***** EXPERIMENT NO: 05 *****

Aim : To write and execute stored procedures and stored functions using Oracle 11g.

Problem Statement : Using the relation schemata established in Experiments - 02, 03, and 05, create and execute the mentioned stored functions and stored procedures.

Author : Bhavesh Kewalramani

Roll No : 025 [5A]

Date : 21-September-2021

****

QUERY-01: Write SQL code to compile and execute a stored procedure - SHOW_EMPLOYEE, to list employee details for the input variable ENO holding employee number. (Use EMPP Table)

****

```
CREATE OR REPLACE PROCEDURE SHOW_EMPLOYEE (
      ID IN EMPP.EID%TYPE,
      NAME OUT EMPP.ENAME%TYPE,
      HDT OUT EMPP.HIREDATE%TYPE,
      DES OUT EMPP.DESIGNATION%TYPE,
      SAL OUT EMPP.SALARY%TYPE)
AS
BEGIN
      SELECT ENAME,HIREDATE,DESIGNATION,SALARY
      INTO NAME,HDT,DES,SAL
      FROM EMPP
      WHERE EID = ID;
END;
/
```

```
Procedure created.
```

```
DECLARE
      ID EMPP.EID%TYPE := &EMPLOYEE_NUMBER;
      NAME EMPP.ENAME%TYPE;
      HDT EMPP.HIREDATE%TYPE;
      DES EMPP.DESIGNATION%TYPE;
      SAL EMPP.SALARY%TYPE;
BEGIN
      SHOW_EMPLOYEE(ID,NAME,HDT,DES,SAL);
      DBMS_OUTPUT.PUT_LINE('EMPLOYEE ID          := '||ID);
      DBMS_OUTPUT.PUT_LINE('EMPLOYEE NAME        := '||NAME);
      DBMS_OUTPUT.PUT_LINE('EMPLOYEE HIREDATE    := '||HDT);
      DBMS_OUTPUT.PUT_LINE('EMPLOYEE DESIGNATION := '||DES);
      DBMS_OUTPUT.PUT_LINE('EMPLOYEE SALARY      := '||SAL);
EXCEPTION
      WHEN NO_DATA_FOUND THEN
            DBMS_OUTPUT.PUT_LINE('EMPLOYEE DETAILS DOES NOT
                                       EXIST FOR EMPLOYEE NUMBER '
                                       ||ID||' .....');
      WHEN OTHERS THEN
            DBMS_OUTPUT.PUT_LINE('SOME ERROR OCCURRED ..... ');
END;
/
```

```
Enter value for employee_number: 7109
EMPLOYEE ID          := 7109
EMPLOYEE NAME        := Martina Jacobson
EMPLOYEE HIREDATE    := 15-NOV-09
EMPLOYEE DESIGNATION := Asst. Professor
EMPLOYEE SALARY      := 91000

PL/SQL procedure successfully completed.
```

```
      /
```

```
Enter value for employee_number: 7111
EMPLOYEE ID          := 7111
EMPLOYEE NAME        := Albert Greenfield
EMPLOYEE HIREDATE    := 12-JUL-16
EMPLOYEE DESIGNATION := Research Asst.
EMPLOYEE SALARY      := 48200

PL/SQL procedure successfully completed.


Enter value for employee_number: 7211
EMPLOYEE DETAILS DOES NOT EXIST FOR EMPLOYEE NUMBER 7211 .....
```

```
PL/SQL procedure successfully completed.
```

****

QUERY-02: Write SQL code to compile and execute a stored procedure - ADD_EMPLOYEE, to add a record to EMPP table. Check the existence of the created procedure using USER_OBJECTS view. Use this procedure to insert following records.
7118,**Your Name**,07-Jul-2020,Teaching Asst.,25000
7119,Atulya Bharat,03-Aug-2005,Professor,162000
****

```
      CREATE OR REPLACE PROCEDURE ADD_EMPLOYEE (
            ID IN EMPP.EID%TYPE,
            NAME IN EMPP.ENAME%TYPE,
            HDT IN EMPP.HIREDATE%TYPE,
            DES IN EMPP.DESIGNATION%TYPE,
            SAL IN EMPP.SALARY%TYPE)
      AS
      BEGIN
            INSERT INTO EMPP VALUES (ID, NAME, HDT, DES, SAL);
      END;
      /

Procedure created.
```

```
      DECLARE
            ID EMPP.EID%TYPE := &EMPLOYEE_NUMBER;
            NAME EMPP.ENAME%TYPE := '&EMPLOYEE_NAME';
            HDT EMPP.HIREDATE%TYPE := '&EMPLOYEE_HIREDATE';
            DES EMPP.DESIGNATION%TYPE := '&EMPLOYEE_DESIGNATION';
            SAL EMPP.SALARY%TYPE := &EMPLOYEE_SALARY;
      BEGIN
            ADD_EMPLOYEE(ID, NAME, HDT, DES, SAL);
            DBMS_OUTPUT.PUT_LINE('EMPLOYEE ADDED SUCCESSFULLY ...');
      END;
      /

Enter value for employee_number: 7118
Enter value for employee_name: Bhavesh Kewalramani
Enter value for employee_hiredate: 07-JUL-2020
Enter value for employee_designation: Teaching Asst.
Enter value for employee_salary: 25000
EMPLOYEE ADDED SUCCESSFULLY ...

PL/SQL procedure successfully completed.


      /

Enter value for employee_number: 7119
Enter value for employee_name: Atulya Bharat
Enter value for employee_hiredate: 03-AUG-2005
Enter value for employee_designation: Professor
Enter value for employee_salary: 162000
EMPLOYEE ADDED SUCCESSFULLY ...

PL/SQL procedure successfully completed.
```

```
************************************************************************************
```

