```
===========================================================================================
                                    EXPERIMENT NO. 08
===========================================================================================

  Author  : Diksha Gupta.
  Roll no.: 08 [27A]
  Date    : 09-DECEMBER-2022.

===========================================================================================
```

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 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
BEFORE INSERT ON LINE
FOR EACH ROW
DECLARE
       CV_CODE CUSTOMER.C_CODE%TYPE;
BEGIN
       SELECT DISTINCT(C_CODE) INTO CV_CODE
           FROM LINE
           NATURAL JOIN INVOICE
           NATURAL JOIN CUSTOMER
           WHERE INV_NUM IN (:NEW.INV_NUM);

       UPDATE CUSTOMER SET BALANCE = BALANCE + (:NEW.L_UNITS * :NEW.L_PRICE)
       WHERE C_CODE = CV_CODE;
END;
/
Trigger created.
```

```
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);

1 row created.

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.
              Add : 7121, Melody Malvankar, SYSDATE, 80000, Asst. Professor
              Add : 7122, Kalpak Gundappa, SYSDATE, 45000, Research Asst.
              Modify : SALARY = SALARY + 2500 for ENO >= 7121
              Remove : ENO = 7122;
********************************************************************************************
```

```
SELECT COUNT(*) FROM EMPP;

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

SELECT COUNT(*) FROM SALARY_CHANGES;

  COUNT(*)
----------
         0

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(*) COUNT
     FROM SALARY_CHANGES;

     COUNT
----------
         5

SELECT *
    FROM SALARY_CHANGES;

OP_TYPE    OP_DATE   OP_TIME      OLD_SAL    NEW_SAL        EID
---------- --------- --------- ---------- ---------- ----------
INSERT     07-DEC-22 09:21:11                  80000       7121
INSERT     07-DEC-22 09:23:47                  45000       7122
UPDATE     07-DEC-22 09:23:56       80000      82500       7121
UPDATE     07-DEC-22 09:23:56       45000      47500       7122
DELETE     07-DEC-22 09:27:16       47500                  7122
```

*************************************** **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'.

The TOT_SAL is computed as(SALARY+PERKS-PF_Deductions)-IT_Deductions. PERKS 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 -
              Add : 7121, Melody Malvankar, SYSDATE, 80000, Asst. Professor
              Add : 7122, Kalpak Gundappa, SYSDATE, 45000, Research Asst.
              Modify : SALARY = SALARY + 2500 for ENO >= 7121
              Remove : ENO = 7122;

*********************************************************************************************

SELECT COUNT(*) FROM EMPP;

  COUNT(*)
----------
        19
SELECT COUNT(*) FROM EMP_SALARY;

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

SET SERVEROUTPUT ON;

CREATE OR REPLACE TRIGGER UPDATE_TOT_SALARY_TRG
   BEFORE DELETE OR INSERT OR UPDATE ON EMPP
   FOR EACH ROW

DECLARE
   V_SAL EMPP.SALARY%TYPE;

BEGIN

   IF DELETING THEN
      DBMS_OUTPUT.PUT_LINE('THE DELETE ENTRY IS LOGGED IN EMP_SALARY TABLE');
      UPDATE EMP_SALARY SET STATUS = 'RETIRED' WHERE ENO = (:OLD.EID);

   ELSIF INSERTING THEN
     V_SAL := (:NEW.SALARY + ((:NEW.SALARY/100)*25)-1200)-
              (((((:NEW.SALARY+(:NEW.SALARY/100)*25))/100)*10)-1200);
     DBMS_OUTPUT.PUT_LINE('THE INSERT ENTRY IS LOGGED IN EMP_SALARY TABLE');
     INSERT INTO EMP_SALARY(ENO,TOT_SAL)VALUES(:NEW.EID,V_SAL);
   ELSIF UPDATING('SALARY')THEN
     V_SAL := (:NEW.SALARY + ((:NEW.SALARY/100)*25)-1200)-
              (((((:NEW.SALARY+(:NEW.SALARY/100)*25))/100)*10)-1200);
     DBMS_OUTPUT.PUT_LINE('THE UPDATE ENTRY IS LOGGED IN EMP_SALARY TABLE');
     UPDATE EMP_SALARY SET TOT_SAL = V_SAL  WHERE ENO = (:OLD.EID);

   ELSE
     INSERT INTO EMP_SALARY(ENO,TOT_SAL)VALUES(:NEW.EID,:NEW.SALARY);
   END IF;
END;
/

INSERT INTO EMPP
   VALUES (7121, 'Melody Malvankar', SYSDATE, 'Asst. Professor', 80000);

THE INSERT ENTRY IS LOGGED IN EMP_SALARY TABLE

1 row created.

INSERT INTO EMPP
   VALUES (7122, 'Kalpak Gundappa', SYSDATE, 'Research Asst.', 4500);

THE INSERT ENTRY IS LOGGED IN EMP_SALARY TABLE

1 row created.

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

2 rows updated.

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

1 row deleted.

SELECT COUNT(*) FROM EMPP;

  COUNT(*)
----------
        20
```

```
SELECT COUNT(*) FROM EMP_SALARY;

  COUNT(*)
----------
        21

SELECT * FROM
     EMP_SALARY ORDER BY ENO;

       ENO    TOT_SAL STATUS
---------- ---------- -------
      7101     167670 ON_ROLL
      7102   163732.5 ON_ROLL
      7103     165420 ON_ROLL
      7104     154620 ON_ROLL
        :            :
        :            :
      7121    92812.5 ON_ROLL
      7122    53437.5 RETIRED

21 rows selected.
```

