```
 --JOURNAL COMPILATION
/*
            AIM:To write and execute stored procedures and functions using Oracle 11g.

            PROBLEM STATEMENT:Using the relation schemata established in
                              Experiments - 01, 02, 04, and 07, create and execute the
                              mentioned stored functions and stored procedures.
*/

 ----------------------------------- QUERY-01 -----------------------------
 /*
Write a SQL code to compile and execute an anonymous block which declares a
cursor – FACULTY. The cursor buffers the records comprising – Employee ID,
 Employee Name (FNAME and LNAME combined) and Designation for the Designation
entered by the user. You may use either EMPLOYEE table or EMPP table for this
 cursor. Use this cursor to print the buffered records. Use %NOFOUND variable
 to enable cursor exit.
 */
 ---------------------------------------------------------------------------
SQL> DECLARE
  2    DESG EMPLOYEE.DESIGNATION%TYPE:= '&FACULTY_DESIGNATION';
  3    CURSOR FACULTY IS
  4    SELECT * FROM EMPLOYEE WHERE DESIGNATION LIKE '%'||DESG||'%';
  5    REC EMPLOYEE%ROWTYPE;
  6  BEGIN
  7    OPEN FACULTY;
  8    LOOP
  9    FETCH FACULTY INTO REC;
 10    EXIT WHEN FACULTY%NOTFOUND;
 11    DBMS_OUTPUT.PUT_LINE(REC.EID||' '||REC.FNAME||' '||REC.LNAME||' '||REC.DESIGNATION);
 12    END LOOP;
 13    CLOSE FACULTY;
 14  DBMS_OUTPUT.PUT_LINE('ALL CURSOR ROWS FETCHED ...');
 15  END;
 16  /
Enter value for faculty_designation: Professor
old   2:   DESG EMPLOYEE.DESIGNATION%TYPE:= '&FACULTY_DESIGNATION';
new   2:   DESG EMPLOYEE.DESIGNATION%TYPE:= 'Professor';
7101 Samantha Jones Professor
7102 Albert Greenfield Asst. Professor
7104 Martina Jacobson Asst. Professor
7105 Alexander Lloyd Professor
7106 William Smithfield Asst. Professor
7107 Eugene Sabatini Professor
ALL CURSOR ROWS FETCHED ...

PL/SQL procedure successfully completed.

 ----------------------------------- QUERY-02 -----------------------------
 /*
CURSOR FOR LOOP: Modify the cursor in Query-01 as FACULTY_CFL which uses the
cursor FOR loop to buffering and displaying the records (as mentioned) when
 employee designation is entered by the user.Use a variation of cursor FOR loop
 to include the ROWCOUNT variable to print serial number for the displayed records.
 */
 ---------------------------------------------------------------------------
SQL> DECLARE
  2    CNT NUMBER :=1;
  3    DESG EMPLOYEE.DESIGNATION%TYPE:= '&FACULTY_DESIGNATION';
  4  BEGIN
  5    FOR FACULTY_FL IN(SELECT * FROM EMPLOYEE WHERE DESIGNATION LIKE '%'||DESG||'%')LOOP
  6    DBMS_OUTPUT.PUT_LINE(FACULTY_FL.EID||' '||FACULTY_FL.FNAME||' '||FACULTY_FL.LNAME||'
'||FACULTY_FL.DESIGNATION);
  7    END LOOP;
  8    FOR FACULTY_FL IN(SELECT * FROM EMPLOYEE WHERE DESIGNATION LIKE '%'||DESG||'%')LOOP
  9    DBMS_OUTPUT.PUT_LINE(CNT||' ' ||' '||FACULTY_FL.EID||' '||FACULTY_FL.FNAME||'
```

```
'||FACULTY_FL.LNAME||' '||FACULTY_FL.DESIGNATION);
 10    CNT:=CNT+1;
 11    END LOOP;
 12  DBMS_OUTPUT.PUT_LINE('CURSOR FOR LOOP EXITED ...');
 13  END;
 14  /
Enter value for faculty_designation: Lecturer
old   3:   DESG EMPLOYEE.DESIGNATION%TYPE:= '&FACULTY_DESIGNATION';
new   3:   DESG EMPLOYEE.DESIGNATION%TYPE:= 'Lecturer';
7103 Julia Martin Lecturer
7108 James Washington Sr. Lecturer
7109 Larry Gomes Lecturer
7110 Svetlana Sanders Lecturer
1  7103 Julia Martin Lecturer
2  7108 James Washington Sr. Lecturer
3  7109 Larry Gomes Lecturer
4  7110 Svetlana Sanders Lecturer
CURSOR FOR LOOP EXITED ...

PL/SQL procedure successfully completed.