QUERY-03: Write SQL code to compile and execute the stored procedure REMOVE_EMPLOYEE, which will remove the employee record(s) from EMPP table when supplied with an input name phrase (entered always as lower case) indicating employee name (use EMPP table). If the matching employee is not found, an appropriate exception should be raised.

```
/*
      Create a table ITEMS that includes P_CODE, DESCRIPT as DESCR, P_DATE
      as IN_DATE, P_MIN as MIN_QTY, QTY, P_FRICE as PRICE and V_CODE from
      the PRODUCT table. ITEMS table must be populated at creation.

      Modify ITEMS table to add P_CODE as primary key. Also configure the
      columns IN_DATE and MIN_QTY to assume default values as SYSDATE and 2
      respectively.
*/
************************************************************************************

      CREATE OR REPLACE PROCEDURE REMOVE_EMPLOYEE(
            NAME IN EMPP.ENAME%TYPE)
      AS
      BEGIN
            DELETE FROM EMPP WHERE LOWER(ENAME)=LOWER(NAME);
      EXCEPTION
            WHEN NO_DATA_FOUND THEN
```

```
                    DBMS_OUTPUT.PUT_LINE('NO DATA FOUND OF THE EMPLOYEE
                                                 HAVING NAME '||NAME||' ...');
             WHEN TOO_MANY_ROWS THEN
                    DBMS_OUTPUT.PUT_LINE('DUPLICATE NAMES EXIST ... ');
             WHEN OTHERS THEN
                    DBMS_OUTPUT.PUT_LINE('SOME ERROR OCCURRED ... ');
      END;
      /

Procedure created.


      DECLARE
          NAME EMPP.ENAME%TYPE := LOWER('&EMPLOYEE_NAME');
      BEGIN
          REMOVE_EMPLOYEE(NAME);
      END;
      /

Enter value for employee_name: simon downing

PL/SQL procedure successfully completed.


      CREATE TABLE ITEMS
      AS (
             SELECT P_CODE,DESCRIPT AS DESCR,
                    P_DATE AS IN_DATE, P_MIN AS MIN_QTY,
                    QTY, P_PRICE AS PRICE, V_CODE
             FROM PRODUCT
      );

Table created.


      ALTER TABLE ITEMS
      ADD
      CONSTRAINT ITEMS_PK_P_CODE PRIMARY KEY (P_CODE);

Table altered.
```

```
      ALTER TABLE ITEMS
      MODIFY IN_DATE DEFAULT SYSDATE;

Table altered.


      ALTER TABLE ITEMS
      MODIFY MIN_QTY DEFAULT 2;

Table altered.
```

****

QUERY-04: Write SQL code to compile and execute the stored function - CHECK_ITEM that will report status as **1** if items with mentioned P_CODE are present in the inventory, otherwise reports status as **0**. No exceptions to be handled.

****

```
      CREATE OR REPLACE FUNCTION CHECK_ITEM (
            CODE ITEMS.P_CODE%TYPE)
                  RETURN NUMBER IS
                              KNT NUMBER;
      BEGIN
            SELECT COUNT(*) INTO KNT
            FROM ITEMS
            WHERE UPPER(P_CODE)=UPPER(CODE);
            IF KNT>=1 THEN
                  KNT := 1;
                  RETURN KNT;
            ELSE
                  KNT := 0;
                  RETURN KNT;
            END IF;
      END;
      /

Function created.

      DECLARE
          CODE ITEMS.P_CODE%TYPE := '&PRODUCT_CODE';
          KNT NUMBER;
      BEGIN
```

```
          KNT := CHECK_ITEM(CODE);
          IF KNT = 1 THEN
                  DBMS_OUTPUT.PUT_LINE('PRODUCT IS PRESENT ... ');
          ELSE
                  DBMS_OUTPUT.PUT_LINE('PRODUCT IS NOT PRESENT ... ');
          END IF;
      END;
      /
```

```
Enter value for product_code: WC025
PRODUCT IS PRESENT ...

PL/SQL procedure successfully completed.
```

```
      /
```

```
Enter value for product_code: HW111
PRODUCT IS NOT PRESENT ...

PL/SQL procedure successfully completed.
```

