TRIGGERS

TRIGGER IS A STORED PL/SQL BLOCK ASSOCIATED WITH A TABLE OR A VIEW OR A DATABASE THAT GETS AUTOMATICALLY EXECUTED WHEN THE SPECIFIED EVENTS ARE OCCUR.

There are 3 types of triggers:

1. Database triggers
2. Instead of triggers
3. Event Triggers

Database Triggers:

Database triggers are associated to a table.

```
create or replace trigger trigger_name
    before | after  insert [or delete [or update [of column(s) ] ] ]
    on  table_name
    [REFERENCING OLD AS o | NEW AS n]
    [for each row]
    [when  (cond) ]
   [ declare
      ---------;]
    begin
   --------;
   exception
    --------;
     end;
```

Instead of Triggers:

```
create or replace trigger trigger_name
 instead of  insert [or delete [or update]] on view_name
 for each row
 [ declare
    ---------;]
  begin
   --------;
  exception
  --------;
  end;
```

Event Triggers

```
create or replace trigger trigger_name
before | after triggering_event on database
begin
   --------;
  ---------;
  exception
  --------;
 end;
```

Database triggers

1. Statement-Level Triggers

   - Fired only once either before triggering-event or after triggering-event.

2. Row-Level Triggers

   - Fires once for each for each row that is affected by the triggering event.

Note: before | after insert [or delete [or update [of column(s)]]] is called triggering statement while the DML statement that causes the trigger to fire is called triggering event.

Note: If "for each row" clause is included while creating the trigger, then it is a row-level trigger else it is a statement-level trigger.

Order Of Firing

The foll. shows the order in which the triggers & events are executed.

1. User executes a DML statement(triggering event).
2. Oracle fires any BEFORE statement-level triggers
3. Oracle fires any BEFORE row-level triggers
4. Oracle executes DML on that row
5. Oracle fires any AFTER row-level triggers
6. If more than 1 row is modified,steps 3 thr' 5 are repeated until all rows have been processed.
7. Oracle fires any AFTER statement-level triggers.

Ex.1 statement-level trigger

```
create trigger check_emp_trig
before insert or update or delete on emp
begin
 if  to_char(sysdate,'HH24') < 10 or to_char(sysdate,'HH24')  > 18   then
   raise_application_error(-20000,'Cannot update other than office hours');
 end if;
 if  to_char(sysdate,'DY')  = 'SAT' or to_char(sysdate,'DY') = 'SUN'  then
   raise_application_error(-20000,'Cannot update on week-ends');
 end if;
end;
```

Note: Extend the above trigger by calling a function"check_h" that returns TRUE if the current date is an holiday else returns FALSE. The function has to verify the holiday dates from a table "holidays_tab".

```
create or replace function check_h(p_date date)
return boolean

as

v_date date;

flag boolean := FALSE;

begin

for hrec in ( select * from emp_hollidays)

loop

   if trunc(hrec.holliday) = trunc(p_date) then

       flag := TRUE;

   end if;

 end loop;

 return flag;

end;
```

```
create or replace trigger check_emp_trig
  after insert or update or delete on emp
    begin
    if  to_char(sysdate,'HH24') < 10 or to_char(sysdate,'HH24')  > 18   then
      raise_application_error(-20000,'Cannot update other than office hours');
     end if;
     if  to_char(sysdate,'DY')  = 'SAT' or to_char(sysdate,'DY') = 'SUN'  then
      raise_application_error(-20300,'Cannot update on week-ends');
     end if;
     if  check_h(sysdate) then
      raise_application_error(-20400,'Cannot update on hollidays');
     end if;
   end;
/
```

Ex. 2 statement-level trigger

Create a trigger that restricts no of rows in table to 100.

```
>create trigger restrict_newrec_trig
before insert on emp
declare
   n   integer;
begin
 select count(*)  into n from emp;
 if n >=100  then
    raise_application_error(-20000,'Table cannot hold more than 100
    records');
 end if;
end;
```

Row Level Triggers

Row-level triggers fire once for each row updated,deleted or inserted.

Correlation names:

Since row level triggers fire for each row being updated,we can access the old & new values contained in these rows by using correlation names,

:old and :new.

Usage: :old.column_name and :new.column_name

DML	:old	:new
INSERT	null	yes
UPDATE	yes	yes
DELETE	yes	null

NOTE: Correlation names are applicable only to row-level triggers.

Ex.1 Row Level Triggers

Create a trigger that ensures that job is stored in the table in upper-case letters

```
create or replace  trigger job_uppercase_trig
 before insert or update of job on emp
for each row
begin
   :new.job := upper(:new.job);
 end;
```

When clause can be used only on row level triggers. It prevents unnecessary firing of trigger.

