

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: Тюленев А.С.

Факультет: ИКТ

Группа: K3240

Преподаватель: Говорова М.М.



Санкт-Петербург 2023

Оглавление

Вариант 6. БД «Пассажир».....	3
Ход работы.....	4
Вывод.....	12

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

Оборудование: компьютерный класс.

Программное обеспечение: СУБД PostgreSQL, pgadmin 4.

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.

Вариант 6. БД «Пассажир»

Описание предметной области: Информационная система служит для продажи железнодорожных билетов. Билеты могут продаваться на текущие сутки или предварительно (не более чем за 45 суток). Цена билета при предварительной продаже снижается на 5%. Билет может быть приобретен в кассе или онлайн. Если билет приобретен в кассе, необходимо знать, в какой. Для каждой кассы известны номер и адрес. Кассы могут располагаться в различных населенных пунктах.

Поезда курсируют по расписанию, но могут назначаться дополнительные поезда на заданный период или определенные даты.

По всем промежуточным остановкам на маршруте известны название, тип населенного пункта, время прибытия, отправления, время стоянки.

Необходимо учитывать, что местом посадки и высадки пассажира могут быть промежуточные пункты по маршруту.

БД должна содержать следующий минимальный набор сведений: Номер поезда. Название поезда. Тип поезда. Пункт назначения. Пункт назначения для проданного билета. Номер вагона. Тип вагона. Количество мест в вагоне. Цена билета. Дата отправления. Дата прибытия. Дата прибытия для пункта назначения проданного билета. Время отправления. Номер вагона в поезде. Номер билета. Место. Тип места. Фамилия пассажира. Имя пассажира. Отчество пассажира. Паспортные данные.

Задание 1.1 (ЛР 1 БД). Выполните инфологическое моделирование базы данных системы. (Ограничения задать самостоятельно.)

Задание 1.2. Создайте логическую модель БД, используя ИЛМ (задание 1.1). Используйте необходимые средства поддержки целостности данных в СУБД.

Задание 2. Создать запросы:

- Свободные места на все поезда, отправляющиеся с вокзала в течение следующих суток.
- Список пассажиров, отправившихся в Москву всеми рейсами за прошедшие сутки.
- Номера поездов, на которые проданы все билеты на следующие сутки.
- Свободные места в купейные вагоны всех рейсов до Москвы на текущие сутки.
- Выручка от продажи билетов на все поезда за прошедшие сутки.

- Общее количество билетов, проданных по всем направлениям в вагоны типа “СВ”.
- Номера и названия поездов, все вагоны которых были заполнены менее чем наполовину за прошедшие сутки.

Задание 3. Создать представление:

- для пассажиров о наличии свободных мест на заданный рейс;
- количество непроданных билетов на все поезда, формирующиеся за прошедшие сутки (номер поезда, тип вагона, количество).

Задание 4. Создать хранимые процедуры:

- Для повышения цен в пригородные поезда на 20%.
- Для создания нового рейса на поезд.
- Для формирования общей выручки по продаже билетов за сутки.

Задание 5. Создать необходимые триггеры.

Ход работы:

1. Процедура для повышения цен в пригородные поезда на 20%.

```
CREATE OR REPLACE PROCEDURE raise_suburban_ticket_prices()
LANGUAGE plpgsql
AS $$
BEGIN
    UPDATE ticket
    SET ticket_price = ticket_price * 1.20
    WHERE trip_id IN (
        SELECT tp.id
        FROM trip tp
        JOIN train tr ON tp.train = tr.id
        WHERE tr.train_type = 'Пригородный'
    );
END;
$$;
```

```
CREATE OR REPLACE PROCEDURE raise_suburban_ticket_prices()
LANGUAGE plpgsql
AS $$
BEGIN
    UPDATE ticket
    SET ticket_price = ticket_price * 1.20
    WHERE trip_id IN (
        SELECT tp.id
        FROM trip tp
        JOIN train tr ON tp.train = tr.id
        WHERE tr.train_type = 'Пригородный'
    );
END;
$$;
```

Data Output Messages Notifications

CREATE PROCEDURE

Query returned successfully in 24 msec.

```

SELECT tkt.*, tr.train_name, tr.train_type, tp.departure_date
FROM ticket tkt
JOIN seats s ON tkt.seat_id = s.id
JOIN wagon w ON s.wagon = w.id
JOIN trip tp ON w.trip = tp.id
JOIN train tr ON tp.train = tr.id
WHERE tr.train_type = 'Пригородный';

