INNER JOIN type_room ON room.type = type_room.id_type
    WHERE room.hotel = hotel_id
    AND type_room.id_type IN (6, 7)
    AND NOT EXISTS (
        SELECT 1
        FROM "order"
        WHERE "order".room = room.id_room
        AND "order".status_order IN ('approved', 'settlement')
        AND "order".arrival_date <= CURRENT_DATE + 1
        AND "order".departure_date > CURRENT_DATE + 1
    );
END;
$$ LANGUAGE plpgsql;
```



```
HOTEL$#    );
HOTEL$# END;
HOTEL$# $$ LANGUAGE plpgsql;
CREATE FUNCTION
HOTEL=# SELECT * FROM get_free_single_rooms(1);
 id_room |   name_type
-----+-----
      25 | single standart
      26 | single comfort
(2 rows)

HOTEL=#
```

После добавления заказа на одноместный номер с id 25, выполнение будет следующим:



```
HOTEL=# SELECT * FROM get_free_single_rooms(1);
 id_room |   name_type
-----+-----
      26 | single comfort
(1 row)

HOTEL=#
```

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

```
CREATE OR REPLACE FUNCTION book_room(  
    room_id INTEGER,  
    client_arrival_date DATE,  
    num_days INTEGER,  
    client_name TEXT,  
    client_email VARCHAR(50),  
    client_passport VARCHAR(50),  
    client_address VARCHAR(50),  
    client_payment TEXT,  
    hotel_admin INTEGER  
)  
RETURNS INTEGER AS $$  
DECLARE  
    booking_id INTEGER;  
    client_id INTEGER;  
BEGIN  
    -- Получаем идентификатор клиента по имени и email  
    SELECT id_client INTO client_id FROM client WHERE full_name = client_name AND  
    passport_datas = client_passport;  
  
    -- Если клиент не найден, то создаем новую запись в таблице client  
    IF client_id IS NULL THEN  
        INSERT INTO client (address, passport_datas, full_name, email) VALUES (client_address,  
    client_passport, client_name, client_email) RETURNING id_client INTO client_id;  
    END IF;  
  
    -- Проверяем, что номер свободен на заданные даты  
    IF NOT EXISTS (  
        SELECT 1 FROM "order" o  
        WHERE o.room = room_id  
        AND o.status_order IN ('approved', 'settlement')  
        AND o.arrival_date <= client_arrival_date + num_days  
        AND o.departure_date >= client_arrival_date  
    ) THEN  
        -- Если номер свободен, то создаем новый заказ на бронирование номера  
        INSERT INTO "order" (client, room, arrival_date, departure_date, payment_type,  
    status_order, booking_date, employee)  
        VALUES (client_id, room_id, client_arrival_date, client_arrival_date + num_days,  
    client_payment, 'approved', CURRENT_DATE, hotel_admin)  
        RETURNING id_order INTO booking_id;
```

```

RETURN booking_id;
ELSE
-- Если номер занят, то возвращаем ноль
RETURN 0;
END IF;
END;
$$ LANGUAGE plpgsql;

```

```

CREATE FUNCTION
HOTEL=# SELECT * FROM book_room(26, '2023-05-12', 12, 'Kotov Ilya Petrovich', 'kotov@gmail.com', 'HB3452312', 'Opolchenia 13, Spb', 'mir pay', 3000);
book_room
-----
      26
(1 row)

HOTEL=#

```

Клиент также добавился

```

HOTEL=# SELECT * FROM client;

```

id_client	address	passport_datas	full_name	email
1	Lensovata 23	HB3121108	Starovoytova Elizaveta Anatolyovna	lstarovoytova17@gmail.com
2	Piskarevski 45	HB3121109	Starovoytova Ekaterina Anatolyovna	katyastarovoytova03@gmail.com
3	Karvata 9	HB3105213	Peskun Maria Vadimovna	mariapeskun@gmail.com
4	Pobedy 24	HB3111234	Yaroshenko Maria Genadievna	myarosh2004@gmail.com
5	Lomonosova 9, Ekb	4022 382190	Sinyta Anastasia Anatolyovna	sinyta@yandex.ru
6	Academica Bardina 44, Ekb	4022 382187	Petrosyan Nikita Andreevich	petrosyan@yandex.ru
7	Kosareva 8, Spb	4892 384187	Ivanov Ivan	ivanov@yandex.ru
8	Pushkina 8, Minsk	HB3456679	Ivanova Katya Olegovna	ivanova@yandex.ru
9	Orange 8, Gomel	HB3445577	Liju John Fedor	fedos@gmail.com
18	Opolchenia 13, Spb	HB3452312	Kotov Ilya Petrovich	kotov@gmail.com

