

Санкт-Петербургский политехнический университет Петра Великого
Институт компьютерных наук и технологий
Кафедра компьютерных систем и программных технологий

Базы данных

Отчет по лабораторной работе №6
SQL программирование: Триггеры, вызовы процедур

Работу
выполнил:
Графов Д.И.
Группа: 33531/2
Преподаватель:
Мяснов А.В.

Санкт-Петербург
2019

Содержание

1. Цель работы	3
2. Программа работы	3
3. Ход работы	3
3.1. Триггер для автоматического заполнения ключевого поля	3
3.2. Триггер для контроля целостности данных в подчинённой таблице	4
3.3. Триггер, автоматически рассчитывающий среднюю цену товаров (напитков и еды) при добавлении новых поставок	6
3.4. Триггер, запрещающий добавлять одни и те же позиции товара с разными стоимостями в один и тот бар	7
4. Выводы	9

1. Цель работы

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

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. Ход работы

3.1. Триггер для автоматического заполнения ключевого поля

Для тестирования создадим таблицу `test_table` и последовательность `test_table_seq`, из которой будут браться значения `primary key`.

Листинг 1: Создание таблицы и последовательности для неё

```
1 create sequence if not exists test_table_seq as int start with 1;
2
3 create table if not exists test_table
4 (
5     id      int primary key not null,
6     value   int
7 );
```

Теперь создадим триггер и процедуру для него. Триггер будет срабатывать перед добавлением новых значений в таблицу. Он будет брать параметр `new` и добавлять в него следующее значение последовательности.

Листинг 2: Создание триггера и процедуры для него

```
1 create or replace function test_id_correction() returns trigger as
2 $$
3 begin
4     new.id = nextval('test_table_seq');
5     return new;
6 end;
7 $$ language plpgsql;
8
9 create trigger auto_upd
10 before insert
11 on test_table
```

```

12  for each row
13  execute procedure test_id_correction();
14
15  insert into test_table (value)
16  values  (1),
17          (2),
18          (3),
19          (4);
20
21  select *
22  from test_table;

```

Протестируем созданный триггер. Добавим последовательность значений в таблицу и убедимся, что primary key расставлены верно.

Листинг 3: Добавление новых значений в таблицу

```

1  insert into test_table (value)
2  values  (1),
3          (2),
4          (3),
5          (4);
6
7  select *
8  from test_table;

```

	id ↕	value ↕
1	1	1
2	2	2
3	3	3
4	4	4

Рисунок 3.1. Результат выполнения запроса

Как видим, первичные ключи автоматически добавились в кортежи.

3.2. Триггер для контроля целостности данных в подчинённой таблице

Создадим две таблицы: test_table_master (далее: master) и test_table_slave (далее: slave), при этом поле master_id таблицы slave ссылается на поле id из таблицы master.

Листинг 4: Создание тестовых таблиц

```

1  create table if not exists test_table_master
2  (
3    id      int primary key not null,
4    value  int
5  );
6
7  create table if not exists test_table_slave
8  (
9    id          int primary key not null,
10   master_id int references test_table_master (id),

```

```

11 | name      text
12 | );

```

Теперь нужно создать два триггера (before update(delete) и after update) и две процедуры для них.

Алгоритм для обновления: перед тем, как обновить данные в таблице master, в ней же создаётся запись с id = -1. Все записи в таблице slave, которые зависят от изменяемого master.id, меняют свои master_id на -1. Затем в таблице master происходит обновление, после чего все записи slave, у которых в поле master_id стоит -1, меняют его на обновлённое значение.

При удалении же вызывается триггер только before delete, в котором удаляются зависимые значения из таблицы slave.

Листинг 5: Код триггеров и процедур

```

1 create or replace function slave_update_before_func()
2   returns trigger as
3   $$
4   begin
5     if (tg_op = 'DELETE') then
6       delete from test_table_slave where master_id = old.id;
7       return old;
8     end if;
9     raise notice 'start_updating';
10    —create fake record in master
11    insert into test_table_master values (-1, 0);
12    —redirect slave to fake record
13    update test_table_slave set master_id = -1 where master_id = old.id;
14    raise notice 'slave_redirected';
15    return new;
16 end;
17 $$ language plpgsql;
18
19 create or replace function slave_update_after_func()
20   returns trigger as
21   $$
22   begin
23    —redirect slave to new record
24    update test_table_slave set master_id = new.id where master_id = -1;
25    raise notice 'slave_redirect_back';
26    —delete fake record
27    delete from test_table_master where id = -1;
28    return new;
29 end;
30 $$ language plpgsql;
31
32 create trigger slave_update_before
33   before update or delete
34   on test_table_master
35   for each row
36 execute procedure slave_update_before_func();
37
38 create trigger slave_update_after
39   after update
40   on test_table_master
41   for each row
42 execute procedure slave_update_after_func();