****

QUERY-05: Write a SQL code to compile and execute the stored procedure - ADD_ITEM, that will insert an item in ITEMS table with given particulars - item code, item description, invoice date, quantity of purchase, minimum quantity, item price and supplier code.

****

```
      CREATE OR REPLACE PROCEDURE ADD_ITEM (
            CODE IN ITEMS.P_CODE%TYPE,
            DESCRIP IN ITEMS.DESCR%TYPE,
            IDT IN ITEMS.IN_DATE%TYPE,
            MQTY IN ITEMS.MIN_QTY%TYPE,
            QT IN ITEMS.QTY%TYPE,
            PR IN ITEMS.PRICE%TYPE,
            VCODE IN ITEMS.V_CODE%TYPE)
      AS
      BEGIN
            INSERT INTO ITEMS VALUES (CODE, DESCRIP, IDT, MQTY, QT, PR, VCODE);
      END;
      /
```

```
Procedure created.

      DECLARE
            CODE ITEMS.P_CODE%TYPE := '&ITEM_CODE';
            DESCRIP ITEMS.DESCR%TYPE := '&ITEM_DESCRIPTION';
            IDT ITEMS.IN_DATE%TYPE := '&ITEM_IN_DATE';
            MQTY ITEMS.MIN_QTY%TYPE := &ITEM_MINIMUM_QUANTITY;
            QT ITEMS.QTY%TYPE := &ITEM_QUANTITY;
            PR ITEMS.PRICE%TYPE := &ITEM_PRICE;
            VCODE ITEMS.V_CODE%TYPE := &ITEM_VENDOR_CODE;
      BEGIN
            ADD_ITEM(CODE, DESCRIP, IDT, MQTY, QT, PR, VCODE);
            DBMS_OUTPUT.PUT_LINE('ITEM ADDED SUCCESSFULLY ...');
      END;
      /

Enter value for item_code: ES111
Enter value for item_description: Electric Saw
Enter value for item_in_date: 22-SEP-2021
Enter value for item_minimum_quantity: 2
Enter value for item_quantity: 5
Enter value for item_price: 500
Enter value for item_vendor_code: 25595
ITEM ADDED SUCCESSFULLY ...

PL/SQL procedure successfully completed.
```

****

QUERY-06: Write a SQL code to compile and execute the stored procedure - UPDATE_ITEM, that will update particulars (quantity and/ or cost) for an item in ITEMS table with given particulars - item code, quantity of purchase, and item price.

Report an error when the said item (to be updated) does not exist in ITEMS table (the NO_DATA_FOUND exception). Use the CHECK_ITEM function created earlier.

****

```
      CREATE OR REPLACE PROCEDURE UPDATE_ITEM (
          CODE IN ITEMS.P_CODE%TYPE,
          QT IN ITEMS.QTY%TYPE := NULL,
          PR IN ITEMS.PRICE%TYPE := NULL)
      AS
          KNT NUMBER;
```

```
       BEGIN
           KNT := CHECK_ITEM(CODE);
           IF KNT = 1 THEN
             IF QT IS NOT NULL THEN
                   UPDATE ITEMS SET QTY = QT WHERE P_CODE = CODE;
             END IF;
             IF PR IS NOT NULL THEN
                   UPDATE ITEMS SET PRICE = PR WHERE P_CODE = CODE;
             END IF;
             DBMS_OUTPUT.PUT_LINE('ITEM INFORMATION HAS BEEN UPDATE
                                                 SUCCESSFULLY ... ');
           ELSE
             RAISE NO_DATA_FOUND;
           END IF;
           EXCEPTION
             WHEN NO_DATA_FOUND THEN
                   DBMS_OUTPUT.PUT_LINE('ENTERED ITEM CODE DETAILS DOES
                                                        NOT EXIST ... ');
             WHEN OTHERS THEN
                   DBMS_OUTPUT.PUT_LINE('SOME ERROR OCCURRED ... ');
       END;
       /

Procedure created.

       DECLARE
           CODE ITEMS.P_CODE%TYPE;
           QT ITEMS.QTY%TYPE;
           PR ITEMS.PRICE%TYPE;
       BEGIN
           CODE := '&ITEM_CODE';
           QT := &ITEM_QUANTITY;
           PR := &ITEM_PRICE;
           UPDATE_ITEM(CODE,QT,PR);
       END;
       /

Enter value for item_code: PP101
Enter value for item_quantity: ''
Enter value for item_price: 100
ITEM INFORMATION HAS BEEN UPDATE SUCCESSFULLY ...
```

```
PL/SQL procedure successfully completed.


      /

Enter value for item_code: PP101
Enter value for item_quantity: 10
Enter value for item_price: ''
ITEM INFORMATION HAS BEEN UPDATE SUCCESSFULLY ...

PL/SQL procedure successfully completed.

      /

Enter value for item_code: PP101
Enter value for item_quantity: 20
Enter value for item_price: 110
ITEM INFORMATION HAS BEEN UPDATE SUCCESSFULLY ...

PL/SQL procedure successfully completed.

      /

Enter value for item_code: PP222
Enter value for item_quantity: 22
Enter value for item_price: 222
ENTERED ITEM CODE DETAILS DOES NOT EXIST ...

PL/SQL procedure successfully completed.
```

