**********Experiment no:-08**********

Author:Saurabh Khandagale
Roll No:46
Date : 22-October-2020

EXPERIMENT:-08

AIM:-

To write and execute SQL programs that allows enforcement of business
rules with database triggers.

Problem Statement:

Using the relation schemata established in Experiments - 02, 03, and 05, create and execute SQL programs that allow enforcement of business rules with database triggers.

==========Query-01==========

Write SQL code to compile and execute a trigger - UPDATE_CUST_BALANCE_TRG that will update the BALANCE in the CUSTOMER table when a new LINE record is entered. (Assume that the sale is a credit sale.) The BALANCE in CUSTOMER is 0 when customer does not have any invoice to his credit. Test the trigger, using the following new LINE record: 1006, 5, 'PP101', 10, 5.87.

==========

```
CREATE OR REPLACE TRIGGER UPDATE_CUST_BALANCE_TRG
AFTER INSERT ON LINE
FOR EACH ROW
DECLARE
 CODE NUMBER;
BEGIN
SELECT C_CODE INTO CODE FROM INVOICE WHERE INV_NUM= :NEW.INV_NUM;
UPDATE CUSTOMER SET BALANCE = :NEW.L_UNITS * :NEW.L_PRICE
 WHERE CUSTOMER.C_CODE=CODE;
END;
/

Trigger created.

SQL> SELECT * FROM LINE WHERE INV_NUM = 1006;

   INV_NUM      L_NUM P_COD    L_UNITS    L_PRICE
---------- ---------- ----- ---------- ----------
      1006          1 MC001          3       6.99
      1006          2 JB012          1     109.92
      1006          3 CH10X          1       9.95
      1006          4 HC100          1     256.99
      1006          5 PP101         10       5.87
```

```
SQL> SELECT * FROM INVOICE WHERE INV_NUM = 1006;

   INV_NUM     C_CODE INV_DATE
---------- ---------- ---------
      1006      10014 17-JAN-20


SQL> delete from line where inv_num=1006 and p_code='PP101';

1 row deleted.

SQL> INSERT INTO LINE VALUES (1006,5,'PP101',10,5.87);

1 row created.


SQL> SELECT * FROM CUSTOMER WHERE C_CODE = 10014;

    C_CODE LNAME      FNAME          C_AREA    C_PHONE    BALANCE
---------- ---------- ---------- ---------- ---------- ----------
     10014 Johnson    Bill              615    2455533       58.7
```

```
==============================Query02================================
Write SQL code to compile and execute a trigger - SALARY_CHANGE_TRG, which
will monitor DML operations on SALARY attribute of EMPP table and will add a
record in SALARY_CHANGES table for each row affected by the DML statement.
Test the trigger by performing following DML operations on EMPP.
=====================================================================
PART A:-
CREATE TABLE SALARY_CHANGES(
 OP_TYPE VARCHAR2(10) NOT NULL,
 OP_DATE DATE DEFAULT SYSDATE,
 OP_TIME VARCHAR2(9) DEFAULT TO_CHAR (SYSTIMESTAMP,'HH24:MM:SS'),
 OLD_SAL NUMBER(8,2),
 NEW_SAL NUMBER(8,2),
 EID NUMBER (4) NOT NULL
);
Table created.

SQL> DESC SALARY_CHANGES;
 Name                                      Null?    Type
 ----------------------------------------- -------- ----------------------------
 OP_TYPE                                   NOT NULL VARCHAR2(10)
 OP_DATE                                            DATE
 OP_TIME                                            VARCHAR2(9)
 OLD_SAL                                            NUMBER(8,2)
 NEW_SAL                                            NUMBER(8,2)
 EID                                       NOT NULL NUMBER(4)
```