```

Data Output

Messages

Notifications

id

integer

passenger

integer

seats

integer

ticket_price

integer

sale_date

date

ticket_status

character varying

tickettype

integer

seat_id

integer

passenger_id

integer

2. Процедура для создания нового рейса на поезд.

```
ALTER TABLE public.trip  
ALTER COLUMN id ADD GENERATED ALWAYS AS IDENTITY;
```

```
ALTER TABLE public.trip  
ALTER COLUMN id ADD GENERATED ALWAYS AS IDENTITY;
```

Data Output	Messages	Notifications
ALTER TABLE		
Query returned successfully in 25 msec.		

```
CREATE OR REPLACE PROCEDURE public.create_new_train_trip(  
    train_id INTEGER,  
    p_departure_date DATE,  
    p_arrival_date DATE,  
    p_status VARCHAR  
)  
LANGUAGE plpgsql  
AS $$  
BEGIN  
    INSERT INTO public.trip (train, departure_date, arrival_date, status)  
    VALUES (train_id, p_departure_date, p_arrival_date, p_status);  
  
    COMMIT;  
END;  
$$;
```

Data Output Messages Notifications

CREATE PROCEDURE

Query returned successfully in 28 msec.

Query Query History

```
1  CREATE OR REPLACE PROCEDURE public.create_new_train_trip(  
2      train_id INTEGER,  
3      p_departure_date DATE,  
4      p_arrival_date DATE,  
5      p_status VARCHAR  
6  )  
7  LANGUAGE plpgsql  
8  AS $$  
9  BEGIN  
10     INSERT INTO public.trip (train, departure_date, arrival_date,  
11     VALUES (train_id, p_departure_date, p_arrival_date, p_status)  
12  
13     COMMIT;  
14 END;  
15 $$;  
16
```

CALL

Query returned successfully in 33 msec.

Query Query History

```
1  CALL public.create_new_train_trip(1, '2024-01-01', '2024-01-02',  
2
```


3. Процедура для формирования общей выручки по продаже билетов за сутки.


```
CREATE OR REPLACE FUNCTION calculate_daily_revenue(p_date DATE)
RETURNS INTEGER AS $$
DECLARE
    v_total_revenue INTEGER;
BEGIN
    SELECT COALESCE(SUM(ticket_price), 0)
    INTO v_total_revenue
    FROM public.ticket
    WHERE sale_date = p_date;

    RETURN v_total_revenue;
END;
$$ LANGUAGE plpgsql;
```

CREATE FUNCTION

Query returned successfully in 22 msec.

Query	Query History
1	CREATE OR REPLACE FUNCTION calculate_daily_revenue(p_date DATE)
2	RETURNS INTEGER AS \$\$
3	DECLARE
4	v_total_revenue INTEGER;
5	BEGIN
6	SELECT COALESCE(SUM(ticket_price), 0)
7	INTO v_total_revenue
8	FROM public.ticket
9	WHERE sale_date = p_date;
10	
11	RETURN v_total_revenue;
12	END;
13	\$\$ LANGUAGE plpgsql;