****

QUERY-07: Modify procedure in Query-06, as UPDATE_ITEM_ADD_WHEN_NOT_FOUND such that when the mentioned item is not present in ITEMS, an item is entered into ITEMS with available particulars supplied in the procedure call.

The default values for item description, vendor code and minimum quantity as 'NEW ITEM ...', NULL and (quantity / 8) truncated respectively. Use ADD_ITEM procedure created earlier.

You need not catch the NO_DATA_FOUND exception.

****

```
      CREATE OR REPLACE PROCEDURE UPDATE_ITEM_ADD_WHEN_NOT_FOUND (
        CODE IN ITEMS.P_CODE%TYPE,
        QT IN ITEMS.QTY%TYPE := NULL,
        PR IN ITEMS.PRICE%TYPE := NULL)
AS
        KNT NUMBER;
        DESCRIP ITEMS.DESCR%TYPE := 'NEW ITEM ...';
        IDT ITEMS.IN_DATE%TYPE   := SYSDATE;
        VCODE ITEMS.V_CODE%TYPE  := NULL;
        MQTY ITEMS.MIN_QTY%TYPE  := (QT/8);
BEGIN
        KNT := CHECK_ITEM(CODE);
        IF KNT = 1 THEN
                IF QT IS NOT NULL THEN
                        UPDATE ITEMS SET QTY = QT WHERE P_CODE = CODE;
                END IF;
                IF PR IS NOT NULL THEN
                        UPDATE ITEMS SET PRICE = PR WHERE P_CODE = CODE;
                END IF;
                DBMS_OUTPUT.PUT_LINE('ITEM INFORMATION HAS BEEN UPDATE
                                                        SUCCESSFULLY ... ');
        END IF;
        IF KNT = 0 THEN
                ADD_ITEM(CODE,DESCRIP,IDT,MQTY,QT,PR,VCODE);
                DBMS_OUTPUT.PUT_LINE('ITEM ADDED SUCCESSFULLY WITH
                                                 THE FOLLOWING DETAILS : ');
                DBMS_OUTPUT.PUT_LINE('ITEM CODE             : '||CODE);
                DBMS_OUTPUT.PUT_LINE('ITEM DESCRIPTION      : '||DESCRIP);
                DBMS_OUTPUT.PUT_LINE('ITEM IN DATE          : '||IDT);
                DBMS_OUTPUT.PUT_LINE('ITEM MINIMUM QUANTITY : '||MQTY);
                DBMS_OUTPUT.PUT_LINE('ITEM QUANTITY         : '||QT);
                DBMS_OUTPUT.PUT_LINE('ITEM PRICE            : '||PR);
                DBMS_OUTPUT.PUT_LINE('ITEM VENDOR CODE      : '||VCODE);
    END IF;
EXCEPTION
    WHEN OTHERS THEN
                DBMS_OUTPUT.PUT_LINE('SOME ERROR OCCURRED ... ');
END;
/
```

```
Procedure created.
```

```
      DECLARE
            CODE ITEMS.P_CODE%TYPE;
            QT ITEMS.QTY%TYPE;
            PR ITEMS.PRICE%TYPE;
      BEGIN
            CODE := '&ITEM_CODE';
            QT := &ITEM_QUANTITY;
            PR := &ITEM_PRICE;
            UPDATE_ITEM_ADD_WHEN_NOT_FOUND(CODE,QT,PR);
      END;
      /
```

```
Enter value for item_code: HP15P
Enter value for item_quantity: 22
Enter value for item_price: 222
ITEM ADDED SUCCESSFULLY WITH THE FOLLOWING DETAILS :
ITEM CODE             : HP15P
ITEM DESCRIPTION      : NEW ITEM ...
ITEM IN DATE          : 23-SEP-21
ITEM MINIMUM QUANTITY : 3
ITEM QUANTITY         : 22
ITEM PRICE            : 222
ITEM VENDOR CODE      :

PL/SQL procedure successfully completed.
```

****

QUERY-08: Write a SQL code to compile and execute the stored procedure - SHOW_ITEM that will list the item particulars for an item in ITEMS table when the item code is supplied as input.

