Санкт-Петербургский Государственный Политехнический
Университет Петра Великого
Институт Компьютерных Наук и Технологий

Кафедра Компьютерных Систем и Программных Технологий

Отчёт по лабораторной работе
Дисциплина: Базы данных
Тема: SQL-программирование: Триггеры, вызовы процедур

Выполнил студент группы 43501/3 _____ Круминьш Д.В.
(подпись)

Преподаватель _____ Мяснов А.В.
(подпись)

Санкт-Петербург
2016

Программа работы

1. Создать два триггера: один триггер для автоматического заполнения ключевого поля, второй триггер для контроля целостности данных в подчиненной таблице при удалении/изменении записей в главной таблице
2. Создать триггер в соответствии с **индивидуальным заданием**, полученным у преподавателя
3. Создать триггер в соответствии с **индивидуальным заданием**, вызывающий хранимую процедуру
4. Выложить скрипт с созданными сущностями в svn
5. Продемонстрировать результаты преподавателю

Ход работы

Триггер - это хранимая процедура, которая не вызывается непосредственно, а исполняется при наступлении определенного события (вставка, удаление, обновление строки).

1 Триггер для автоматического заполнения ключевого поля.

Был создан генератор STORAGE_GEN.

```
1 CREATE sequence STORAGE_GEN;
2 ALTER SEQUENCE STORAGE_GEN RESTART WITH 3;
```

Листинг 1: Создание генератора

Далее был создан триггер, использующий данный генератор. В случае если id был введен вручную, то автоматически изменяется значения генератора для последующего использования.

```
1  create trigger storage_id_autoinc for storage
2  active before insert position 0
3  as
4  declare variable tmp DECIMAL(18,0);
5  begin
6      if(new.storage_id is null) then
7         new.storage_id=gen_id(STORAGE_GEN,1);
8      else
9      begin
10         tmp=gen_id(STORAGE_GEN,0);
11         if(tmp<new.storage_id) then
12             tmp=gen_id(STORAGE_GEN, new.storage_id-tmp);
13     end
14 end
```

Листинг 2: Создание триггера

Далее приведен пример успешно выполнившихся комманд insert.

```
insert into storage(storage_id, address, phone)
            values(55, 'qwerty','12345');
commit;
insert into storage(address, phone)
            values('qwerty','12345');
commit;
insert into storage(storage_id, address, phone)
            values(82, 'qwerty','12345');
commit;
insert into storage(address, phone)
            values('qwerty','12345');
commit;
```

Листинг 3: Пример успешных команд insert

STORAGE_ID	ADDRESS	PHONE
1	ул. Никакая, д. 123	8(905)123-45-67
2	ул. Пушкина, дом 44	8(909)331-89-49
3	address3	12345
55	qwerty	12345
56	qwerty	12345
82	qwerty	12345
83	qwerty	12345

Таблица 1: Содержимое таблицы STORAGE

2 Триггер для контроля целостности данных в подчиненной таблице при удалении/изменении записей в главной таблице.

Был создан триггер для контроля целостности данных в подчиненных таблицах при удалении/изменении записей в главной таблице my_user.

```
create exception ex_modify_user 'This user in other tables!';
create trigger user_delete_update for my_user
before delete or update
as
begin
   if(old.user_id in
   (select user_id from sells
   union
   select user_id from review))
   then exception ex_modify_user;
end
```

Листинг 4: Создание триггера при удалении/изменении пользователя

При попытке удалить строку в my_user, ключ которой содержится в подчиненных таблицах, произойдет ошибка и выведено исключение.

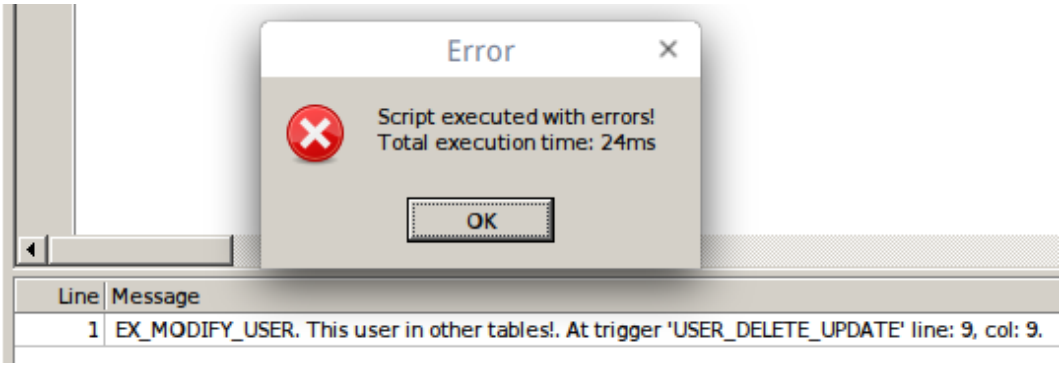

Рис. 1: Результат попытки удаления

3 Реализовать проверку при создании/изменении записей в типах товаров: необходимо проверять, что тип не является родителем самого себя.