```

(10 rows)

```

Заказ также создан успешно

```

HOTEL=# SELECT * FROM "order" WHERE client = 18;

```

id_order	client	employee	room	arrival_date	departure_date	payment_type	status_order	booking_date
26	18	3000	26	2023-05-12	2023-05-24	mir pay	approved	2023-05-07

```

(1 row)

HOTEL=#

```

Триггеры:

1. Триггер для контроля уникальности полей passport_datas и full_name при вставке и изменении данных в таблице client.

```

CREATE OR REPLACE FUNCTION unique_client()
RETURNS trigger AS $$
BEGIN
IF (SELECT COUNT(*) FROM client WHERE passport_datas = NEW.passport_datas AND
full_name = NEW.full_name AND id_client != NEW.id_client) > 0 THEN
RAISE EXCEPTION 'Client with these passport datas and full name already exists';
END IF;
RETURN NEW;
END;

```

```
$$ LANGUAGE plpgsql;
```

```
CREATE TRIGGER unique_client_trigger
BEFORE INSERT OR UPDATE ON client
FOR EACH ROW
EXECUTE FUNCTION unique_client();
```

```
HOTEL=# INSERT INTO client (address, passport_datas, full_name, email) VALUES ('Some Address', 'AB123456', 'John Doe', 'john.doe@example.com');
INSERT 0 1
HOTEL=# INSERT INTO client (address, passport_datas, full_name, email) VALUES ('Another Address', 'AB123456', 'John Doe', 'j.doe@example.com');
ERROR:  Client with these passport datas and full name already exists
CONTEXT:  PL/pgSQL function unique_client() line 4 at RAISE
HOTEL=#
```

2. Триггер, который проверяет, что для каждой комнаты в таблице cleaning не может быть двух уборок со статусом "clean" или "no clean".

```
CREATE OR REPLACE FUNCTION check_cleaning_status() RETURNS TRIGGER AS $$
BEGIN
    IF EXISTS (
        SELECT 1 FROM cleaning
        WHERE id_room = NEW.id_room
        AND status_clean = NEW.status_clean
        AND id_cleaning <> NEW.id_cleaning
    ) THEN
        RAISE EXCEPTION 'Cleaning with status % already exists for room %', NEW.status_clean,
NEW.id_room;
    END IF;
    RETURN NEW;
END;
$$ LANGUAGE plpgsql;
```

```
CREATE TRIGGER cleaning_status_trigger
BEFORE INSERT OR UPDATE ON cleaning
FOR EACH ROW
EXECUTE FUNCTION check_cleaning_status();
```

```
CREATE TRIGGER
HOTEL=# insert into cleaning (date_clean, status_clean, id_employee, id_room) values ('2023-05-07', 'clean', 3003, 25);
INSERT 0 1
HOTEL=# insert into cleaning (date_clean, status_clean, id_employee, id_room) values ('2023-05-07', 'clean', 3006, 24);
ERROR:  Cleaning with status clean already exists for room 24
CONTEXT:  PL/pgSQL function check_cleaning_status() line 9 at RAISE
HOTEL=#
```

Комментарий: Уборка для номера с идентификатором 24 и статусом “clean” уже существует, поэтому выдало ошибку.

3. Триггер, который не позволит удалить клиента, если на него есть заказы.

```
CREATE OR REPLACE FUNCTION prevent_delete_client()
RETURNS TRIGGER AS $$
BEGIN
    IF EXISTS (
        SELECT 1 FROM "order"
        WHERE client = OLD.id_client
    ) THEN
        RAISE EXCEPTION 'Cannot delete client with existing orders';
    END IF;
    RETURN OLD;
END;
$$ LANGUAGE plpgsql;
```

```
CREATE TRIGGER prevent_delete_client
BEFORE DELETE ON "client"
FOR EACH ROW
EXECUTE FUNCTION prevent_delete_client();
```

```
HOTEL=# DELETE FROM client WHERE id_client = 18;
ERROR:  Cannot delete client with existing orders
CONTEXT:  PL/pgSQL function prevent_delete_client() line 7 at RAISE
HOTEL=#
```

Выводы:

В данной лабораторной работе при выполнении варианта 1 я овладела практическими навыками создания и использования процедур, функций и триггеров в базе данных PostgreSQL.