PART B:

```
CREATE OR REPLACE TRIGGER SALARY_CHANGE_TRG
AFTER INSERT OR UPDATE OF SALARY OR DELETE ON EMPP
FOR EACH ROW
BEGIN
 CASE WHEN INSERTING THEN
INSERT INTO SALARY_CHANGES(OP_TYPE,NEW_SAL,EID)
VALUES('INSERT',:NEW.SALARY,:NEW.ENO);
 WHEN UPDATING THEN
INSERT INTO SALARY_CHANGES(OP_TYPE,OLD_SAL,NEW_SAL,EID)
VALUES('UPDATE',:OLD.SALARY,:NEW.SALARY,:NEW.ENO);
 WHEN DELETING THEN
INSERT INTO SALARY_CHANGES(OP_TYPE,OLD_SAL,EID)
VALUES('DELETE',:OLD.SALARY,:OLD.ENO);
 END CASE;
END;
/
Trigger created.


ALTER TRIGGER SALARY_CHANGE_TRG DISABLE;
Trigger altered.
SELECT COUNT(*) FROM EMPP;
 COUNT(*)
----------
 17
SELECT COUNT(*) FROM SALARY_CHANGES;
 COUNT(*)
----------
 0
ALTER TRIGGER SALARY_CHANGE_TRG ENABLE;
Trigger altered.
INSERT INTO EMPP VALUES(7121,'Melody Malvankar',SYSDATE,'Asst.
Professor',80000);
1 row created.
INSERT INTO EMPP VALUES(7122,'Kalpak Gundappa',SYSDATE,'Research
Asst.',45000);
1 row created.
UPDATE EMPP SET SALARY=SALARY+2500 WHERE ENO>=7121;
2 rows updated.
DELETE FROM EMPP WHERE ENO=7122;
1 row deleted.
SELECT * FROM SALARY_CHANGES;
OP_TYPE OP_DATE        OP_TIME   OLD_SAL    NEW_SAL    EID
---------- --------- --------- ---------- ---------- ----------
INSERT     16-OCT-20 23:10:14                80000     7121
INSERT     16-OCT-20 23:10:20                45000     7122
UPDATE     16-OCT-20 23:10:25    80000       82500     7121
UPDATE     16-OCT-20 23:10:25    45000       47500     7122
DELETE     16-OCT-20 23:10:30    47500                 7122

ROLLBACK;
Rollback complete.

ALTER TRIGGER SALARY_CHANGE_TRG DISABLE;
Trigger altered.
```

==========Query03==========
Write SQL code to compile and execute a trigger - UPDATE_TOT_SAL_TRG, which will monitor DML operations on SALARY attribute of EMPP table and will keep EMP_SALARY table updated with the current total salary of the employee. When a new employee record is added in EMPP, a record in EMP_SALARY is also inserted with appropriate values. When employee salary is changed, the EMP_SALARY records for affected employees are updated. When an employee is removed from EMPP, the corresponding record in EMP_SALARY is not removed, but the STATUS filed is set to 'RETIRED'.
The TOT_SAL is computed as - (SALARY+PERKS-PF_Deductions) -IT_Deductions
KS are 25% of SALARY and PF_Deductions are fixed at 1200. The IT_Deductions are 10% of the cumulative of (Salary, Perks) minus PF_Deductions
Before testing UPDATE_TOT_SAL_TRG, disable the trigger - SALARY_CHANGE_TRG using the command... ALTER TRIGGER SALARY_CHANGE_TRG DISABLE; (which may be enabled when required)
Test UPDATE_TOT_SAL_TRG trigger by performing following DML operations on EMPP
==========

