Санкт-Петербургский Политехнический Университет Петра Великого
Институт компьютерных наук и технологий
Кафедра компьютерных систем и программных технологий

ОТЧЕТ
по лабораторной работе

«Триггеры, вызовы процедур»

Базы данных

Работу выполнил студент

группа 43501/3 Дьячков В.В.

Работу принял преподаватель

_____ Мяснов А.В.

Санкт-Петербург

2018

Содержание

1. Цель работы

Познакомить студентов с возможностями реализации более сложной обработки данных на стороне сервера с помощью хранимых процедур и триггеров.

2. Программа работы

1. Создание двух триггеров: один триггер для автоматического заполнения ключевого поля, второй триггер для контроля целостности данных в подчиненной таблице при удалении/изменении записей в главной таблице.
2. Создание триггера в соответствии с индивидуальным заданием, полученным у преподавателя.
3. Создание триггера в соответствии с индивидуальным заданием, вызывающего хранимую процедуру.
4. Выкладывание скрипта с созданными сущностями в GitLab.
5. Демонстрация результатов преподавателю.

3. Хранимые процедуры

3.1. Автоматическое заполнение ключевого поля

Создадим триггер для автоматического заполнения ID в таблице `room_type` . Внутри триггера, для наглядности, будем брать следующее значение последовательности `room_type_id_seq` и прибавлять 100.

```
CREATE OR REPLACE FUNCTION room_type_primary_key()
    RETURNS TRIGGER
LANGUAGE plpgsql
AS $$
BEGIN
    SELECT nextval('room_type_id_seq') + 100 INTO new.id;
    RETURN new;
END;
$$;

CREATE TRIGGER room_type_primary_key
    BEFORE INSERT
    ON room_type
    FOR EACH ROW EXECUTE PROCEDURE room_type_primary_key();

SELECT *
FROM room_type
WHERE id = (SELECT MAX(id) FROM room_type);
-- id, type,   capacity, description
-- 9,  Studio, 4,        A room with a studio bed - a couch that can be converted into a bed

```

```sql
INSERT INTO room_type
VALUES (DEFAULT, 'Twin', 2, 'A room with two beds');

SELECT *
FROM room_type
WHERE id = (SELECT MAX(id) FROM room_type);
-- id,  type, capacity, description
-- 110, Twin, 2,        A room with two beds
```

Листинг 1: `primary_key_trigger.sql`

Видно, что триггер сработал и заполнил значение ключевого поля.

Контроль целостности данных в подчиненных таблицах обеспечивается ограничениями `constraints` внешних ключей.

```sql
UPDATE room SET id = 5009 WHERE id = 1212
-- [23503] ERROR: update or delete on table "room" violates foreign key constraint "
    reservation_room_id_fkey" on table "reservation"
-- Detail: Key (id)=(1212) is still referenced from table "reservation".

DELETE FROM room WHERE id = 1212
-- [23503] ERROR: update or delete on table "room" violates foreign key constraint "
    reservation_room_id_fkey" on table "reservation"
-- Detail: Key (id)=(1212) is still referenced from table "reservation".
```

Листинг 2: `foreign_key_trigger.sql`

3.2. Расчет стоимости бронирования

Задание: Расчет стоимости бронирования по данным стоимости номеров в разные моменты времени

С помощью функции `generate_series` сгенерируем набор дат, на которые забронирован номер (исключим последний, так как будем считать что день выезда не входит в стоимость бронирования).

Список сгенерированных дат соединим (`JOIN`) с таблицей цен по условию: ID комнат совпадают и сгенерированная дата попадает в интервал дат нужной цены. Тогда сумма цен (`price`) будет является рассчитанной ценой данного бронирования.

```sql
CREATE OR REPLACE FUNCTION calculate_price()
    RETURNS trigger
LANGUAGE plpgsql
AS $$
BEGIN
    SELECT SUM(price.price) INTO new.price
    FROM generate_series(new."from", new."to" - INTERVAL '1 day', '1 day') AS date
         JOIN price ON price.room_id = 1 AND date BETWEEN "from" AND "to";
    RETURN new;
END;
$$;

CREATE TRIGGER calculate_price
    BEFORE INSERT OR UPDATE
    ON reservation
    FOR EACH ROW EXECUTE PROCEDURE calculate_price();
```