Ex.2 Create a trigger that ensures that total pay of an employee doesn't exceed Rs. 50000/- except for president

```
>create or replace  trigger check_pay_trig
before insert or update of sal,comm on emp
for each row
WHEN ( new.job <> 'president')
begin
    if (:new.sal + :new.comm) > 50000  then
     raise_application_error(-20100, 'Total exceeding Rs.50000');
    end if;
end;
```

NOTE: When referencing :new and :old in **when clause**, do not prefix with :

Restriction on triggers

1. DDL & DCL statements are not permitted inside trigger body.
2. Commit,rollback and savepoint are not permitted.
2. The table which is currently being modified by the SQL statement is called **mutating table.** Within the body of a row level trigger,we cannot issue any SQL statement including select statement on the table on which the trigger is fired.
3. In addition tables with foreign keys pointing to the mutating table are also in the state of mutating & therefore cannot be modified by the trigger.

Differentiating Between Database Triggers and Stored Procedures

Triggers	Procedures
Defined with CREATE TRIGGER	Defined with CREATE PROCEDURE
Data dictionary contains source code in USER_TRIGGERS	Data dictionary contains source code in USER_SOURCE
Implicitly invoked	Explicitly invoked
COMMIT, SAVEPOINT, and ROLLBACK are not allowed	COMMIT, SAVEPOINT, and ROLLBACK are allowed

TRIGGER PREDICATES

Trigger predicates are used to selectively execute part of a trigger body rather than the entire trigger body.

There are 4 trigger predicates:

1.INSERTING:Returns TRUE if the triggering event is INSERT.

2.DELETING: Returns TRUE if the triggering event is DELETE.

3.UPDATING: Returns TRUE if the triggering event is UPDATE.

4.UPDATING(column_name): Returns TRUE if the triggering event is UPDATE on specified column.

Ex. III. Create a database trigger to record changes made to the EMP table.

I. Create a table:

```
>create table emp_audit
    (empno number(10),
     prev_job varchar2(30),
     pres_job varchar2(30),
    prev_sal number(10,2),
    pres_sal  number(10,2),
    change_typ varchar2(30),
    changed_by varchar2(30),
    changed_on date,
    deptno number(3)
    );
```

II. Create Database Trigger

```
create or replace trigger emp_audit_trig
after insert or update of job,sal or delete on emp
for each row
declare
user_name  varchar2(20) ;
begin
select user into user_name from dual;
if inserting then
insert into emp_audit
values(:new.empno,NULL, :new.job,NULL, :new.sal,
'INSERT' ,user_name, sysdate , :new.deptno);
```

```
elsif  deleting then
insert into   emp_audit
values(:old.empno,:old.job, NULL, :old.sal,NULL,'DELETE',
user_name, sysdate , :old.deptno);
elsif updating('sal') then
insert into   emp_audit    values(:old.empno,:old.job, NULL,
:old.sal,:new.sal,'UPDATE SALARY',user_name, sysdate,
:old.deptno);
else
insert into emp_audit
values(:old.empno,:old.job,:new.job, :old.sal, :new.sal,
'UPDATE',  user_name,sysdate,:old.deptno);
end if;
end;
```

Example

```
create table items (
        icode number constraint items_pk primary key,
        idesc varchar2(30),
        rate number(8,2),
        qty number,
        reorder number(4)
  );
```

```
create table Reorder_Status (
        sno number constraint reorder_pk primary key,
        icode number references items(icode),
        curr_stock number(6),
        short_fall number(5),
        status_dt date
    );
```

```
create sequence reorder_status_seq ;
```

```
create table stock_issue (
        stock_issue_no number(10) constraint issue_pk primary key,
        icode number references items(icode),
        stock_issue_date date,
        issue_qty number,
        issue_to varchar2(30)
    );
```

```
create sequence stock_issue_seq;
```

```
create table stock_receive (
        stock_receive_no number(10) constraint receive_pk primary key,
        icode number references items(icode),
        stock_receive_date date,
        receive_qty number,
        receive_by varchar2(30)
    );
```

```
create sequence stock_receive_seq;
```

```
insert into items values(&icode,&idesc,&rate,&qty,&reorder);
```

Trigger on Stock issue table

```
create or replace trigger stock_issue_trg
after insert on stock_issue
for each row
declare
  quantity number;
begin
select qty into quantity from items where icode = :new.icode;
 if :new.issue_qty > quantity then
    raise_application_error(-20200,'Insufficiant quantity');
          end if;
 update items set qty = qty - :new.issue_qty
 where icode = :new.icode;
end;
```

Trigger on stock receive table

```
create or replace trigger stock_receive_trg
after insert on stock_receive
for each row
begin

 update items set qty = qty + :new.receive_qty
  where icode = :new.icode;

end;
```

