***** EXPERIMENT NO: 07 *****

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.

Author : Bhavesh Kewalramani

Roll No : 025 [5A]

Date : 04-October-2021

****

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 LI NE record: 1006, 5, 'PP101', 10, 5.87.

****

```
CREATE OR REPLACE TRIGGER UPDATE_CUST_BALANCE_TRG
      AFTER INSERT ON LINE
      FOR EACH ROW
BEGIN
      UPDATE CUSTOMER SET BALANCE = BALANCE + (:NEW.L_UNITS * :NEW.L_PRICE);
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 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.

****

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

```
                DBMS_OUTPUT.PUT_LINE('THE UPDATE ENTRY IS LOGGED
                                                IN SALARY_CHANGES TABLE');
        ELSE
                INSERT INTO SALARY_CHANGES(OP_TYPE,NEW_SAL,EID)
                VALUES ('INSERT',:NEW.SALARY,:NEW.EID);
                DBMS_OUTPUT.PUT_LINE('THE INSERT ENTRY IS LOGGED
                                                IN SALARY_CHANGES TABLE');
        END IF;
    END;
    /

Trigger created.

    SELECT COUNT(*)
    FROM EMPP;

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

    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(*)
      FROM EMPP;
```

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

```
      SELECT COUNT(*)
      FROM SALARY_CHANGES;
```

```
  COUNT(*)
----------
         5
```

```
      SELECT *
      FROM SALARY_CHANGES
      ORDER BY OP_TIME;
```

```
OP_TYPE    OP_DATE   OP_TIME      OLD_SAL    NEW_SAL        EID
---------- --------- --------- ---------- ---------- ----------
INSERT     06-OCT-21 03:05:28                 80000       7121
INSERT     06-OCT-21 03:07:15                 45000       7122
UPDATE     06-OCT-21 03:09:31       80000     82500       7121
UPDATE     06-OCT-21 03:09:31       45000     47500       7122
DELETE     06-OCT-21 03:10:27       47500                 7122

5 rows selected.
```

****

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)

****

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

```
            END IF;
        END;
        /

Trigger created.

        SELECT COUNT(*)
        FROM EMPP;

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

        SELECT COUNT(*)
        FROM EMP_SALARY;

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

        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(*)
----------
        20

SELECT COUNT(*)
FROM EMPP;

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

SELECT *
FROM EMP_SALARY;

       EID    TOT_SAL STATUS
---------- ---------- -------
      7101     167670 ON_ROLL
      7102   163732.5 ON_ROLL
      7103     165420 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
      7118      27045 ON_ROLL
      7119     181170 ON_ROLL
      7121    91732.5 ON_ROLL
      7122    52357.5 RETIRED

20 rows selected.
```

```
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
DECLARE
    QT PRODUCT.QTY%TYPE;
BEGIN
    SELECT QTY INTO QT
    FROM PRODUCT
    WHERE P_CODE = :NEW.P_CODE;
    IF QT >= :NEW.L_UNITS THEN
            UPDATE PRODUCT
            SET QTY = QTY - :NEW.L_UNITS
            WHERE P_CODE = :NEW.P_CODE;
    ELSE
            DBMS_OUTPUT.PUT_LINE('PRODUCT QUANTITY IS NOT AVAILABLE
                                                    RIGHT NOW ... ');
    END IF;
END;
/
```

```
Trigger created.
```

```
SELECT P_CODE, DESCRIPT, QTY, P_PRICE
FROM PRODUCT
WHERE P_CODE='RF100';
```

```
P_COD DESCRIPT                          QTY    P_PRICE
----- ------------------------------ ---------- ----------
RF100 Rat Tail File                          43       4.99
```

```
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, P_PRICE
      FROM PRODUCT
      WHERE P_CODE='RF100';
```

```
P_COD DESCRIPT                             QTY    P_PRICE
----- ------------------------------ ---------- ----------
RF100 Rat Tail File                          23       4.99
```

****

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 QTY, P_MIN
            ON PRODUCT_T
      BEGIN
```

```
            UPDATE PRODUCT_T
            SET REORDER = 1
            WHERE QTY <= P_MIN;

            UPDATE PRODUCT_T
            SET REORDER = 0
            WHERE QTY > P_MIN;
      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
```

```
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          6          5          0
```

```
UPDATE PRODUCT_T
SET P_MIN = P_MIN + 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          6          7          1
```

```
UPDATE PRODUCT_T
SET P_MIN = P_MIN - 1, 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          8          6          0

       ALTER TRIGGER CHECK_REORDER_STATUS_TRG DISABLE;

Trigger altered.



       CREATE OR REPLACE TRIGGER CHECK_REORDER_STATUS_TRG_RL
              BEFORE 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;
       /

Trigger created.


       ALTER TRIGGER CHECK_REORDER_STATUS_TRG_RL ENABLE;

Trigger altered.


       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';
```

```
1 row updated.
```

```
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';
```

```
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          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
```

```
ALTER TRIGGER CHECK_REORDER_STATUS_TRG_RL DISABLE;
```

```
Trigger altered.
```