*************************************** **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
BEFORE 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 P_CODE, DESCRIPT, QTY FROM PRODUCT
  WHERE P_CODE = 'RF100';

P_COD DESCRIPT                              QTY
----- ------------------------------ ----------
RF100 Rat Tail File                          23


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
```

```
*************************************** 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 UPDATE OF P_MIN ON PRODUCT_T
DECLARE
        PROD PRODUCT_T%ROWTYPE;
BEGIN
        FOR PROD IN (SELECT * FROM PRODUCT_T)
        LOOP
                IF(PROD.QTY>PROD.P_MIN) THEN
                        UPDATE PRODUCT_T
                        SET REORDER=0
                        WHERE P_CODE=PROD.P_CODE;
                ELSE
                        UPDATE PRODUCT_T
                        SET REORDER=1
                        WHERE P_CODE=PROD.P_CODE;
                END IF;
        END LOOP;
END;
/

Trigger created.


SELECT *
    FROM PRODUCT_T
       WHERE P_CODE='JB008';

P_COD            DESCRIPT         QTY       P_MIN     P_PRICE      V_CODE     REORDER
----- ------------------ ---------- ---------- ---------- ---------- ----------
JB008    Jigsaw 8in Blade           6           5       99.87       24288           0
```

```
UPDATE PRODUCT_T SET P_MIN=P_MIN+2
      WHERE P_CODE='JB008';

1 row updated.

SELECT *
   FROM PRODUCT_T WHERE P_CODE='JB008';

P_COD           DESCRIPT         QTY       P_MIN     P_PRICE      V_CODE     REORDER
----- ------------------ ---------- ---------- ---------- ---------- ----------
JB008   Jigsaw 8in Blade           6           7       99.87       24288           1

ROLLBACK;

Rollback complete.

CREATE OR REPLACE TRIGGER CHECK_REORDER_STATUS_TRG_RL
AFTER UPDATE OF QTY,P_MIN ON PRODUCT_T
FOR EACH ROW
BEGIN
       IF :NEW.QTY>:NEW.P_MIN THEN
              UPDATE PRODUCT_T
              SET REORDER=0
              WHERE P_CODE=:NEW.P_CODE;
       ELSIF :NEW.QTY<:NEW.P_MIN THEN
              UPDATE PRODUCT_T
              SET REORDER=1
              WHERE P_CODE=:NEW.P_CODE;
       END IF;
END;
/

Trigger created.


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

1 row updated.
```

```
SELECT *
    FROM PRODUCT_T
         WHERE P_CODE='JB008';

P_COD           DESCRIPT        QTY      P_MIN     P_PRICE     V_CODE    REORDER
----- ------------------ ---------- ---------- ---------- ---------- ----------
JB008   Jigsaw 8in Blade          4          5       99.87      24288
```

==

INFERENCES OF THE EXPERIMENT

==

Hence , we have successfully write and execute SQL programs that allows enforcement of business rules with database triggers.