```

Протестируем триггеры. Сначала посмотрим, что находится в таблице master и slave.

	id	value
1	1	1
2	2	2
3	3	3

Рисунок 3.2. Таблица master

	id	master_id	name
1	1	1	first slave of 1
2	2	1	second slave of 2
3	3	2	first slave of 2

Рисунок 3.3. Таблица slave

Выполним следующие запросы.

Листинг 6: Обновление и удаление данных из master

```

1 update test_table_master
2 set id = 4
3 where id = 2;
4
5 delete
6 from test_table_master
7 where id = 1;

```

	id	value
1	3	3
2	4	2

Рисунок 3.4. Таблица master

	id	master_id	name
1	3	4	first slave of 2

Рисунок 3.5. Таблица slave

Как видим, данные были успешно удалены и изменены как в master, так и в slave

3.3. Триггер, автоматически рассчитывающий среднюю цену товаров (напитков и еды) при добавлении новых поставок

Листинг 7: sp1.sql

```

1 create or replace function fill_avg_price_drinks() returns trigger as
2 $$
3 begin
4     new.price_per_item = (select avg(average_price) from drinks);
5     return new;
6 end;
7 $$ language plpgsql;
8
9 create or replace function fill_avg_price_food() returns trigger as
10 $$
11 begin
12     new.price_per_item = (select avg(average_price) from food);
13     return new;
14 end;
15 $$ language plpgsql;
16
17 create trigger fill_avg_price_drinks_trig
18 before insert
19 on supplies_drinks

```

```

20     for each row
21 execute procedure fill_avg_price_drinks();
22
23 create trigger fill_avg_price_food
24     before insert
25     on supplies_food
26     for each row
27 execute procedure fill_avg_price_food();
28
29 insert into supplies_drinks (place_id, drink_id, amount, date)
30 values (1, 1, 1, now());
31
32 insert into supplies_food (place_id, food_id, amount, date)
33 values (1, 1, 1, now());
34
35 select * from supplies_drinks where date is not null order by date desc limit 1;
36 select * from supplies_food where date is not null order by date desc limit 1;

```

Создадим две процедуры, рассчитывающие и возвращающие в качестве атрибута new.price_per_item среднюю цену из таблицы drinks и таблицы food (строки 1-15). Далее создадим триггеры, использующие данные процедуры перед вставкой нового кортежа (строки 17 - 27). Таким образом, при добавлении новой записи в таблицу supplies_drinks поле average_price автоматически заполняется средним значением цены из таблицы drinks. Протестируем работу триггеров (строки 29 - 36).

supplies_drinks_id	place_id	drink_id	amount	price_per_item	date
1	100011	1	1	576.9019607843137	2019-06-28 04:52:47.902543

Рисунок 3.6. Новый кортеж таблицы supplies_drinks

supplies_food_id	place_id	food_id	amount	price_per_item	date
1	1011	1	1	4947.73	2019-06-28 04:52:47.935514

Рисунок 3.7. Новый кортеж таблицы supplies_food

3.4. Триггер, запрещающий добавлять одни и те же позиции товара с разными стоимостями в один и тот бар

Листинг 8: sp1.sql

```

1 insert into drinks(title)
2 values ('test'),
3       ('test');
4
5 create or replace function places_drinks_correction() returns trigger as
6 $$
7 declare
8     _title      varchar(50);
9     equal_titles boolean;
10    old_price    double precision;
11 begin
12     _title = (select d.title
13              from drinks d
14              where drink_id = new.drink_id