[PART -A]

```
CREATE TABLE EMP_SALARY AS
 (SELECT ENO , SALARY AS TOT_SAL FROM EMPP WHERE 1=2);
Table created.

ALTER TABLE EMP_SALARY
 ADD CONSTRAINT EMP_SALARY_PK_ENO PRIMARY KEY(ENO);
Table altered.

ALTER TABLE EMP_SALARY
 ADD STATUS VARCHAR2(7) DEFAULT 'ON_ROLL';

Table altered.

DESC EMP_SALARY;

DESC EMP_SALARY;
 Name                                      Null?    Type
 ----------------------------------------- -------- ----------------------
 ENO                                       NOT NULL NUMBER(4)
 TOT_SAL                                   NOT NULL NUMBER(8,2)
 STATUS                                             VARCHAR2(7)
```

```
INSERT INTO EMP_SALARY (ENO,TOT_SAL) SELECT ENO,((SALARY*1.25-1200)*0.90)
FROM EMPP;
17 rows created.
SELECT * FROM EMP_SALARY;
 ENO           TOT_SAL STATUS
---------- ---------- -------
 7101        167670 ON_ROLL
 7102      163732.5 ON_ROLL
 7103        165420 ON_ROLL
 7104        154620 ON_ROLL
 7105        142245 ON_ROLL
 7106        142245 ON_ROLL
 7107        142245 ON_ROLL
 7108      133582.5 ON_ROLL
 7109        101295 ON_ROLL
 7110         96120 ON_ROLL
 7111         53145 ON_ROLL
 7112         49095 ON_ROLL
 7113         38970 ON_ROLL
 7114      35876.25 ON_ROLL
 7115         32670 ON_ROLL
 7116         32670 ON_ROLL
 7117         35145 ON_ROLL
17 rows selected.

[PART-B]
CREATE OR REPLACE TRIGGER UPDATE_TOT_SAL_TRG
AFTER INSERT OR UPDATE OF SALARY OR DELETE ON EMPP
FOR EACH ROW
DECLARE
N NUMBER;
BEGIN
 CASE
 WHEN INSERTING THEN
 N:= (:NEW.SALARY*1.25-1200)*0.90;
 INSERT INTO EMP_SALARY(ENO,TOT_SAL) VALUES(:NEW.ENO,N);
 WHEN UPDATING THEN
 N:= (:NEW.SALARY*1.25-1200)*0.90;
 UPDATE EMP_SALARY SET TOT_SAL=N WHERE ENO=:NEW.ENO;
 WHEN DELETING THEN
UPDATE EMP_SALARY SET STATUS='RETIRED' WHERE
ENO=:OLD.ENO;
 END CASE;
END;
/
Trigger created.

ALTER TRIGGER UPDATE_TOT_SAL_TRG ENABLE;
Trigger altered.
SELECT COUNT(*) FROM EMPP;
 COUNT(*)
----------
 17
SELECT COUNT(*) FROM EMP_SALARY;
 COUNT(*)
----------
 17
INSERT INTO EMPP VALUES(7121,'Melody Malvankar',SYSDATE,'Asst.
```

```
Professor',80000);
1 row created.
INSERT INTO EMPP VALUES(7122,'Kalpak Gundappa',SYSDATE,'Research
Asst.',45000);
1 row created.
UPDATE EMPP SET SALARY=SALARY+2500 WHERE ENO>=7121;
2 rows updated.
DELETE FROM EMPP WHERE ENO=7122;
1 row deleted.
SELECT COUNT(*) FROM EMPP;
 COUNT(*)
----------
 18
SELECT COUNT(*) FROM EMP_SALARY;
 COUNT(*)
----------
 19



SELECT * FROM EMP_SALARY;
 ENO        TOT_SAL    STATUS
---------- ---------- -------
 7121        91732.5  ON_ROLL
 7122        52357.5 RETIRED
 7101         167670 ON_ROLL
 7102       163732.5 ON_ROLL
 7103         165420 ON_ROLL
 7104         154620 ON_ROLL
 7105         142245 ON_ROLL
 7106         142245 ON_ROLL
 7107         142245 ON_ROLL
 7108       133582.5 ON_ROLL
 7109         101295 ON_ROLL
 7110          96120 ON_ROLL
 7111          53145 ON_ROLL
 7112          49095 ON_ROLL
 7113          38970 ON_ROLL
 7114       35876.25 ON_ROLL
 7115          32670 ON_ROLL
 7116          32670 ON_ROLL
 7117          35145 ON_ROLL
19 rows selected.
ALTER TRIGGER UPDATE_TOT_SAL_TRG DISABLE;
Trigger altered.
```

```
=====================================Query04============================
Write SQL code to compile and execute a trigger - LINE_INS_UPD_QTY_TRG that
will automatically update the quantity on hand (QTY) for each product sold
after a new LINE row is added.
========================================================================
CREATE OR REPLACE TRIGGER LINE_INS_UPD_QTY_TRG
AFTER INSERT ON LINE
FOR EACH ROW
BEGIN
 UPDATE PRODUCT
 SET QTY= QTY - :NEW.L_UNITS WHERE P_CODE=:NEW.P_CODE;
END;
/
Trigger created.


SELECT P_CODE, DESCRIPT , QTY FROM PRODUCT
WHERE P_CODE = 'RF100';
P_COD DESCRIPT                         QTY
----- ------------------------------ ----------
RF100 Rat Tail File                      43

SELECT INV_NUM,L_NUM,P_CODE,L_UNITS
FROM LINE WHERE INV_NUM=1005;
 INV_NUM    L_NUM      P_COD L_UNITS
---------- ---------- ----- ----------
 1005           1      PP101      12
INSERT INTO LINE VALUES(1005,2,'RF100',20,4.99);
1 row created.


SELECT INV_NUM,L_NUM,P_CODE,L_UNITS
FROM LINE WHERE INV_NUM=1005;
 INV_NUM        L_NUM P_COD    L_UNITS
---------- ---------- ----- ----------
 1005               1 PP101        12
 1005               2 RF100        20

SELECT P_CODE, DESCRIPT , QTY FROM PRODUCT
WHERE P_CODE = 'RF100';
P_COD DESCRIPT                           QTY
----- ------------------------------ ----------
RF100 Rat Tail File                      23
```