	calculate_daily_revenue integer	
1		0

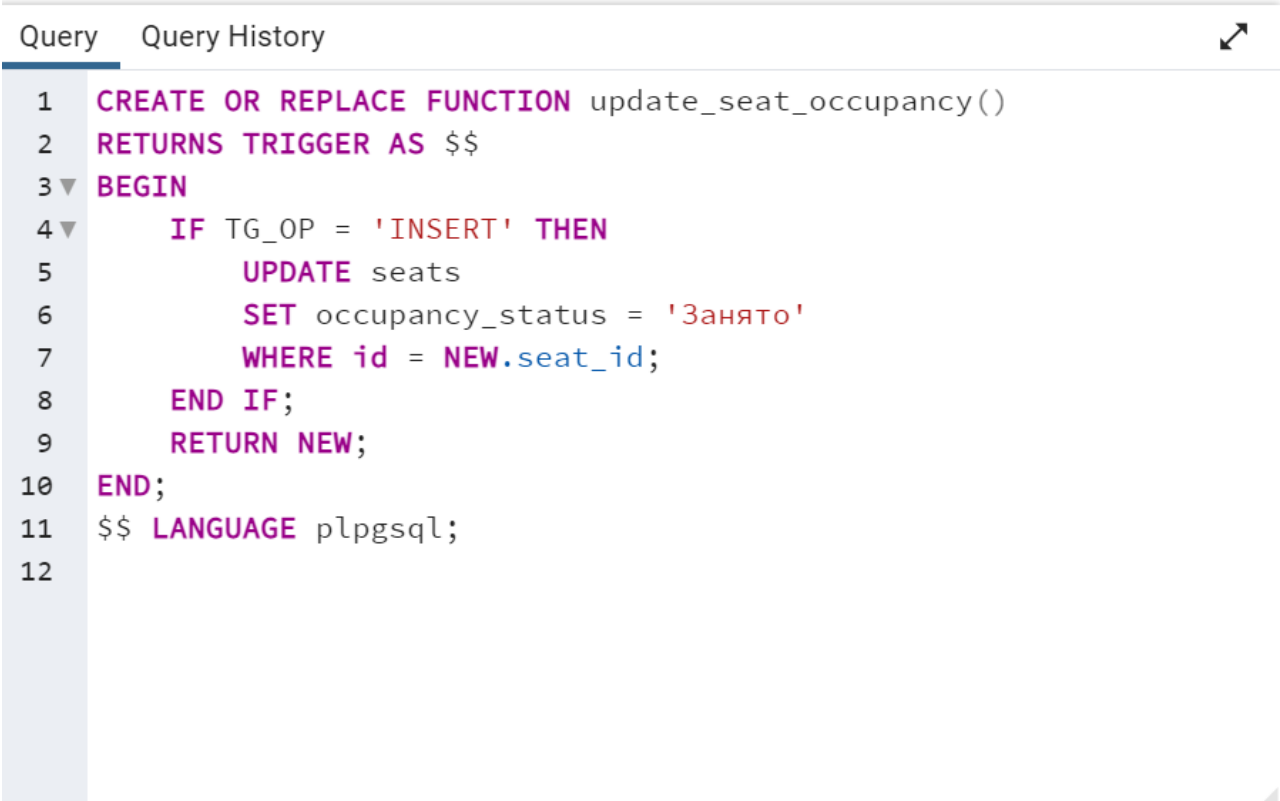
Query	Query History
1	SELECT calculate_daily_revenue('2024-01-01');
2	

Триггер

Триггер, который изменяет статус занятости места, при вставки новой записи в ticket

```
CREATE OR REPLACE FUNCTION update_seat_occupancy()  
RETURNS TRIGGER AS $$  
BEGIN  
    IF TG_OP = 'INSERT' THEN  
        UPDATE seats  
        SET occupancy_status = 'Занято'  
        WHERE id = NEW.seat_id;  
    END IF;  
    RETURN NEW;  
END;  
$$ LANGUAGE plpgsql;  
  
CREATE FUNCTION
```

Query returned successfully in 35 msec.



```
Query  Query History  ↗  
1  CREATE OR REPLACE FUNCTION update_seat_occupancy()  
2  RETURNS TRIGGER AS $$  
3  BEGIN  
4  IF TG_OP = 'INSERT' THEN  
5      UPDATE seats  
6      SET occupancy_status = 'Занято'  
7      WHERE id = NEW.seat_id;  
8  END IF;  
9  RETURN NEW;  
10 END;  
11 $$ LANGUAGE plpgsql;  
12
```

```
CREATE TRIGGER ticket_after_insert  
AFTER INSERT ON ticket  
FOR EACH ROW EXECUTE FUNCTION update_seat_occupancy();
```

CREATE TRIGGER

Query returned successfully in 36 msec.

Query Query History

```
1 CREATE TRIGGER ticket_after_insert
2 AFTER INSERT ON ticket
3 FOR EACH ROW EXECUTE FUNCTION update_seat_occupancy();
4
```

INSERT 0 1

Query returned successfully in 22 msec.

Query Query History

```
1 INSERT INTO ticket (id, passenger, seats, ticket_price, sale_date
2 VALUES (100, 1, 1, 5000, CURRENT_DATE, 'Выкуплен', 1, 10, 201, 1,
3
```


Вывод

В процессе выполнения данной лабораторной работы мы освоили навыки создания и использования процедур, функций и триггеров в PostgreSQL.