Report an error when the said item to be updated does not exist in ITEMS. Use the CHECK_ITEM function created earlier.

```
/*
      ALTER TABLE ITEMS DROP PRIMARY KEY;

      EXEC ADD_ITEM{'HH15P','NEW ITEM-2',150,NULL,25);

*/
```

****

```
      CREATE OR REPLACE PROCEDURE SHOW_ITEM (
            CODE IN ITEMS.P_CODE%TYPE)
      AS
```

```
        DESCRIP ITEMS.DESCR%TYPE;
        IDT ITEMS.IN_DATE%TYPE;
        MQTY ITEMS.MIN_QTY%TYPE;
        QT ITEMS.QTY%TYPE;
        PR ITEMS.PRICE%TYPE;
        VCODE ITEMS.V_CODE%TYPE;
        KNT NUMBER;
  BEGIN
        KNT := CHECK_ITEM(CODE);
        IF KNT=1 THEN
              SELECT DESCR,IN_DATE,MIN_QTY,QTY,PRICE,V_CODE
              INTO DESCRIP,IDT,MQTY,QT,PR,VCODE
              FROM ITEMS
              WHERE P_CODE = CODE;
              DBMS_OUTPUT.PUT_LINE('ITEM CODE             := '||CODE);
              DBMS_OUTPUT.PUT_LINE('ITEM DESCRIPTION      := '||DESCRIP);
              DBMS_OUTPUT.PUT_LINE('ITEM IN DATE          := '||IDT);
              DBMS_OUTPUT.PUT_LINE('ITEM MINIMUM QUANTITY := '||MQTY);
              DBMS_OUTPUT.PUT_LINE('ITEM QUANTITY         := '||QT);
              DBMS_OUTPUT.PUT_LINE('ITEM PRICE            := '||PR);
              DBMS_OUTPUT.PUT_LINE('ITEM VENDOR CODE      := '||VCODE);
        ELSE
              RAISE NO_DATA_FOUND;
        END IF;
  EXCEPTION
        WHEN NO_DATA_FOUND THEN
              DBMS_OUTPUT.PUT_LINE('NO ITEM WITH ITEM CODE '||CODE||
                                                    ' FOUND ...');
        WHEN OTHERS THEN
              DBMS_OUTPUT.PUT_LINE('SOME ERROR OCCURRED ...');
  END;
  /

Procedure created.

  DECLARE
        CODE ITEMS.P_CODE%TYPE := '&ITEM_CODE';
  BEGIN
        SHOW_ITEM(CODE);
  END;
  /
```

```
Enter value for item_code: PP101
ITEM CODE              := PP101
ITEM DESCRIPTION       := PVC Pipe
ITEM IN DATE           := 20-FEB-20
ITEM MINIMUM QUANTITY := 75
ITEM QUANTITY          := 20
ITEM PRICE             := 110
ITEM VENDOR CODE       :=

PL/SQL procedure successfully completed.
```

```
      /
```

```
Enter value for item_code: PP223
NO ITEM WITH ITEM CODE PP223 FOUND ...

PL/SQL procedure successfully completed.
```

****

QUERY-09: Modify the procedure in Query-08 as SHOW_ITEM_TMR_E which will handle TOO_MANY_ROWS exception in SELECT query.
In addition to exceptions in Query-06 (NO_DATA_FOUND and OTHERS) the TOO_MANY_ROWS exception should be caught when a call to the procedure call - EXEC ADD_ITEM('HH15P', 'NEW ITEM-2', 150,NULL,25); fetches more than one row in the result set.

