In this lab you will have hands-on-experience with Oracle triggers. You should note that Oracle follows SQL standards for triggers. Triggers provided by the Microsoft SQL server, on the other hand, are more like statement-level-after-triggers and do not follow SQL standards. In this lab we will use only Oracle triggers.

A database trigger is defined on a single table and for INSERT, DELETE and UPDATE SQL statements performed on the table. A trigger is invoked when an SQL statement get executed on the table on which the trigger is defined.

ASSIGNMENT:

(30 MINUTES)

In the following you will create several Oracle triggers defined on the tables: DollParts and Dolls.

Create the following tables:

```sql
drop table DollParts;
drop table Dolls;
create table DollParts (  
    name   varchar2(10),
    cnt    number,
    constraint PKdollParts primary key(name),
    constraint ck1 check (cnt >= 0)
);
```

```sql
insert into DollParts(name, cnt) values('HEAD', 7);
insert into DollParts(name, cnt) values('BODY', 18);
insert into DollParts(name, cnt) values('ARM', 6);
insert into DollParts(name, cnt) values('LEG', 12);
commit;
create table Dolls (  
    name   varchar2(20),
```
cnt number,
    constraint PKdolls primary key(name)
);

insert into Dolls(name, cnt) values('Barbie', 0);
insert into Dolls(name, cnt) values('Ken', 0);
commit;

Also add the following tables to your database:
create table Message1(name char(50));
create table Message2(Count number(3));

1. Create and invoke the following trigger on table DollParts for INSERT.

    CREATE or REPLACE TRIGGER trigg1
    After Insert ON DollParts
    Begin
    Insert into Message1 values('new part inserted into the database');
    end;
    /
    select * from Message1;

    INSERT into DollParts values('HAT', 17);
    commit;
    select * from Message1;

    What do you see? What does the trigger trigg1 accomplish?

2. Create and invoke the following trigger on table DollParts for the event DELETE.

    CREATE or REPLACE TRIGGER trigg2
    After Delete ON DollParts
    begin
    Insert into Message2
(select SUM(cnt)
   from DollParts);
end;
/
select * from Message2;
Delete from DollParts where name='HAT';
commit;
select * from Message2;

What does the trigger trigg2 accomplish?

3. Create and invoke statement level triggers that will insert into a table Message3 the word 'insert' and the total number of parts inserted when rows are inserted into the table DollParts, and the word 'delete' and the total number of parts deleted when rows are deleted from the table DollParts (you may need one before and one after trigger, and a temporary table).

(30 MINUTES)

4. Above are statement level triggers (statement level is the default case).

Now you work on Oracle row-level triggers.

Following is a row level trigger:

select * from Message2;

CREATE or REPLACE TRIGGER trigg3
After update of cnt ON DollParts
For each row
when (new.cnt-old.cnt>10)
Begin
Insert into Message1 values('Dollparts inventory increased by more than 10');
end;
/
select * from Message1;
update DollParts set cnt=cnt+20 where name='LEG';
commit;
select * from Message1;
Run the above update statement. Indicate the number of times the trigger is invoked as a result of running the update statement. What does the trigger trig3 accomplish?

5. References for this problem:

http://download-west.oracle.com/docs/cd/B10501_01/appdev.920/a96590/adg10pck.htm#23517
http://download-west.oracle.com/docs/cd/B10501_01/appdev.920/a96624/07_errs.htm#943

Consider the following exception handler inside a trigger. This trigger raises exception using PL/SQL. The error handler raise_application_error will roll back the original insert or update if exception occurs.

CREATE or Replace TRIGGER trigg4
AFTER INSERT or UPDATE on DollParts
for each row
When (new.cnt > 100)
declare
cnt_too_big exception;
Begin
raise cnt_too_big;
exception
when cnt_too_big then
raise_application_error(-20000, 'Parts cnt is too large');
end;

INSERT into DollParts values('PANT', 115);

In this particular example, raise_application_error(-20000, 'Parts count is too large'); could have been used without defining the exception variable cnt_too_big.
We are defining the exception variable cnt_too_big here to show the general PL/SQL structure which is as follows:

Declare
Stmt;
Begin
    Stmt;
exception
    Stmt;
end;

Parameter -20000 is a user defined error code.

(a) What does the above trigger accomplish?
(b) Check DollParts table to verify the effect of the exception handler.

6. Define a trigger, OverSupply, which will print a message on the terminal if the total number of dolls exceeds 10. You also have to undo the operation which caused the oversupply. Note this trigger is affected by the events INSERT and UPDATE.

Execute the following event and write your results here:
INSERT into Dolls Values('newDoll', 21);
UPDATE Dolls set cnt=2*cnt;
commit;

(25 MINUTES)

7. Recursive trigger: An application updates table T1, which invokes trigger TR1 updating table T1 itself. This is a direct recursion. Recursion can be indirect as well where an application updates table T1, which invokes trigger TR1 updating table T2. Trigger TR2 defined on table T2 then updates table T1.

(a) Define a recursive trigger, oversupply, for the following:
If the total number of parts in DollParts exceeds 20 then reduce each part count by 10%
(b) Execute the following event and indicate how many times the trigger will be invoked.
UPDATE DollParts set cnt=cnt+50 where name='ARM';
commit;
(c) Give an example of a trigger where the recursion can go into an infinite loop.

8. Nested triggers:
If a trigger changes a table on which there is another trigger, the second trigger is then invoked and can then call a third trigger, and so on. There can be a nesting of up to 32 levels in Oracle. If any trigger in the chain sets off an infinite loop, the nesting level is exceeded and the trigger is cancelled.

Now you do the following:

When the number of "barbie" dolls goes below 10, add 10 more Barbie dolls to the stock. Whenever, more dolls are added to the Doll table, corresponding number of parts are subtracted from the DollParts table. When the part cnt for a part in the DollParts table goes below 10, increase the inventory of that part by 10.

9. Create a table DollPartsStat to keep statistics on the total number of tuples inserted for each insert, the total number of tuples deleted for each delete, and the total number of tuples updated for each update in the table DollParts. Thus, the DollPartsStat table will have the following type of tuples: <insert, number of tuples>, <delete, number of tuples>, <update, number of tuples>. Implement the above by defining a trigger.

(20 MINUTES)

10. Materialized Views:

Consider the following regular view:

Table of names of all those parts whose \( cnt > 10 \).

Define view OverSupply(name)

\[
\text{SELECT name} \\
\text{FROM DollParts} \\
\text{WHERE cnt>10;}
\]

Following creates a materialized view using Refresh approach (see lecture note, page 31-33). Thus, the materialized view is recreated (refreshed) from the base table for each insert, delete and update. Following trigger achieves this.

Create or replace table OverSupply(name varchar2(10));

CREATE TRIGGER TriggOverSupply

6
AFTER INSERT OR DELETE OR UPDATE of cnt on
DollParts
Begin
DELETE * FROM OverSupply;
INSERT INTO OverSupply(name)
SELECT name
FROM DollParts
WHERE cnt>10;
end;

(a) Run an sql insert and an sql delete on the base table DollParts
and check the effect of this insert and delete on the materialized
view.

(b) The refresh approach used above is not very efficient because
it is not incremental. Reimplement the materialized view using
incremental approach (lecture note on triggers: page 31-33).