 ---------------------------------- QUERY-03 ------------------------------
 /*
EXITING A CURSOR AFTER FETCHING SPECIFIED NUMBER OF ROWS: Modify the cursor
 FACULTY_CFL_A to display only those many records as desired by the user.
 Use %ROWCOUNT to enable the cursor to ensure this.
 */
 ------------------------------------------------------------------------
SQL> DECLARE
  2    DESG EMPLOYEE.DESIGNATION%TYPE:= '&FACULTY_DESIGNATION';
  3    NROWS NUMBER :='&HOW_MANY_ROWS';
  4    CURSOR FACULTY_CFL_A IS
  5    SELECT * FROM EMPLOYEE WHERE DESIGNATION LIKE '%'||DESG||'%';
  6    REC EMPLOYEE%ROWTYPE;
  7  BEGIN
  8    OPEN FACULTY_CFL_A;
  9    LOOP
 10    FETCH FACULTY_CFL_A INTO REC;
 11    EXIT WHEN FACULTY_CFL_A%ROWCOUNT>NROWS;
 12    DBMS_OUTPUT.PUT_LINE(REC.EID||' '||REC.FNAME||' '||REC.LNAME||' '||REC.DESIGNATION);
 13    END LOOP;
 14    CLOSE FACULTY_CFL_A;
 15  END;
 16  /
Enter value for faculty_designation: Professor
old   2:   DESG EMPLOYEE.DESIGNATION%TYPE:= '&FACULTY_DESIGNATION';
new   2:   DESG EMPLOYEE.DESIGNATION%TYPE:= 'Professor';
Enter value for how_many_rows: 3
old   3:   NROWS NUMBER :='&HOW_MANY_ROWS';
new   3:   NROWS NUMBER :='3';
7101 Samantha Jones Professor
7102 Albert Greenfield Asst. Professor
7104 Martina Jacobson Asst. Professor

PL/SQL procedure successfully completed.

 ---------------------------------- QUERY-04 ------------------------------
 /*
PARAMETERIZED CUSRSOR WITH DEFAULT VALUES: Write a SQL code to compile and
execute an anonymous block which declares a cursor – EMP_SAL_INFO
(Salary, Designation). The cursor buffers the records comprising – Employee ID,
 Employee Name (FNAME and LNAME combined), Designation and Salary for the Salary
and Designation entered by the user. Use EMPLOYEE table for this cursor.
Use this cursor to print the buffered records.
 */
 ------------------------------------------------------------------------
```

```
SQL> DECLARE
  2  EREC EMPLOYEE%ROWTYPE;
  3  CURSOR EMP_SAL_INFO(SAL EMPLOYEE.SALARY%TYPE:='&SPECIFIED_SALARY',DESG
EMPLOYEE.DESIGNATION%TYPE:='&SPECIFIED_DESIGNATION') IS
  4    SELECT * FROM EMPLOYEE WHERE SALARY>SAL AND UPPER(DESIGNATION)LIKE '%'||DESG||'%';
  5  BEGIN
  6  OPEN EMP_SAL_INFO;
  7       DBMS_OUTPUT.put_line('WITH DEFAULT VALUES...');
  8       LOOP
  9         FETCH EMP_SAL_INFO INTO  EREC;
 10         EXIT WHEN EMP_SAL_INFO%NOTFOUND;
 11         DBMS_OUTPUT.PUT_LINE(EREC.EID||'    '||EREC.FNAME||'  '||EREC.LNAME||'
'||EREC.DESIGNATION||'    '||EREC.SALARY);
 12       END LOOP;
 13  CLOSE EMP_SAL_INFO;
 14  OPEN EMP_SAL_INFO(17000);
 15       DBMS_OUTPUT.put_line('WITH SOME DEFAULT VALUES...');
 16       LOOP
 17         FETCH EMP_SAL_INFO INTO  EREC;
 18         EXIT WHEN EMP_SAL_INFO%NOTFOUND;
 19         DBMS_OUTPUT.PUT_LINE(EREC.EID||'    '||EREC.FNAME||'  '||EREC.LNAME||'
'||EREC.DESIGNATION||'    '||EREC.SALARY);
 20       END LOOP;
 21  CLOSE EMP_SAL_INFO;
 22  OPEN EMP_SAL_INFO(13000,'LECTURER');
 23       DBMS_OUTPUT.put_line('WITH ALL SUPPLIED VALUES...');
 24       LOOP
 25         FETCH EMP_SAL_INFO INTO  EREC;
 26         EXIT WHEN EMP_SAL_INFO%NOTFOUND;
 27         DBMS_OUTPUT.PUT_LINE(EREC.EID||'    '||EREC.FNAME||' '||EREC.LNAME||'
'||EREC.DESIGNATION||'    '||EREC.SALARY);
 28       END LOOP;
 29  CLOSE EMP_SAL_INFO;
 30  END;
 31  /
Enter value for specified_salary: 15000
Enter value for specified_designation: PROFESSOR
old   3: CURSOR EMP_SAL_INFO(SAL EMPLOYEE.SALARY%TYPE:='&SPECIFIED_SALARY',DESG
EMPLOYEE.DESIGNATION%TYPE:='&SPECIFIED_DESIGNATION') IS