****

```
CREATE OR REPLACE PROCEDURE SHOW_ITEM_TMR_E(
      CODE IN ITEMS.P_CODE%TYPE)
AS
      DESCRIP ITEMS.DESCR%TYPE;
      IDT ITEMS.IN_DATE%TYPE;
      MQTY ITEMS.MIN_QTY%TYPE;
      QT ITEMS.QTY%TYPE;
      PR ITEMS.PRICE%TYPE;
      VCODE ITEMS.V_CODE%TYPE;
      KNT NUMBER;
      KNT1 NUMBER;
BEGIN
      KNT := CHECK_ITEM(CODE);
      SELECT COUNT(*) INTO KNT1
```

```
            FROM ITEMS
            WHERE P_CODE=CODE;
            IF KNT=1 AND KNT1 = 1 THEN
                  SELECT DESCR,IN_DATE,MIN_QTY,QTY,PRICE,V_CODE
                  INTO DESCRIP,IDT,MQTY,QT,PR,VCODE
                  FROM ITEMS
                  WHERE P_CODE = CODE;
                  DBMS_OUTPUT.PUT_LINE(RPAD(CODE,5)||' '||RPAD(DESCRIP,30)
                                     ||' '||RPAD(IDT,11)||' '||RPAD(MQTY,3)
                                     ||' '||RPAD(QT,4)||' '||RPAD(PR,6)
                                     ||' '||RPAD(VCODE,6));
            ELSIF KNT1 > 1 THEN
                  RAISE TOO_MANY_ROWS;
            ELSE
                  RAISE NO_DATA_FOUND;
            END IF;
      EXCEPTION
            WHEN NO_DATA_FOUND THEN
                  DBMS_OUTPUT.PUT_LINE('NO ITEM WITH ITEM CODE '||CODE||
                                                         ' FOUND ...');
            WHEN TOO_MANY_ROWS THEN
                  DBMS_OUTPUT.PUT_LINE(CODE||' MULTIPLE ROWS ...........
                                   ....................................');
            WHEN OTHERS THEN
                  DBMS_OUTPUT.PUT_LINE('SOME ERROR OCCURRED ...');
      END;
      /

Procedure created.

      BEGIN
            SHOW_ITEM_TMR_E('HH15P');
            SHOW_ITEM_TMR_E('HH15X');
      END;
      /

HH15P MULTIPLE ROWS ..............................................
HH15X Hanging Hook 15in             10-JAN-13   25  200  5.75   24992

PL/SQL procedure successfully completed.
```

```
CREATE OR REPLACE PROCEDURE UPDATE_ITEM_TMR_E (
       CODE IN ITEMS.P_CODE%TYPE,
       QT IN ITEMS.QTY%TYPE := NULL,
       PR IN ITEMS.PRICE%TYPE := NULL)
AS
       KNT NUMBER;
       KNT1 NUMBER;
BEGIN
       KNT := CHECK_ITEM(CODE);
       SELECT COUNT(*) INTO KNT1
       FROM ITEMS
       WHERE P_CODE = CODE;
       IF KNT = 1 AND KNT1 = 1 THEN
              IF QT IS NOT NULL THEN
                     UPDATE ITEMS SET QTY = QT WHERE P_CODE = CODE;
              END IF;
              IF PR IS NOT NULL THEN
                     UPDATE ITEMS SET PRICE = PR WHERE P_CODE = CODE;
              END IF;
              DBMS_OUTPUT.PUT_LINE('ITEM INFORMATION HAS BEEN UPDATE
                                                            SUCCESSFULLY ... ');
       ELSIF KNT1 > 1 THEN
              RAISE TOO_MANY_ROWS;
       ELSE
              RAISE NO_DATA_FOUND;
       END IF;
EXCEPTION
       WHEN NO_DATA_FOUND THEN
              DBMS_OUTPUT.PUT_LINE('ENTERED ITEM CODE DETAILS DOES
                                                           NOT EXIST ... ');
       WHEN TOO_MANY_ROWS THEN
              DBMS_OUTPUT.PUT_LINE('MULTIPLE ROWS FOUND ...........
                                                           ............');
       WHEN OTHERS THEN
              DBMS_OUTPUT.PUT_LINE('SOME ERROR OCCURRED ... ');
END;
/
```

```
Procedure created.
```

```
BEGIN
```

```
		UPDATE_ITEM_TMR_E('HH15P',5,NULL);
	END;
	/

MULTIPLE ROWS FOUND .........................

PL/SQL procedure successfully completed.


	CREATE OR REPLACE PROCEDURE ADD_ITEM (
		CODE IN ITEMS.P_CODE%TYPE,
		DESCRIP IN ITEMS.DESCR%TYPE,
		IDT IN ITEMS.IN_DATE%TYPE,
		MQTY IN ITEMS.MIN_QTY%TYPE,
		QT IN ITEMS.QTY%TYPE,
		PR IN ITEMS.PRICE%TYPE,
		VCODE IN ITEMS.V_CODE%TYPE)
	AS
		KNT NUMBER;
	BEGIN
		KNT := CHECK_ITEM(CODE);
		IF KNT>=1 THEN
			RAISE TOO_MANY_ROWS;
		ELSE
			INSERT INTO ITEMS VALUES (CODE, DESCRIP, IDT, MQTY,
											QT, PR, VCODE);
		END IF;
	EXCEPTION
		WHEN TOO_MANY_ROWS THEN
			DBMS_OUTPUT.PUT_LINE('MULTIPLE ROWS FOUND .....');
	END;
	/

Procedure created.

	EXEC ADD_ITEM('HH15P','NEW ITEM-2','23-SEP-2023',2,150,25,NULL);

MULTIPLE ROWS FOUND .....

PL/SQL procedure successfully completed.
```