Conclusion:

In this lab, we learned about oracle triggers with their syntaxes. We got to know the working of triggers and we also discussed a few examples to broaden our knowledge on different types of triggers that can be used/are used in the industry to solve business problems. TRIGGERS are stored programmes that are automatically run by the Oracle engine when DML statements such as insert, update, and delete are executed on the table or when certain events occur. The code that will be executed in the event of a trigger can be defined as needed. You can specify the event on which the trigger should be triggered as well as the timing of the execution. The goal of a trigger is to keep the information in the database as accurate as possible.

Viva Questions:

1. **Differentiate between a statement level trigger and row level trigger.**

   Row level triggers are executed once for each and every row in the transaction, whereas statement level triggers are executed once for each and every transaction.

Row level triggers are used only for data auditing, whereas statement level triggers are used to enforce all extra security on the table's transactions.

The "FOR EACH ROW" phrase is present in the CREATE TRIGGER command of a row level trigger, but it is not in the CREATE TRIGGER command of a statement level trigger.

If 1500 rows are to be added into a table, the row level trigger will be triggered 1500 times, while The statement level trigger would only be executed once if 1500 rows were to be added into a table.

2. How many different types of triggers a table can have? List them all

Triggers are categorised into three types:

- Level Triggers
- Event Triggers
- Timing Triggers

which are further divided into different parts.

- Level Triggers

  Level triggers are classified into two types:

  - ROW LEVEL TRIGGERS: It fires for each record that was impacted by DML statements like as INSERT, UPDATE, DELETE, and so on. A FOR EACH ROW clause is always used in a triggering statement.
  - STATEMENT LEVEL TRIGGERS: It fires once for each executed statement.

- Event Triggers

  Event triggers are classified into three types:

  - DDL EVENT TRIGGER: It is triggered when a DDL statement is executed (CREATE, ALTER, DROP, TRUNCATE).
  - DML EVENT TRIGGER: It is triggered when a DML statement is executed (INSERT, UPDATE, DELETE).
  - DATABASE EVENT TRIGGER
  - It is triggered by the execution of any database action, such as LOGON, LOGOFF, SHUTDOWN, SERVERERROR, and so forth.

- Timing Triggers

  Timing triggers are classified into two types:

  - BEFORE TRIGGER: It fires before the DML statement is executed. Depending on the preceding condition block, the triggering statement may or may not be performed.
  - AFTER TRIGGER: It is triggered after the DML statement has been executed.
  - INSTEAD OF TRIGGER: Describe how to perform insert, update, and delete operations against views that are too complex to support these operations natively. They allow applications to use a view as the sole interface for all SQL operations (insert, delete, update and select).

3. What are cascading triggers?

When a statement in a trigger body causes another trigger to be fired, the triggers are said to be cascading triggers.

Maximum no of cascading triggers at a time = 32.

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

A commit inside a trigger would defeat the basic definition of an atomic transaction. Trigger logic is an extension of the original DML operation. Changes made within triggers should thus be committed or rolled back as part of the transaction in which they execute. For this reason, triggers are NOT allowed to execute COMMIT or ROLLBACK statements (apart from autonomous triggers).

In partitioned database environments procedures cannot be invoked from triggers or SQL UDFs. On symmetric multi-processor (SMP) machines, procedure calls from triggers are executed on a single processor. A procedure that is to be called from a trigger must not contain a COMMIT statement or a ROLLBACK statement that attempts to roll back the unit of work. The ROLLBACK TO SAVEPOINT statement is supported within the procedure however the specified savepoint must be in the procedure. A rollback of a CALL statement from within a trigger will not roll back any external actions effected by the procedures, such as writing to the file system. The procedure must not modify any federated table. This means that the procedure must not contain a searched UPDATE of a nickname, a searched DELETE from a nickname or an INSERT to a nickname. Result sets specified for the procedure will not be accessible from inline SQL PL statements. If a cursor defined as **WITH RETURN TO CLIENT** is opened during the execution of a compiled trigger, result sets from the cursor will be discarded.

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

Yes, it is possible to create a trigger that will fire when a row is read during a query. Such types of triggers are called row level triggers.