new   3: CURSOR EMP_SAL_INFO(SAL EMPLOYEE.SALARY%TYPE:='15000',DESG
EMPLOYEE.DESIGNATION%TYPE:='PROFESSOR') IS
WITH DEFAULT VALUES...
7101    Samantha  Jones    Professor    16500
7104    Martina  Jacobson    Asst. Professor    15550
7105    Alexander  Lloyd    Professor    17500
7106    William  Smithfield    Asst. Professor    15660
7107    Eugene  Sabatini    Professor    16500
WITH SOME DEFAULT VALUES...
7105    Alexander  Lloyd    Professor    17500
WITH ALL SUPPLIED VALUES...
7103    Julia Martin    Lecturer    13320
7108    James Washington    Sr. Lecturer    14000
7109    Larry Gomes    Lecturer    13650

PL/SQL procedure successfully completed.


 ---------------------------------- QUERY-05 ------------------------------
 /*
BULK COLLECT with CURSORS: Write SQL code to compile and execute a procedure
– PRINT_EMPLOYEE which receives employee salary as input and prints the
following particulars – employee number, employee name and salary, for
employees whose salary exceeds the inputted salary.
 */
 ------------------------------------------------------------------------
SQL> CREATE OR REPLACE PROCEDURE PRINT_EMPLOYEE(ESAL IN EMPLOYEE.SALARY%TYPE)IS
```

```
  2    CURSOR SAL_CURSOR IS
  3      SELECT * FROM EMPLOYEE WHERE SALARY>ESAL;
  4    TYPE EMP_AAT IS TABLE OF EMPLOYEE%ROWTYPE;
  5      L_EMP_AAT EMP_AAT;
  6  BEGIN
  7      DBMS_OUTPUT.PUT_LINE('EMPLOYEES HAVING SALARY > 14000
  8  EID   EMPLOYEE NAME         SALARY
  9  ----  -------------------- ----------');
 10      OPEN SAL_CURSOR;
 11      LOOP
 12      FETCH SAL_CURSOR BULK COLLECT INTO L_EMP_AAT;
 13        FOR INDX IN 1..SAL_CURSOR%ROWCOUNT
 14        LOOP
 15        DBMS_OUTPUT.PUT_LINE(L_EMP_AAT(INDX).EID||'   '||RPAD(L_EMP_AAT(INDX).FNAME||'
'||L_EMP_AAT(INDX).LNAME,20,' ')||'   '||L_EMP_AAT(INDX).SALARY);
 16        END LOOP;
 17        EXIT WHEN SAL_CURSOR%NOTFOUND;
 18      END LOOP;
 19      CLOSE SAL_CURSOR;
 20      DBMS_OUTPUT.PUT_LINE('
 21  ----  -------------------- ----------');
 22      DBMS_OUTPUT.PUT_LINE('END OF BULK FETCH^.....');
 23  END;
 24  /

Procedure created.

SQL> CALL PRINT_EMPLOYEE(14000);
EMPLOYEES HAVING SALARY > 14000
EID   EMPLOYEE NAME         SALARY
----
-------------------- ----------
7101   Samantha Jones         16500
7102   Albert Greenfield      14200
7104   Martina Jacobson       15550
7105   Alexander Lloyd        17500
7106   William Smithfield     15660
7107   Eugene Sabatini        16500

----  -------------------- ----------
END OF BULK FETCH^.....

Call completed.

 ----------------------------------- QUERY-06 -----------------------------
 /*
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.
 */
 ------------------------------------------------------------------------------
SQL> CREATE OR REPLACE TRIGGER UPDATE_CUST_BALANCE_TRG
  2  AFTER INSERT ON LINE
  3  FOR EACH ROW
  4  BEGIN
  5    UPDATE CUSTOMER SET BALANCE=:NEW.L_PRICE*10 WHERE C_CODE=(
  6    SELECT C_CODE FROM INVOICE WHERE INV_NUM=:NEW.INV_NUM);
  7  END;
  8  /

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

4 rows selected.

SQL> SELECT * FROM INVOICE WHERE INV_NUM = 1006;

   INV_NUM     C_CODE INV_DATE
---------- ---------- ---------
      1006      10014 17-JAN-12

1 row selected.

SQL>
SQL> SELECT * FROM CUSTOMER WHERE C_CODE = 10014;

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

1 row selected.

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

1 row created.

SQL>
SQL> SELECT * FROM CUSTOMER WHERE C_CODE = 10014;

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

1 row selected.

