*** EXPERIMENT NO: 08 ***

Author: Atharva Paliwal

Roll No: 40 [5B]

Date: 07-November-2020

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;
/
```

```
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
```

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

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

```
SELECT * FROM CUSTOMER WHERE C_CODE=10014;

   C_CODE LNAME      FNAME          C_AREA    C_PHONE    BALANCE
--------- ---------- ---------- ---------- ---------- ----------
   10014  Johnson    Bill              615    2455533          0

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

SELECT * FROM CUSTOMER WHERE C_CODE=10014;

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

****

QUERY 02: 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.

****

```
CREATE TABLE SALARY_CHANGES(
   OP_TYPE VARCHAR2(10) NOT NULL,
   OP_DATE DATE DEFAULT SYSDATE,
   OP_TIME CHAR(9) DEFAULT TO_CHAR(SYSTIMESTAMP, 'HH:MI:SS') ,
   OLD_SAL NUMBER(8,2),
   NEW_SAL NUMBER(8,2),
   EID NUMBER(4) NOT NULL
   );

SELECT COUNT(*) FROM EMPP;

   COUNT(*)
 ----------
        17

SELECT COUNT(*) FROM SALARY_CHANGES;
   COUNT(*)
 ----------
         0

CREATE OR REPLACE TRIGGER SALARY_CHANGE_TRG
   AFTER INSERT OR UPDATE OR DELETE ON EMPP
   FOR EACH ROW
BEGIN
   CASE
      WHEN INSERTING THEN
         DBMS_OUTPUT.PUT_LINE('THE INSERT ENTRY IS LOGGED IN
         SALARY_CHANGES TABLE ');
         INSERT INTO SALARY_CHANGES(OP_TYPE,NEW_SAL,EID) VALUES
         ('INSERT',:NEW.SALARY,:NEW.EID);
      WHEN DELETING THEN
         DBMS_OUTPUT.PUT_LINE('THE DELETE ENTRY IS LOGGED IN
         SALARY_CHANGES TABLE ');
         INSERT INTO SALARY_CHANGES(OP_TYPE,OLD_SAL,EID) VALUES
         ('DELETE',:OLD.SALARY,:OLD.EID);
      WHEN UPDATING('SALARY') THEN
         DBMS_OUTPUT.PUT_LINE(' THE UPDATE ENTRY IS LOGGED IN
         SALARY_CHANGES TABLE ');
         INSERT INTO SALARY_CHANGES(OP_TYPE,OLD_SAL,NEW_SAL,EID) VALUES
         ('UPDATE',:OLD.SALARY,:NEW.SALARY,:NEW.EID);
   END CASE;
END;
/

ALTER TRIGGER SALARY_CHANGE_TRG ENABLE;

INSERT INTO EMPP VALUES (7121, 'Melody Malvankar', SYSDATE,'Asst. Professor',
80000);
THE INSERT ENTRY IS LOGGED IN SALARY_CHANGES TABLE

1 row created.

INSERT INTO EMPP VALUES (7122, 'Kalpak Gundappa', SYSDATE,'Research Asst.',
45000);
THE INSERT ENTRY IS LOGGED IN SALARY_CHANGES TABLE

1 row created.

UPDATE EMPP SET SALARY = SALARY+2500 WHERE EID>=7121;
THE UPDATE ENTRY IS LOGGED IN SALARY_CHANGES TABLE
THE UPDATE ENTRY IS LOGGED IN SALARY_CHANGES TABLE

2 rows updated.

DELETE FROM EMPP WHERE EID=7122;
THE DELETE ENTRY IS LOGGED IN SALARY_CHANGES TABLE

1 row deleted.

SELECT COUNT(*) FROM EMPP;

   COUNT(*)
 ----------
        18

SELECT COUNT(*) FROM SALARY_CHANGES;
   COUNT(*)
 ----------
          5

SELECT * FROM SALARY_CHANGES;

OP_TYPE    OP_DATE   OP_TIME      OLD_SAL    NEW_SAL        EID
---------- --------- --------- ---------- ---------- ----------
INSERT     22-OCT-20 01:35:21                  80000       7121
INSERT     22-OCT-20 01:35:29                  45000       7122
UPDATE     22-OCT-20 01:35:38       80000      82500       7121
UPDATE     22-OCT-20 01:35:38       45000      47500       7122
DELETE     22-OCT-20 01:35:48       47500                  7122

ROLLBACK;

Rollback complete.
```

```
ALTER TRIGGER SALARY_CHANGE_TRG DISABLE;

Trigger altered.

CREATE TABLE EMP_SALARY AS
   SELECT EID,SALARY AS TOT_SAL
   FROM EMPP WHERE 1=2;

Table created.

INSERT INTO EMP_SALARY (EID, TOT_SAL)
   SELECT EID, (SALARY*1.25 -12000)*0.90
   FROM EMPP;

ALTER TABLE EMP_SALARY
   ADD CONSTRAINT EMP_SALARY_PK_EID PRIMARY KEY (EID);

Table altered.

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

Table altered.

SELECT * FROM EMP_SALARY;

       EID    TOT_SAL   STATUS
---------- ---------- -------
      7102   154012.5 ON_ROLL
      7101     157950 ON_ROLL
      7103     155700 ON_ROLL
      7104     144900 ON_ROLL
      7107     132525 ON_ROLL
      7105     132525 ON_ROLL
      7106     132525 ON_ROLL
      7108   123862.5 ON_ROLL
      7109      91575 ON_ROLL
      7110      86400 ON_ROLL
      7111      43425 ON_ROLL
      7112      39375 ON_ROLL
      7113      29250 ON_ROLL
      7114   26156.25 ON_ROLL
      7115      22950 ON_ROLL
      7116      22950 ON_ROLL
      7117      25425 ON_ROLL
```