==========Query05==========

Write SQL code to compile and execute a statement level trigger - CHECK_REORDER_STATUS_TRG that will keep check on REORDER flag in PRODUCT_T table (set to 1) when the product quantity on hand (QTY) falls below the minimum quantity (P_MIN) in stock. You must ensure that if the P_MIN is updated (such that QTY > P_MIN) the REORDER flag should be toggled.
Now modify the trigger CHECK_REORDER_STATUS_TRG to a row level trigger - CHECK_REORDER_STATUS_TRG_RL such that it also handles the updating to QTY values (i.e., while REORDER flag is 1, QTY is updated and QTY > P_MIN).

==========

```
CREATE TABLE PRODUCT_T AS
(SELECT P_CODE , DESCRIPT ,QTY,P_MIN,P_PRICE,V_CODE FROM PRODUCT
WHERE 1=2 );
Table created.
ALTER TABLE PRODUCT_T ADD REORDER NUMBER;


Table altered
ALTER TABLE PRODUCT_T  MODIFY REORDER DEFAULT 0;

DESC PRODUCT_T;
 Name Null? Type
 ----------------------------------------- -------- --------------------
 P_CODE                                    NOT NULL CHAR(5)
 DESCRIPT                                  NOT NULL VARCHAR2(30)
 QTY                                       NOT NULL NUMBER(4)
 P_MIN                                     NOT NULL NUMBER(3)
 P_PRICE                                   NOT NULL NUMBER(6,2)
 V_CODE                                             NUMBER(5)
 REORDER                                            NUMBER

INSERT INTO PRODUCT_T (P_CODE , DESCRIPT ,QTY,P_MIN,P_PRICE,V_CODE) SELECT
P_CODE , DESCRIPT ,QTY,P_MIN,P_PRICE,V_CODE FROM PRODUCT;
22 rows created.
[Part-b]
CREATE OR REPLACE TRIGGER CHECK_REORDER_STATUS_TRG
AFTER UPDATE OF QTY,P_MIN ON PRODUCT_T
DECLARE P PRODUCT_T %ROWTYPE;
BEGIN
 FOR P IN (SELECT P_CODE,QTY,P_MIN FROM PRODUCT_T) LOOP
 IF P.QTY<=P.P_MIN THEN
 UPDATE PRODUCT_T SET REORDER=1
 WHERE P_CODE = P.P_CODE;
 ELSE
 UPDATE PRODUCT_T SET REORDER=0
 WHERE P_CODE = P.P_CODE;
 END IF;
 END LOOP;
END;
/
Trigger created.
ALTER TRIGGER CHECK_REORDER_STATUS_TRG ENABLE;
Trigger altered.
SELECT P_CODE,QTY,P_MIN,REORDER FROM PRODUCT_T WHERE P_CODE='JB008';
P_COD        QTY      P_MIN   REORDER
----- ---------- ---------- ----------
JB008          6          5         0
```

```
UPDATE PRODUCT_T SET QTY=QTY-2 WHERE P_CODE = 'JB008';
1 row updated.

SELECT P_CODE,QTY,P_MIN,REORDER FROM PRODUCT_T WHERE P_CODE='JB008';
P_COD        QTY     P_MIN    REORDER
----- ---------- ---------- ----------
JB008         4          5          1


UPDATE PRODUCT_T SET QTY=QTY+1 WHERE P_CODE = 'JB008';


1 row updated.
SELECT P_CODE,QTY,P_MIN,REORDER FROM PRODUCT_T WHERE P_CODE='JB008';
P_COD        QTY      P_MIN    REORDER
----- ---------- ---------- ----------
JB008          5          5          1

SELECT P_CODE,QTY,P_MIN,REORDER FROM PRODUCT_T WHERE P_CODE='SH100';
P_COD        QTY      P_MIN   REORDER
----- ---------- ---------- ----------
SH100          6          5         0

UPDATE PRODUCT_T SET P_MIN = P_MIN+3 WHERE P_CODE = 'SH100';
1 row updated

SELECT P_CODE,QTY,P_MIN,REORDER FROM PRODUCT_T WHERE P_CODE='SH100';
P_COD        QTY      P_MIN   REORDER
----- ---------- ---------- ----------
SH100          6          8         1


UPDATE PRODUCT_T SET P_MIN = P_MIN-2 WHERE P_CODE = 'SH100';
1 row updated.
SELECT P_CODE,QTY,P_MIN,REORDER FROM PRODUCT_T WHERE P_CODE='SH100';
P_COD        QTY      P_MIN    REORDER
----- ---------- ---------- ----------
SH100          6          6          1
```