 ------------------------------------ QUERY-07 -------------------------------
 /*
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, Systola Systematica, SYSDATE, 12000
Add : 7122, Boom Boom Bloom, SYSDATE, 15000
Modify : SALARY = SALARY + 2500 for EID >= 7121
Remove : EID = 7122
 */
 ------------------------------------------------------------------------------
SQL> CREATE TABLE SALARY_CHANGES(
  2  OP_TYPE VARCHAR2(10) NOT NULL,
  3  OP_DATE DATE DEFAULT SYSDATE,
  4  OP_TIME CHAR(9) DEFAULT TO_CHAR(SYSTIMESTAMP,'HH:MI:SS'),
  5  OLD_SAL NUMBER(7,2),
  6  NEW_SAL NUMBER(7,2),
  7  EID NUMBER(4) NOT NULL
  8  );

Table created.

SQL> CREATE OR REPLACE TRIGGER SALARY_CHANGE_TRG
  2  AFTER INSERT OR UPDATE OR DELETE ON EMPP
  3  FOR EACH ROW
```

```
  4  BEGIN
  5    IF INSERTING THEN
  6        INSERT INTO SALARY_CHANGES
VALUES('INSERT',SYSDATE,TO_CHAR(SYSDATE,'HH:MI:SS'),:OLD.SALARY,:NEW.SALARY,:NEW.EID);
  7        DBMS_OUTPUT.PUT_LINE('THE INSERT ENTRY IS LOGGED IN SALARY_CHANGES TABLE');
  8    END IF;
  9    IF DELETING THEN
 10        INSERT INTO SALARY_CHANGES
VALUES('DELETE',SYSDATE,TO_CHAR(SYSDATE,'HH:MI:SS'),:OLD.SALARY,:NEW.SALARY,:OLD.EID);
 11        DBMS_OUTPUT.PUT_LINE('THE DELETE ENTRY IS LOGGED IN SALARY_CHANGES TABLE');
 12    END IF;
 13    IF UPDATING THEN
 14      INSERT INTO SALARY_CHANGES
VALUES('UPDATE',SYSDATE,TO_CHAR(SYSDATE,'HH:MI:SS'),:OLD.SALARY,:NEW.SALARY,:OLD.EID);
 15      DBMS_OUTPUT.PUT_LINE('THE UPDATE ENTRY IS LOGGED IN SALARY_CHANGES TABLE');
 16    END IF;
 17  END;
 18  /

Trigger created.

SQL> SELECT COUNT(*) FROM EMPP;

  COUNT(*)
----------
        11

1 row selected.

SQL> INSERT INTO EMPP VALUES (7121, 'Systola Systematica', SYSDATE, 12000);

1 row created.

SQL> INSERT INTO EMPP VALUES (7122, 'Boom Boom Bloom', SYSDATE, 15000);

1 row created.

SQL> UPDATE EMPP SET SALARY = SALARY + 2500 WHERE EID >= 7121;

2 rows updated.

SQL> DELETE FROM EMPP WHERE EID = 7122;

1 row deleted.

SQL> SELECT COUNT(*) FROM EMPP;

  COUNT(*)
----------
        12

1 row selected.

SQL> SELECT * FROM SALARY_CHANGES;

OP_TYPE    OP_DATE   OP_TIME      OLD_SAL    NEW_SAL        EID
---------- --------- --------- ---------- ---------- ----------
INSERT     08-APR-17 04:13:35       12000      12000       7121
INSERT     08-APR-17 04:13:35       15000      15000       7122
UPDATE     08-APR-17 04:13:35       12000      14500       7121
UPDATE     08-APR-17 04:13:35       15000      17500       7122
DELETE     08-APR-17 04:13:35       17500                  7122

5 rows selected.

 ----------------------------------- QUERY-08 ------------------------------
```

```
 /*
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'
 */
 --------------------------------------------------------------------------------
SQL> CREATE TABLE EMP_SALARY AS SELECT EID,SALARY AS TOT_SAL FROM EMPP WHERE 2=0;

Table created.

SQL> ALTER TABLE EMP_SALARY  ADD CONSTRAINT EMP_SAL_PK PRIMARY KEY(EID);

Table altered.

SQL> ALTER TABLE EMP_SALARY ADD STATUS VARCHAR2(7) DEFAULT 'ON_ROLL' NOT NULL;

Table altered.

SQL> INSERT INTO EMP_SALARY SELECT EID,(SALARY * 1.35 - 1200)* 0.90,'ON_ROLL' FROM EMPP;

11 rows created.

SQL> CREATE OR REPLACE TRIGGER UPDATE_TOT_SAL_TRG
  2  AFTER INSERT OR UPDATE OR DELETE ON EMPP
  3  FOR EACH ROW
  4  BEGIN
  5    IF INSERTING THEN
  6    INSERT INTO EMP_SALARY VALUES(:NEW.EID,(:NEW.SALARY*1.35-1200)*0.90,'ON_ROLL');
  7    END IF;
  8    IF UPDATING THEN
  9    UPDATE EMP_SALARY SET TOT_SAL=(:NEW.SALARY*1.35-1200)*0.90 WHERE EID=:OLD.EID;
 10    END IF;
 11    IF DELETING THEN