Листинг 3: `calculate_price.sql`

Попробуем добавить номер, не указывая цену, причем для наглядности возьмем даты бронирования на границе период действия цен.

```sql
SELECT *
FROM price
WHERE room_id = 1;
-- id,    from,       to,         price,       room_id
-- 35019, 2017-01-01, 2017-11-08, "$2,951.00", 1
-- 35020, 2017-11-09, 2018-04-26, "$2,666.00", 1
-- 35021, 2018-04-27, 2019-01-26, "$6,319.00", 1

INSERT INTO reservation
VALUES (DEFAULT, 1, 1, '2018-04-20', '2018-04-30', NULL, TRUE);

SELECT *
FROM reservation
WHERE id = (SELECT MAX(id) FROM reservation);
-- id,     room_id, user_id, from,       to,         price,        is_paid
-- 100031, 1,       1,       2018-04-20, 2018-04-30, "$37,619.00", true
```

Листинг 4: `calculate_price_example.sql`

Видно, что цена 10-дневного бронирования была рассчитана верно:

$$7 \text{ дней} \cdot \$2,666 + 3 \text{ дня} \cdot \$6,319 = \$37,619,00.$$

3.3. Проверка доступности номера

Задание: При добавлении бронирования проверять доступность номера, в случае недоступности – выбрасывать исключение.

Для проверки доступности номера на весь период бронирования выполним подзапрос для поиска конфликтующих бронирований (с тем же номером и указанным условием на даты). После этого, если список конфликтующих бронирований не пуст, то бросим исключение, содержащее поясняющее сообщение и список.

```sql
CREATE OR REPLACE FUNCTION reservation_validate_availability()
    RETURNS trigger
LANGUAGE plpgsql
AS $$
DECLARE
    conflicts BIGINT [];
BEGIN
    conflicts := ARRAY(
        SELECT id
        FROM reservation
        WHERE id != new.id
            AND room_id = new.room_id
            AND "to" > new."from"
            AND "from" < new."to"
    );
    IF array_length(conflicts, 1) > 0 THEN
        RAISE EXCEPTION 'Room is not available for the specified dates. Conflicts: %',
    conflicts;
    END IF;
    RETURN new;
END;
```

```
$$;

CREATE TRIGGER reservation_validate_availability
    BEFORE INSERT OR UPDATE
    ON reservation
    FOR EACH ROW EXECUTE PROCEDURE reservation_validate_availability();

INSERT INTO reservation
VALUES (DEFAULT, 2, 1, '2017-01-01', '2017-01-27', 1234, TRUE);
```

Листинг 5: `reservation_validate_availability.sql`

Попробуем добавить бронирование, конфликтующее по датам с бронированием, созданным в предыдущем пункте. Для наглядности попробуем несколько разных комбинаций: новое начинается до предыдущего и кончается в середине предыдущего; новое начинается до предыдущего и кончается после предыдущего и т.д.

```
SELECT * FROM reservation
WHERE id = (SELECT MAX(id) FROM reservation);
-- id,     room_id, user_id, from,       to,         price,        is_paid
-- 100031, 1,       1,       2018-04-20, 2018-04-30, "$37,619.00", true

INSERT INTO reservation
VALUES (DEFAULT, 1, 1, '2018-04-01', '2018-04-25', NULL, TRUE);
-- [P0001] ERROR: Room is not available for specified date. Conflicts: {100031}
-- Where: PL/pgSQL function reservation_validate_availability() line 14 at RAISE

INSERT INTO reservation
VALUES (DEFAULT, 1, 1, '2018-04-01', '2018-05-31', NULL, TRUE);
-- [P0001] ERROR: Room is not available for specified date. Conflicts: {26126,75734,100031}
-- Where: PL/pgSQL function reservation_validate_availability() line 14 at RAISE

INSERT INTO reservation
VALUES (DEFAULT, 1, 1, '2018-04-25', '2018-05-31', NULL, TRUE);
-- [P0001] ERROR: Room is not available for specified date. Conflicts: {26126,75734,100031}
-- Where: PL/pgSQL function reservation_validate_availability() line 14 at RAISE

INSERT INTO reservation
VALUES (DEFAULT, 1, 1, '2018-04-25', '2018-04-26', NULL, TRUE);
-- [P0001] ERROR: Room is not available for specified date. Conflicts: {100031}
-- Where: PL/pgSQL function reservation_validate_availability() line 14 at RAISE
```

Листинг 6: `reservation_validate_availability_example.sql`

Видно, что добавление конфликтующих бронирований было отклонено.

4. Выводы

В процессе выполнения данной работы:

- изучены возможности языка `PL/pgSQL` ;
- создан триггер для автоматического заполнения ключевого поля;
- создан триггер для автоматического расчета стоимости бронирования;
- создан триггер для автоматической проверки доступности номера на указанные даты при добавлении нового бронирования.