Viva Voice
========
****************************** VIVA-VOCE **********************************

Q1.Differentiate between a statement-level and a row-level trigger

ANS :

Row Level Trigger

Row Level Trigger is fired each time row is affected by Insert, Update or Delete command. If statement doesn't affect any row, no trigger action happens.

Statement Level Trigger

This kind of trigger fires when a SQL statement affects the rows of the table. The trigger activates and performs its activity irrespective of number of rows affected due to SQL statement.

****

Q2. How many different triggers a table can have? List all of these.
Trigger is stored in the database which includes SQL and PL/SQL or Java statements to run as a unit and invokes stored procedures. Triggers are implicitly fired by Oracle when a triggering event occurs, no matter which user is connected or which application is being used. There are 12 types of triggers can exist in a table in Oracle: 3 before statement, 3 after statement, 3 before each row and 3 and 3 after each row.

****

Q3.What are cascading triggers?

Ans:

The database server allows triggers other than Select triggers to cascade, meaning that the trigger actions of one trigger can activate another trigger.

****

Q4. Why COMMIT and ROLLBACK cannot be used in triggers? Can a trigger call a stored function or procedures that perform a COMMIT or a ROLLBACK?.

Ans:- See trigger, Not only do triggers not need a COMMIT you can't put one in: a

trigger won't compile if the body's code includes a COMMIT (or a rollback).This is because triggers fire during a transaction. When the trigger fires the current transaction is still not complete. As COMMIT terminates a transaction allowing them in triggers would break the unit of work.So, changes executed in a trigger are committed (or rolled back) by the owning transaction which issued the DML that fired the trigger.

****

Q5.Is it possible to create a trigger that will fire when a row is read during

a query?

Ans:

When a trigger is fired, the tables referenced in the trigger action might be currently undergoing changes by SQL statements in other users' transactions. In all cases, the SQL statements executed within triggers follow the common rules used for standalone SQL statements. In particular, if an uncommitted transaction has modified values that a trigger being fired either needs to read (query) or write (update), the SQL statements in the body of the trigger being fired use the following guidelines:
Queries see the current read-consistent snapshot of referenced tables and any data changed within the same transaction.
Updates wait for existing data locks to be released before proceeding.

****

==========

Inferences

==========

1.Learn working of trigger with dml statement.
2.Learn trigger with cursor.