 12    UPDATE EMP_SALARY SET STATUS='RETIRED' WHERE EID=:OLD.EID;
 13    END IF;
 14  END;
 15  /

Trigger created.

SQL> ALTER TRIGGER SALARY_CHANGE_TRG DISABLE;

Trigger altered.

SQL> SELECT COUNT(*) FROM EMPP;

  COUNT(*)
----------
        11

1 row selected.

SQL>
SQL> INSERT INTO EMPP VALUES (7120, 'Solomon Dataminer', SYSDATE, 14000);

1 row created.

SQL> INSERT INTO EMPP VALUES (7122, 'Boom Boom Bloom', SYSDATE, 15000);

1 row created.

SQL> UPDATE EMPP SET SALARY = SALARY + 2500 WHERE EID in (7120, 7122);
```

```
2 rows updated.

SQL> DELETE FROM EMPP WHERE EID = 7122;

1 row deleted.

SQL> SELECT COUNT(*) FROM EMPP;

  COUNT(*)
----------
        12

1 row selected.

SQL> SELECT * FROM EMP_SALARY;

       EID    TOT_SAL STATUS
---------- ---------- -------
      7101    18967.5 ON_ROLL
      7102      16173 ON_ROLL
      7103    15103.8 ON_ROLL
      7104   17813.25 ON_ROLL
      7105    20182.5 ON_ROLL
      7106    17946.9 ON_ROLL
      7107    18967.5 ON_ROLL
      7108      15930 ON_ROLL
      7109   15504.75 ON_ROLL
      7110      11070 ON_ROLL
      7121    16537.5 ON_ROLL

       EID    TOT_SAL STATUS
---------- ---------- -------
      7120    18967.5 ON_ROLL
      7122    20182.5 RETIRED

13 rows selected.

 ----------------------------------- QUERY-09------------------------------
 /*
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.
 */
 -------------------------------------------------------------------------------
SQL> CREATE OR REPLACE TRIGGER LINE_INS_UPD_QTY_TRG
  2  AFTER INSERT ON LINE
  3  FOR EACH ROW
  4  BEGIN
  5    UPDATE PRODUCT SET QTY=QTY-:NEW.L_UNITS WHERE P_CODE=:NEW.P_CODE;
  6  END;
  7  /

Trigger created.

SQL> SELECT P_CODE, DESCRIPT, QTY FROM PRODUCT WHERE P_CODE = 'SM48X';

P_COD DESCRIPT                              QTY
----- ----------------------------- ----------
SM48X Steel Malting Mesh                    18

1 row selected.

SQL> SELECT INV_NUM, L_NUM, P_CODE, L_UNITS FROM LINE WHERE INV_NUM = 1009;

   INV_NUM      L_NUM P_COD    L_UNITS
```

```
---------- ---------- ----- ----------
      1009          1 HH15X         20

1 row selected.

SQL> INSERT INTO LINE VALUES(1009, 2,'SM48X', 10, 119.95);

1 row created.

SQL> SELECT INV_NUM, L_NUM, P_CODE, L_UNITS FROM LINE WHERE INV_NUM = 1009;

   INV_NUM      L_NUM P_COD    L_UNITS
---------- ---------- ----- ----------
      1009          1 HH15X         20
      1009          2 SM48X         10

2 rows selected.

SQL> SELECT P_CODE, DESCRIPT, QTY FROM PRODUCT WHERE P_CODE = 'SM48X';

P_COD DESCRIPT                              QTY
----- ------------------------------ ----------
SM48X Steel Malting Mesh                      8

1 row selected.

 ----------------------------------- QUERY-10------------------------------
 /*
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).
 */
 ------------------------------------------------------------------------------
SQL> CREATE TABLE PRODUCT_T  AS SELECT P_CODE,DESCRIPT,QTY,P_MIN,P_PRICE,V_CODE FROM PRODUCT;

Table created.

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

Table altered.

SQL> CREATE OR REPLACE TRIGGER CHECK_REORDER_STATUS_TRG
  2  AFTER UPDATE OF QTY,P_MIN ON PRODUCT_T
  3  BEGIN
  4    UPDATE PRODUCT_T SET REORDER=1 WHERE QTY<=P_MIN ;
  5    UPDATE PRODUCT_T SET REORDER=0 WHERE QTY>P_MIN ;
  6  END;
  7  /

Trigger created.

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

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

1 row selected.

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