****

QUERY-10: Now extend the procedure in Query-09 as SHOW_ITEM_TMR_HANDLED to print the rows returned by the SELECT query after catching the appropriate exception.

****

```
CREATE OR REPLACE PROCEDURE SHOW_ITEM_TMR_HANDLED(
      CODE IN ITEMS.P_CODE%TYPE)
AS
      I_REC ITEMS%ROWTYPE;
      KNT NUMBER;
      KNT1 NUMBER;
      CURSOR SHOW_ITEM_CUR IS
                  SELECT *
                  FROM ITEMS
                  WHERE P_CODE = CODE;
BEGIN
      KNT := CHECK_ITEM(CODE);
      SELECT COUNT(*) INTO KNT1
      FROM ITEMS
      WHERE P_CODE=CODE;
      IF KNT=1 AND KNT1 = 1 THEN
            OPEN SHOW_ITEM_CUR;
            LOOP
                  FETCH SHOW_ITEM_CUR INTO I_REC;
                  EXIT WHEN SHOW_ITEM_CUR%NOTFOUND;
                  DBMS_OUTPUT.PUT(RPAD(I_REC.P_CODE,5)||' ');
                  DBMS_OUTPUT.PUT(RPAD(I_REC.DESCR,30)||' ');
                  DBMS_OUTPUT.PUT(RPAD(I_REC.IN_DATE,11)||' ');
                  DBMS_OUTPUT.PUT(RPAD(I_REC.MIN_QTY,3)||' ');
                  DBMS_OUTPUT.PUT(RPAD(I_REC.QTY,4)||' ');
                  DBMS_OUTPUT.PUT(RPAD(I_REC.PRICE,6)||' ');
                  DBMS_OUTPUT.PUT_LINE(RPAD(I_REC.V_CODE,6));
            END LOOP;
            CLOSE SHOW_ITEM_CUR;
      ELSIF KNT1 > 1 THEN
            RAISE TOO_MANY_ROWS;
      ELSE
            RAISE NO_DATA_FOUND;
      END IF;
EXCEPTION
      WHEN NO_DATA_FOUND THEN
```

```
                DBMS_OUTPUT.PUT_LINE('NO ITEM WITH ITEM CODE '||CODE||
                                                        ' FOUND ...');
          WHEN TOO_MANY_ROWS THEN
                DBMS_OUTPUT.PUT_LINE(CODE||' MULTIPLE ROWS .............
                                    ..................................');
                OPEN SHOW_ITEM_CUR;
                LOOP
                      FETCH SHOW_ITEM_CUR INTO I_REC;
                      EXIT WHEN SHOW_ITEM_CUR%NOTFOUND;
                      DBMS_OUTPUT.PUT(RPAD(I_REC.P_CODE,5)||' ');
                      DBMS_OUTPUT.PUT(RPAD(I_REC.DESCR,30)||' ');
                      DBMS_OUTPUT.PUT(RPAD(I_REC.IN_DATE,11)||' ');
                      DBMS_OUTPUT.PUT(RPAD(I_REC.MIN_QTY,3)||' ');
                      DBMS_OUTPUT.PUT(RPAD(I_REC.QTY,4)||' ');
                      DBMS_OUTPUT.PUT(RPAD(I_REC.PRICE,6)||' ');
                      DBMS_OUTPUT.PUT_LINE(RPAD(I_REC.V_CODE,6));
                END LOOP;
                CLOSE SHOW_ITEM_CUR;
          WHEN OTHERS THEN
                DBMS_OUTPUT.PUT_LINE('SOME ERROR OCCURRED ...');
    END;
    /

Procedure created.


    BEGIN
          SHOW_ITEM_TMR_HANDLED('HH15P');
          DBMS_OUTPUT.PUT_LINE('----------------------------------------
                                      -----------------------------');
          SHOW_ITEM_TMR_HANDLED('HH15X');
    END;
    /


HH15P MULTIPLE ROWS ..................................................
HH15P NEW ITEM ...                  23-SEP-21   3   22   222
HH15P NEW ITEM-2                    23-SEP-21   2   150  25
----------------------------------------------------------------------
HH15X Hanging Hook 15in             10-JAN-13   25  200  5.75   24992

PL/SQL procedure successfully completed.
```

Conclusion:

So there you have it – procedures and functions. They are, perhaps, the most fundamental aspects of computer programming; they are the Lego bricks upon which everything else must be built. All well-written programs are made up of procedures that include other procedures and call functions that, in turn, may call other functions and perform other processes.