```

```

15         limit 1);
16     equal_titles = (select count(*) from drinks where title = _title) > 0;
17     if (equal_titles) then
18         old_price = (select pd.price
19                     from places_drinks pd
20                     join drinks d
21                     on pd.drink_id = d.drink_id and d.title =
↪ _title and pd.place_id = new.place_id);
22         if (new.price != old_price) then
23             raise exception 'Error! Trying to add new item with different price.
↪ ';
24         end if;
25     end if;
26     return new;
27 end;
28 $$ language plpgsql;
29
30 create trigger places_drinks_correction_tr
31     before insert
32     on places_drinks
33     for each row
34 execute procedure places_drinks_correction();
35
36 insert into places_drinks(place_id, drink_id, price, amount)
37 values (1, (select drink_id from drinks where title = 'test' limit 1), 100, 1);
38
39 insert into places_drinks(place_id, drink_id, price, amount)
40 values (1, (select drink_id from drinks where title = 'test' offset 1 limit 1),
↪ 200, 1);

```

Добавим новые позиции в таблицу drinks с одинаковыми названиями (строки 1 - 3). Создадим функцию, которая при равенстве названия и неравенстве стоимости существующей записи и новой в таблице places_drinks в бросает exception с описанием 'Error! Trying to add new item with different price.' (строки 5 - 28). Создадим триггер, вызывающий процедуру перед добавлением записи в places_drinks (строки 30 - 34). Попробуем добавить две новые позиции с одинаковым названием и разной стоимостью в бар с place_id = 1.

```

sql> insert into places_drinks(place_id, drink_id, price, amount)
      values (1, (select drink_id from drinks where title = 'test' limit 1), 100, 1)
[2019-06-28 08:20:18] 1 row affected in 13 ms
sql> insert into places_drinks(place_id, drink_id, price, amount)
      values (1, (select drink_id from drinks where title = 'test' offset 1 limit 1), 200, 1)
[2019-06-28 08:20:18] [P0001] ERROR: Error! Trying to add new item with different price.
[2019-06-28 08:20:18] Where: PL/pgSQL function places_drinks_correction() line 18 at RAISE

```

Рисунок 3.8. Попытка добавить одну и ту же позицию товара с разными стоимостями в один бар

Как мы можем видеть, мы получили ошибку. Ошибка также возникнет, если drink_id двух записей будут равны. Но если стоимость этих товаров одинакова, операция пройдёт успешно.

4. Выводы

В ходе работы были изучены триггеры в PostgreSQL.

Это очень мощный инструмент для обработки данных на стороне сервера, расширяющий возможности разработчика.

Триггер является указанием, что база данных должна автоматически выполнить заданную функцию, всякий раз когда выполнен определённый тип операции. Триггеры можно использовать с таблицами, представлениями и внешними таблицами.

Для обычных и сторонних таблиц можно определять триггеры, которые будут срабатывать до или после любой из команд INSERT, UPDATE или DELETE; либо один раз для каждой модифицируемой строки, либо один раз для оператора SQL. Для представлений триггеры могут быть определены для выполнения вместо операций INSERT, UPDATE и DELETE. Такие триггеры INSTEAD OF вызываются единожды для каждой строки, которая должна быть изменена в этом представлении. Именно функция триггера отвечает за то, чтобы произвести необходимые изменения в нижележащих базовых таблицах представления и должным образом возвращать изменённые строки, чтобы они появлялись в представлении. Триггеры для представлений тоже могут быть определены так, что они будут выполняться единожды для всего оператора SQL, до или после операций INSERT, UPDATE или DELETE. Однако такие триггеры срабатывают, только если для представления определён триггер INSTEAD OF. В противном случае все операторы, обращающиеся к представлению, должны быть переписаны в виде операторов, обращающихся к нижележащим базовым таблицам, и тогда будут срабатывать триггеры, установленные для этих таблиц.

Триггерная функция должна быть создана до триггера. Она должна быть объявлена без аргументов и возвращать тип trigger. (Триггерная функция получает данные на вход посредством специально переданной структуры TriggerData, а не в форме обычных аргументов.)

После создания триггерной функции создаётся триггер с помощью CREATE TRIGGER. Одна и та же триггерная функция может быть использована для нескольких триггеров.

В результате работы были созданы триггеры для автоматического заполнения ключевого поля в таблице и сохранения целостности данных в зависимой таблице при изменении или удалении данных в главной таблице. Также были созданы триггеры по заданию преподавателя: для подсчёта средней стоимости позиции и для запрета добавления новой позиции с существующим названием и новой стоимостью.