```
1 row updated.

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

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

1 row selected.

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

1 row updated.

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

P_COD        QTY      P_MIN    REORDER
----- ---------- ---------- ----------
SH100          6          5          0

1 row selected.

SQL> UPDATE PRODUCT_T SET P_MIN = P_MIN + 3 WHERE P_CODE = 'SH100';

1 row updated.

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

P_COD        QTY      P_MIN    REORDER
----- ---------- ---------- ----------
SH100          6          8          1

1 row selected.

SQL> UPDATE PRODUCT_T SET P_MIN = P_MIN - 4 WHERE P_CODE = 'SH100';

1 row updated.

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

P_COD        QTY      P_MIN    REORDER
----- ---------- ---------- ----------
SH100          6          4          0

1 row selected.

SQL> CREATE OR REPLACE TRIGGER CHECK_REORDER_STATUS_TRG_RL
  2  BEFORE  UPDATE ON PRODUCT_T
  3  FOR EACH ROW
  4  BEGIN
  5       IF :NEW.QTY<=:NEW.P_MIN THEN
  6       :NEW.REORDER:=1;
  7       END IF;
  8       IF :NEW.QTY>:NEW.P_MIN THEN
  9       :NEW.REORDER:=0;
 10       END IF;
 11  END;
 12  /

Trigger created.

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

P_COD        QTY      P_MIN    REORDER
----- ---------- ---------- ----------
SH100          6          4          0
```

```
1 row selected.

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

1 row updated.

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

P_COD        QTY      P_MIN    REORDER
----- ---------- ---------- ----------
SH100          3          4          1

1 row selected.

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

1 row updated.

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

P_COD        QTY      P_MIN    REORDER
----- ---------- ---------- ----------
SH100          4          4          1

1 row selected.

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

1 row updated.

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

P_COD        QTY      P_MIN    REORDER
----- ---------- ---------- ----------
SH100          5          4          0

1 row selected.

SQL> UPDATE PRODUCT_T SET P_MIN = P_MIN + 3 WHERE P_CODE = 'SH100';

1 row updated.

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

P_COD        QTY      P_MIN    REORDER
----- ---------- ---------- ----------
SH100          5          7          1

1 row selected.

SQL> UPDATE PRODUCT_T SET P_MIN = P_MIN - 4 WHERE P_CODE = 'SH100';

1 row updated.

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

P_COD        QTY      P_MIN    REORDER
----- ---------- ---------- ----------
SH100          5          3          0

1 row selected.

 ---------------------------------- END OF QUERIES--------------------------

SQL> SET FEEDBACK OFF
```

```
SQL> SET ECHO OFF
SQL> SPOOL OFF
```