****
QUERY 03: 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'.
****

```
SELECT COUNT(*) FROM EMPP;

   COUNT(*)
 ----------
         17

SELECT COUNT(*) FROM EMP_SALARY;

   COUNT(*)
 ----------
         17

CREATE OR REPLACE TRIGGER UPDATE_TOT_SAL_TRG
   AFTER INSERT OR UPDATE OR DELETE ON EMPP
   FOR EACH ROW
BEGIN
   CASE
   WHEN INSERTING THEN
      INSERT INTO EMP_SALARY(EID,TOT_SAL) VALUES(:NEW.EID,(:NEW.SALARY*1.25-
      1200)*0.90);
   WHEN UPDATING('SALARY') THEN
      UPDATE EMP_SALARY SET TOT_SAL = (:NEW.SALARY*1.25-1200)*0.90 WHERE EID =
      :OLD.EID;
   WHEN DELETING THEN
      UPDATE EMP_SALARY SET STATUS = 'RETIRED' WHERE EID = :OLD.EID;
   END CASE;
END;
/

Trigger created.

ALTER TRIGGER UPDATE_TOT_SAL_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 EID>=7121;

2 rows updated.

DELETE FROM EMPP WHERE EID=7122;

1 row deleted.

SELECT COUNT(*) FROM EMP_SALARY;

   COUNT(*)
 ----------
         19
```

```
SELECT COUNT(*) FROM EMPP;

   COUNT(*)
 ----------
         18

SELECT * FROM EMP_SALARY;

      EID    TOT_SAL STATUS
---------- ---------- -------
      7102   154012.5 ON_ROLL
      7101     157950 ON_ROLL
      7103     155700 ON_ROLL
      7104     144900 ON_ROLL
      7107     132525 ON_ROLL
      7105     132525 ON_ROLL
      7106     132525 ON_ROLL
      7108   123862.5 ON_ROLL
      7109      91575 ON_ROLL
      7110      86400 ON_ROLL
      7111      43425 ON_ROLL
      7112      39375 ON_ROLL
      7113      29250 ON_ROLL
      7114   26156.25 ON_ROLL
      7115      22950 ON_ROLL
      7116      22950 ON_ROLL
      7117      25425 ON_ROLL
      7121    91732.5 ON_ROLL
      7122    52357.5 RETIRED

ALTER TRIGGER UPDATE_TOT_SAL_TRG DISABLE;


Trigger altered.


*********************************************************************************
QUERY 04: 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

CREATE TABLE PRODUCT_T AS
   SELECT P_CODE,DESCRIPT,QTY,P_MIN,P_PRICE,V_CODE
   FROM PRODUCT;

ALTER TABLE PRODUCT_T
   ADD REORDER NUMBER(1) DEFAULT 0;

COMMIT;
```

****

QUERY 05: 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 OR REPLACE TRIGGER CHECK_REORDER_STATUS_TRG
   AFTER INSERT OR UPDATE OF QTY, P_MIN ON PRODUCT_T
DECLARE
   PROD PRODUCT_T%ROWTYPE;
BEGIN
   FOR PROD IN (SELECT P_CODE, QTY, P_MIN FROM PRODUCT_T) LOOP
      IF PROD.QTY<=PROD.P_MIN THEN
         UPDATE PRODUCT_T
         SET REORDER=1
         WHERE P_CODE =PROD.P_CODE;
      ELSE
```

```
          UPDATE PRODUCT_T
             SET REORDER=0
             WHERE P_CODE= PROD.P_CODE;
       END IF;
   END LOOP;
END;
/

CREATE OR REPLACE TRIGGER CHECK_REORDER_STATUS_TRG_RL
   BEFORE INSERT OR UPDATE OF QTY, P_MIN ON PRODUCT_T
   FOR EACH ROW
BEGIN
   IF :NEW.QTY <= :NEW.P_MIN THEN
      :NEW.REORDER := 1;
   ELSE
      :NEW.REORDER :=0;
   END IF;
END;
/