The terms to remember are reusability and modularity. If we know we will do an action more than once (for example, add a new friend), we must construct a procedure to accomplish it; if we know we will ask a question more than once (what is a friend's phone number?), we must create a function to answer it. And, to the greatest extent feasible, we should make our methods and functionalities simple and modest. That way, when we need to develop complex algorithms, all we have to do is call one procedure after the other in whichever order we choose.

The more concentrated our methods are, the more easily complicated structures may be built. It's similar to Lego.

Viva Questions:

1. **State the advantages of using stored functions and procedures.**

   The following are the primary benefits of stored procedures:

- Increased Performance – Because stored procedures are built once and saved in executable form, procedure calls are rapid and efficient. As a result, the answer is prompt. Because the executable code is automatically cached, the memory needs are reduced.
- Better Productivity - Because the same piece of code is used repeatedly, it results in higher productivity.
- Ease of Use – To construct a stored procedure, any Java Integrated Development Environment may be used (IDE). They may then be deployed on any network architecture layer.
- Scalability – By separating application processing on the server, stored procedures improve scalability.
- Maintainability – Because scripts reside in one area, maintaining a process on a server is considerably easier than keeping copies on multiple client workstations.
- Security - Access to Oracle data can be limited by enabling users to alter the data only through stored procedures that run with the privileges of their definer.

2. **Explain about IN, OUT and IN OUT variables in PL/SQL procedures.**

   IN

   A variable given as mode IN is always read-only. A variable in IN mode can be read and utilized by the procedure/function, but it cannot be modified and cannot be the recipient of an assignment operation. Variables sent through IN mode are regarded constants inside the scope of the procedure or function. The IN mode is the default method for passing a variable, however it is advised for maintainability reasons to always declare the variable passing mode when you define the variable. Variables passed IN can also be set a default value, as previously mentioned.

OUT

In OUT mode, a variable is utilised to return information from the procedure to the caller application. It is a read-only variable with no value until the block assigns one to it. When the procedure is called, an OUT variable is created but not initialized. When the procedure is finished, the variable value is transferred to the variable given in the call. As a result, a variable supplied in OUT mode cannot be set to a default value or read within the method. The caller code cannot provide a literal value to an OUT variable because the variable value is copied back to the supplied variable after the function finishes. If the procedure throws an exception that is not caught, the OUT variable will not be copied when the procedure exits.

IN OUT

A variable passed in IN OUT mode has properties from both the IN and the OUT modes. The variable value is handed in and may be read by the method. The procedure can also alter the value, which will be copied back to the given variable after the process is finished. An IN OUT variable, like one provided in OUT mode, cannot have a default value and cannot be passed as a literal. If the method finishes abnormally (like in an exception), the IN OUT variable is not transferred back to the variable given in.

3. **Differentiate between a stored function and a stored procedure.**

- The function must return a value, although it is optional in Stored Procedure. A process can also return zero or n values.
- Functions can only have input parameters, but Procedures can contain both input and output parameters.
- Functions can be invoked from Procedures, while Procedures cannot be invoked from Functions.
- The procedure supports both SELECT and DML (INSERT/UPDATE/DELETE) statements, whereas the function only supports SELECT statements.
- Procedures cannot be incorporated in a SELECT query, however Functions may.
- Stored Procedures cannot be used anywhere in the WHERE/HAVING/SELECT portion of SQL queries, but Functions may.
- Table-returning functions can be thought of as another row set. This may be used in conjunction with other tables in JOINS.
- Inline Functions are views that accept arguments and may be utilised in JOINS and other row set procedures.
- A try-catch block can handle an exception in a Procedure, but it cannot be used in a Function.
- Transactions are permitted in Procedure but are not permitted in Function.

4. **Write about the RAISE_APPLICATION_ERROR() procedure of Oracle.**

Oracle's raise application error method allows the developer to raise an exception and correlate an error number and message with the operation. This enables the program to generate application errors as well as Oracle faults. Error numbers range from -20,000 to -20,999.

Oracle's raise application error function allows you to generate custom error numbers within your applications. Starting with -20000 and progressing through -20999, you may produce errors and their accompanying messages (a grand total of 1,000 error numbers that you can use).