Санкт-Петербургский Политехнический университет Петра Великого

Институт компьютерных наук и технологий

Кафедра компьютерных систем и программных технологий

**Отчёт о лабораторной работе №6**

**Дисциплина**: Базы данных

**Тема**: Триггеры

Выполнил студент гр. 43501/1 Веселов И.Н.

(подпись)

Руководитель Мяснов А.В.

(подпись)

“ ” 2015 г.

Санкт-Петербург

2015

1. **Цель**

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

1. **Программа работы**
2. 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
3. Создать триггер в соответствии с **индивидуальным заданием**, полученным у преподавателя
4. Создать триггер в соответствии с **индивидуальным заданием**, вызывающий хранимую процедуру
5. Выложить скрипт с созданными сущностями в svn
6. Продемонстрировать результаты преподавателю
7. **Выполнение работы**

Вначале была создана тестовая таблица *AUTOTRIG* с полями *INCR* и *ZNACH* типа integer. Далее был написан скрипт для создания триггера для автоматического заполнения поля *INCR* в таблице *AUTOTRIG.*

set term ^;

CREATE OR ALTER trigger AUTOINCR for AUTOTRIG active before insert AS

begin

select max(AUTOTRIG.INCR) from AUTOTRIG into NEW.INCR;

NEW.INCR = NEW.INCR + 1;

end^

set term ;^

Работа триггера:

SQL> select \* from AUTOTRIG;

INCR ZNACH

============ ============

1 20

SQL> insert into AUTOTRIG values (100, 50);

SQL> select \* from AUTOTRIG;

INCR ZNACH

============ ============

1 20

2 50

SQL> insert into AUTOTRIG values (1, 45);

SQL> select \* from AUTOTRIG;

INCR ZNACH

============ ============

1 20

2 50

3 45

По результатам работы триггера видно, что при добавлении значения в таблицу сначала проверяется последнее (максимальное) значение поля *INCR*, оно инкрементируется, а затем подставляется в добавляемую величину.

Затем был создан триггер для контроля целостности данных в подчиненной таблице при удалении/изменении записей в главной таблице *MODELS;*

set term ^;

create or alter exception DEL\_UPD\_FAILED ‘FAILED TO CREATE OR UPDATE THE INFORMATION’^

CREATE OR ALTER trigger CONTRZEL for MODELS

active before delete or update

AS

begin

if (OLD.ID in (select Models\_Code from CARS)) then

exception DEL\_UPD\_FAILED;

end^

set term ;^

**Индивидуальное задание:**

Реализовать триггеры:

1. Ввести проверку при добавлении новой опции в заказ: если она уже есть - не добавлять.
2. В случае, если общая сумма, принесенная клиентом в салон за некоторый промежуток времени более некоторой величины - следующее посещение сервиса должно быть бесплатным.

**Первый триггер:**

set term ^;

create or alter exception OPTALREADY ’OPTION IS ALREADY EXISTS’^

CREATE OR ALTER trigger OPTIONEXISTS for ORD\_OPT

active before insert or update

AS

begin

IF(NEW.OPTION\_CODE in (select OPTION\_CODE from ORD\_OPT

where ORDER\_CODE = NEW.ORDER\_CODE))

then

exception OPTALREADY;

end^

set term ;^

Работа скрипта:

SQL> select \* from ord\_opt where order\_code=54;

ID ORDER\_CODE OPTION\_CODE PRICE\_ORD

============ ============ ============ ============

9 54 1 150

SQL> insert into ORD\_OPT values (10, 54, 1, 345);

Statement failed, SQLSTATE = 42000

exception 2

-OPTALREADY

-OPTION IS ALREADY EXISTS

-At trigger 'OPTIONEXISTS' line: 7, col: 1

При попытке добавления в таблицу *ORD\_OPT* значения опции по заказу, по которому данная опция уже была добавлена ранее, мы видим срабатывание триггера.

**Второй триггер:**

set term ^;

create or alter trigger free\_serv for SERVICE after insert

as

declare variable porog int;

declare variable summa int;

begin

porog = 10000;

summa = 0;

select sum(price\_serv) from service

where service.car\_code = new.car\_code and

date\_start between '2015-09-01' and new.date\_start

group by car\_code into :summa;

if (summa > porog) then

execute procedure add\_free\_serv(new.serv\_code);

end^

set term ;^

Данный тригер вызывает следующую процедуру:

set term ^;

create or alter procedure add\_free\_serv(srv\_code int)

as

begin

UPDATE service

set price\_serv = 0

where serv\_code = :srv\_code;

end^

set term ;^

Работа скрипта:

select \* from service;

SERV\_CODE CAR\_CODE WORK\_TYPE DATE\_START DAY\_WORK PRICE\_SERV

============ ============ ============ =========== ============ ============

1 1003 9 2015-10-13 1 100

2 1005 6 2015-08-02 3 400

3 1001 5 2015-03-27 10 850

4 1002 3 2015-09-10 2 300

5 1002 4 2015-09-15 3 500

6 1002 6 2015-09-25 7 3000

7 1002 7 2015-09-27 8 4900

8 1002 8 2015-09-30 10 5000

SQL> insert into service values (9, 1002, 6, '2015-10-01', 7, 3000);

SQL> select \* from service;

SERV\_CODE CAR\_CODE WORK\_TYPE DATE\_START DAY\_WORK PRICE\_SERV

============ ============ ============ =========== ============ ============

1 1003 9 2015-10-13 1 100

2 1005 6 2015-08-02 3 400

3 1001 5 2015-03-27 10 850

4 1002 3 2015-09-10 2 300

5 1002 4 2015-09-15 3 500

6 1002 6 2015-09-25 7 3000

7 1002 7 2015-09-27 8 4900

8 1002 8 2015-09-30 10 5000

9 1002 6 2015-10-01 7 0

1. **Вывод**

В данной работе мы познакомились с реализацией триггеров. Было создано несколько стандартных триггеров, а так же реализованы триггеры в соответствие с индивидуальным заданием.

С помощью триггеров можно накладывать ограничения на вносимые данные согласно требованиям предметной области БД.

Триггеры полезно использовать для проверки корректности вносимых в БД данных и их целостности.

Также триггеры удобно использовать для оповещения об изменении данных в таблицах.