ALTER TRIGGER CHECK_REORDER_STATUS_TRG DISABLE;

SELECT P_CODE, QTY, P_MIN, REORDER FROM PRODUCT_T WHERE P_CODE = 'JB008';

P_COD        QTY      P_MIN    REORDER
----- ---------- ---------- ----------
JB008          6          5          0
```

CHECK_REORDER_STATUS_TRG :

```
ALTER TRIGGER CHECK_REORDER_STATUS_TRG ENABLE;

UPDATE PRODUCT_T SET QTY = QTY - 2 WHERE P_CODE = 'JB008';

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';

SELECT P_CODE, QTY, P_MIN, REORDER FROM PRODUCT_T WHERE P_CODE = 'JB008';

P_COD        QTY      P_MIN    REORDER
----- ---------- ---------- ----------
JB008          5          5          1


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

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 P_MIN = P_MIN + 2 WHERE P_CODE = 'JB008';
```

SELECT P_CODE, QTY, P_MIN, REORDER FROM PRODUCT_T WHERE P_CODE = 'JB008';

```
P_COD        QTY      P_MIN    REORDER
----- ---------- ---------- ----------
JB008          6          7          1
```

UPDATE PRODUCT_T SET P_MIN = P_MIN - 1,QTY= QTY + 2 WHERE P_CODE = 'JB008';

SELECT P_CODE, QTY, P_MIN, REORDER FROM PRODUCT_T WHERE P_CODE = 'JB008';

```
P_COD        QTY      P_MIN    REORDER
----- ---------- ---------- ----------
JB008          8          6          0
```

ALTER TRIGGER CHECK_REORDER_STATUS_TRG DISABLE;

CHECK_REORDER_STATUS_TRG_RL:

ALTER TRIGGER CHECK_REORDER_STATUS_TRG_RL ENABLE;
SELECT P_CODE, QTY, P_MIN, REORDER FROM PRODUCT_T WHERE P_CODE = 'SH100';

```
P_COD        QTY      P_MIN    REORDER
----- ---------- ---------- ----------
SH100          8          5          0
```

UPDATE PRODUCT_T SET QTY = QTY - 3 WHERE P_CODE = 'SH100';

SELECT P_CODE, QTY, P_MIN, REORDER FROM PRODUCT_T WHERE P_CODE = 'SH100';

```
P_COD        QTY      P_MIN    REORDER
----- ---------- ---------- ----------
SH100          5          5          1
```

UPDATE PRODUCT_T SET QTY = QTY + 1 WHERE P_CODE = 'SH100';

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';

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';

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

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

Row-level triggers for data-related activities
• Row-level triggers execute once for each row in a transaction.
• Row-level triggers are the most common type of triggers; they are often used in data auditing applications.
• Row-level trigger is identified by the FOR EACH ROW clause in the CREATE TRIGGER command.
Statement-level triggers for transaction-related activities
• Statement-level triggers execute once for each transaction. For example, if a single transaction inserted 500 rows into the Customer table, then a statement level trigger on that table would only be executed once.
• Statement-level triggers therefore are not often used for data-related activities; they are normally used to enforce additional security measures on the types of transactions that may be performed on a table.
• Statement-level triggers are the default type of triggers created and are identified by omitting the FOR EACH ROW clause in the CREATE TRIGGER command.

Q2.. How many different triggers a table can have? List all of these.

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 after each row.
In a single table you can define as many triggers as you need

Classification based on the timing
BEFORE Trigger: It fires before the specified event has occurred.
AFTER Trigger: It fires after the specified event has occurred.
INSTEAD OF Trigger: A special type. You will learn more about the further topics. (only for DML)
Classification based on the level
STATEMENT level Trigger: It fires one time for the specified event statement.
ROW level Trigger: It fires for each record that got affected in the specified event. (only for DML)
Classification based on the Event
DML Trigger: It fires when the DML event is specified (INSERT/UPDATE/DELETE)
DDL Trigger: It fires when the DDL event is specified (CREATE/ALTER)
DATABASE Trigger: It fires when the database event is specified (LOGON/LOGOFF/STARTUP/SHUTDOWN)

Q3. What are cascading triggers?

At times when SQL statement of a trigger can fire other triggers. This results in cascading triggers. Oracle allows around 32 cascading triggers. Cascading triggers can cause result in abnormal behavior of the application.

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?

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
Triggers can not affect the current transaction, so they can not contain COMMIT or ROLLBACK statements. If you need some code to perform an operation that needs to commit, regardless of the current transaction, you should put it in a stored procedure defined as an autonomous transaction.

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

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

- We learnt about database triggers.
- We executed SQL programs using trigger.

** END **