Trigger on items table
Cascading triggers one trigger will fire another trigger

```
create or replace trigger stock_status_trg
after update of qty on items
for each row
when (new.qty  <  old.reorder )
begin
 insert into Reorder_status
 values (reorder_status_seq.nextval, :old.icode,
 :new.qty, :old.reorder - :new.qty, sysdate);
end;
```

Enabling/Disabling triggers

1. One trigger at a time:

> ***alter trigger trigger_name enable|disable;***

2. All triggers associated to the table

> ***alter table table_name enable|disable all triggers;***

3. Dropping a trigger

> ***drop trigger trigger_name;***

Note : Dropping a table will implicitly drop all the triggers.

Instead of triggers

There are restrictions to update the base tables thr' views .

Instead of triggers enables us to update the base tables via views .

NOTE: Instead of triggers can be defined only on views and always fire for each row.

To understand instead of triggers , you need to know about updateable views.

```
SQL>  create or replace view  vempdept
      as
      select empno, ename , job, sal, comm, d.deptno as deptno , dname ,    loc
      from emp e , dept d
      where e.deptno=d.deptno;

SQL>   update vempdept set sal = sal +1000 where empno = 7369;

       1 row updated.

SQL>  update vempdept set  dname = 'HRD' where deptno= 30;

       update vempdept set dname = 'HRD' where deptno= 30
              *

       ERROR at line 1:

ORA-01779: cannot modify a column which maps to a
                  non key-preserved table
```

Create Instead of trigger

Ex. 1

```
create or replace  trigger empdept_it
  instead of   update  on vempdept
  for each row
  begin
  update dept set dname = :new.dname ,loc = :new.loc
  where deptno = :new.deptno;
  exception
  when others  then
  raise_application_error(-20000,'Error');
  end;

  SQL> update vempdept set dname = 'EDP',loc = 'Hyderabad'
  where deptno = 20;

  1 row updated.
```

```
SQL>create  view vemp(id,name,designation,apay,dno)
        as
        select empno , ename , job ,  12 * sal, deptno  from emp ;
SQL> desc vemp
        Name
        -----------------------------------------------------------

        ID
        NAME
        DESIGNATION
        APAY
        DNO
```

```
SQL> update vemp set designation = 'programmer'
        where id = 7369;

1 row updated.


SQL> update vemp set apay = 350000
        where id = 7369;

        update vemp set sal = sal + 1600
        where id = 7369
              *
        ERROR at line 1:
```

```
create or replace trigger vemp_t
 instead of insert or update of apay on vemp
 for each row
 begin
   if inserting then
   insert into emp(empno,ename,sal,deptno)
   values(:new.id,:new.name,:new.apay/12,:new.dno);
   elsif updating then
   update emp set
            sal = round(:new.apay/12)
            where empno = :new.empno;
  end if;
end;
```

```
SQL> update vemp set apay = 350000
        where id = 7369;

1 row updated.

SQL> select sal  from emp where empno = 7369;
        SAL
        ---------
        350000
```

Event Triggers

Event Triggers are fired whenever particular event occurs on the schema. The Events are ALTER TABLE, DROP TABLE…etc.

```
create table schema_audit  (
        action varchar2(40),
        object_name varchar2(30),
        ch_date date
   );
```

```
create or replace trigger schema_trg1
before alter on scott.schema
begin

        insert into schema_audit
            values (‘alter the  object ‘ , ora_dict_obj_name, sysdate);

end;
```

Note:

ORA_DICT_OBJ_NAME is a Oracle predefined function it returns the changed Object name(Ex. Table name.)

Assignment:

Create the foll. Tables:

Table_name: Item

Column_name	datatype	constraint
Item_id	number(5)	Primary Key
Item_name	varchar2(30)	
Item_desc	clob	

Table_name: Customer

Column_name	datatype	constraint
Cust_id	number(5)	Primary Key
Cust_name	varchar2(30)	
Cust_addr	varchar2(50)	
Norders	number(4)	

Table_name: Orders

Column_name	datatype	constraint
Ord_id	number(5)	Primary Key
Order_dt	date	
Cust_id	number(5)	referencing to primary key of customer table
Item_id	number(5)	referencing to primary key of item table

II. Insert 5 rows into item table

III. Insert 5 rows customer table (leave norders column blank).

IV. A row is inserted into orders table when a customer orders for an item and a row is deleted from the orders table if the customer cancels the order.

Create a database trigger that updates the norders column of customer table as given below:

When a row is inserted into orders table then the norders column of customer table of corresponding customer has to be increased by 1.

Similarly , when a row is deleted from orders table then the norders column of customer table of the corresponding customer has to reduced by 1.

Thank You!