Была создана процедура, которая для входного значения type_id выводит type_id всех родительских элементов. Работа реализована с помощью рекурсии, добавлено ограничение в 1024, так как это максимальная длина рекурсии.

```
1  create procedure proc3(
2      input_type_id integer)
3  returns(output_p_id integer)
4  as
5  begin
6  for
7  with recursive RR as(
8      select type.* from TYPE Where   TYPE_ID=:input_type_id
9      union all
10     select type.* FROM TYPE, RR where TYPE.TYPE_ID=RR.P_ID
11 )
12 select first 1024  RR.P_ID from RR into output_p_id
13 do
14 begin
15 suspend;
16 end
17 end
```

Листинг 5: Процедура по поиску родителей

Далее был создан триггер, который в процесс своей работы вызывает процедуру по поиску родителей.

```
1  create exception ex_modify_type 'Type can not be father of himself.';
2  create trigger type_id_checker for TYPE
3  active before insert or update
4  as
5  declare variable temp integer;
6  begin
7  for
8  select * from proc3(new.p_id)
9  into :temp
10 do
```

```
11 begin
12 if(:temp=new.type_id)
13 then exception ex_modify_type;
14 end
15 if(new.type_id=new.p_id)
16 then exception ex_modify_type;
17 end
```

Листинг 6: Триггер для проверки родителей

Для проверки работоспособности, в таблицу TYPE были добавлены следующие строчки.

```
1 insert into type values(20, null,'qwe',1);
2 commit;
3 insert into type values(21, 20,'qwe',2);
4 commit;
```

Листинг 7: Добавление данных для тестирования триггера

Далее были произведены попытки изменить введенные данные так, чтобы тип являлся родителем самого себя.

```
1 update type set p_id=20 where type_id=20;
```

Листинг 8: Изменение данных

```
1 update type set p_id=21 where type_id=20;
```

Листинг 9: Изменение данных

В обоих случаях сработал триггер и было выведено исключение.

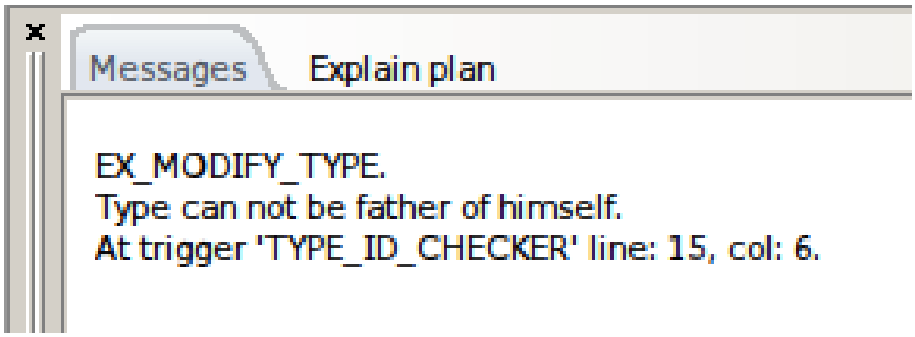

Рис. 2: Результат попытки изменения данных

4 Сделать вызов процедуры индексации цен при создании новой поставки. Индексировать только цену на товары, участвующие в поставке.

Была написана процедура, которая индексирует цену заданного товара на заданный процент.

```
1 CREATE procedure PROC4(
2 prod_id integer,
3 inflation boolean,
4 percents1 integer
5 )
6 as
```

```
declare variable currentProduct integer;
declare variable currentCount integer;
declare variable currentPrice float;
begin
if (:inflation) then
begin
update product set price=price+(price*:percents1/100) where product_id
    ↪ =:prod_id;
end
else
begin
update product set price=price-(price*:percents1/100) where product_id
    ↪ =:prod_id;
end
end
```

Листинг 10: Поцедуры инксации цены товара

Дополнительно были созданы 2 генератора, которые в будущем используются как переменные. INFLATION_TYPE может иметь следующие значения:

- 0 - когда экономика стабильно, и цены не нужно изменять
- 1 - когда происходит инфляция и необходимо повысить цены
- 2 - когда происходит деинфляция и необходимо понизить цены

INFLATION_PERCENT в свою очерь определяет процент на сколько необходимо повысить или понизить цену.

```
CREATE sequence INFLATION_TYPE;
ALTER SEQUENCE INFLATION_TYPE RESTART WITH 0;
CREATE sequence INFLATION_PERCENT;
ALTER SEQUENCE INFLATION_PERCENT RESTART WITH 1;
```

Листинг 11: Создание генераторов(переменных)

Созданный триггер вызывает процедуры индексации, когда значение INFLATION_TYPE равно 1 или 2.

```
create trigger supply_inflation_checker for supply
active before insert
as
declare variable infl boolean;
declare variable percent integer;
begin
if(gen_id(inflation_type,0)=1) then
execute procedure proc4(new.product_id, true,gen_id(inflation_percent
    ↪ ,0));
else if(gen_id(inflation_type,0)=2) then
execute procedure proc4(new.product_id, false,gen_id(inflation_percent
    ↪ ,0));
end
```

Листинг 12: Триггер при новой поставке

Вывод

В результате работы было проведено знакомство с триггерами. Триггер можно считать автоматической процедурой, срабатывающей на стороне сервера в результате некоторого события (insert, update, delete). Триггер может сработать до наступления (before) события, или же после (after).

Главным преимуществом триггеров является контроль целостности базы данных любой сложности. Так-же упрощается приложение, так как часть логики уже выполняется на сервере.

К недостаткам можно отнести следующее: большое количество триггеров сильно уменьшит производительность системы, несколько неправильно реализованных триггеров могут привести к рекурсивной модификации таблиц и большим сложностям при отладке.

Таким образом, триггеры - весьма полезный инструмент при разработки базы данных, требующий аккуратной реализации.