***** EXPERIMENT NO: 06 *****

Author: Atharva Paliwal
Roll No: 40 [5B]
Date: 06-November-2020

AIM: To write and execute stored procedures and 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.

****

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
(ENO EMPP.EID%TYPE,EMP_REC IN OUT EMPP%ROWTYPE)
AS
BEGIN
 SELECT * INTO EMP_REC FROM EMPP WHERE EID = ENO;
END;
/
```

```
Procedure created.
```

```
DECLARE
 EMP_REC EMPP%ROWTYPE;
BEGIN
 SHOW_EMPLOYEE(&ENO,EMP_REC);
 DBMS_OUTPUT.PUT_LINE('EMPLOYEE INFO: '||EMP_REC.EID||' '||EMP_REC.ENAME
 ||' '||EMP_REC.HIREDATE||' '||EMP_REC.SALARY);
END;
/
```

```
Enter value for eno: 7101
old   4:  SHOW_EMPLOYEE(&ENO,EMP_REC);
new   4:  SHOW_EMPLOYEE(7101,EMP_REC);
EMPLOYEE INFO: 7101 Eugene Sabatini 10-OCT-06 150000

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.

****

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

Procedure created.

SELECT OBJECT_NAME, OBJECT_TYPE, CREATED FROM USER_OBJECTS
WHERE OBJECT_TYPE='PROCEDURE';

OBJECT_NAME   OBJECT_TYPE         CREATED
------------- ------------------- ---------
ADD_EMPLOYEE  PROCEDURE           05-NOV-20
SHOW_EMPLOYEE PROCEDURE           05-NOV-20

SELECT COUNT(*) FROM EMPP;

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

BEGIN
```

```
ADD_EMPLOYEE(&EID,'&ENAME','&HIREDATE','&DESIGNATION',&SALARY);
END;

Enter value for eid: 7118
Enter value for ename: Atharva Paliwal
Enter value for hiredate: 07-Jul-2020
Enter value for designation: Teaching Asst.
Enter value for salary: 25000
old   2: ADD_EMPLOYEE(&EID,'&ENAME','&HIREDATE','&DESIGNATION',&SALARY);
new   2: ADD_EMPLOYEE(7118,'Atharva Paliwal','07-Jul-2020','Teaching
Asst.',25000);

SQL> /
Enter value for eid: 7119
Enter value for ename: Atulya Bharat
Enter value for hiredate: 03-Aug-2005
Enter value for designation: Professor
Enter value for salary: 162000
old   2: ADD_EMPLOYEE(&EID,'&ENAME','&HIREDATE','&DESIGNATION',&SALARY);
new   2: ADD_EMPLOYEE(7119,'Atulya Bharat','03-Aug-
2005','Professor',162000);

PL/SQL procedure successfully completed.

SELECT COUNT(*) FROM EMPP;

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

****

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 OR REPLACE PROCEDURE REMOVE_EMPLOYEE
(NAME EMPP.ENAME%TYPE)
AS
```

```
BEGIN
DELETE FROM EMPP WHERE LOWER(ENAME) = NAME;
IF SQL%NOTFOUND THEN
DBMS_OUTPUT.PUT_LINE('RECORD NOT FOUND');
ELSE
DBMS_OUTPUT.PUT_LINE('RECORD DELETED');
END IF;
END;
/

Procedure created.

BEGIN
REMOVE_EMPLOYEE('&NAME');
END;
/

Enter value for name: atulya bharat
old   2: REMOVE_EMPLOYEE('&NAME');
new   2: REMOVE_EMPLOYEE('atulya bharat');
RECORD DELETED

PL/SQL procedure successfully completed.

SQL> /
Enter value for name: ritika mehta
old   2: REMOVE_EMPLOYEE('&NAME');
new   2: REMOVE_EMPLOYEE('ritika mehta');
RECORD NOT FOUND

PL/SQL procedure successfully completed.
```

****

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 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 MIN_QTY DEFAULT 2;

Table altered.

ALTER TABLE ITEMS
MODIFY IN_DATE DEFAULT SYSDATE;

Table altered.

SELECT CONSTRAINT_NAME,CONSTRAINT_TYPE FROM USER_CONSTRAINTS
WHERE TABLE_NAME ='ITEMS';

CONSTRAINT_NAME               C
------------------------------ -
SYS_C0011927                  C
SYS_C0011928                  C
SYS_C0011929                  C
SYS_C0011930                  C
SYS_C0011931                  C
SYS_C0011932                  C
ITEMS_PK_P_CODE               P

7 rows selected.
```

```
CREATE OR REPLACE FUNCTION CHECK_ITEM(INPUT_CODE ITEMS.P_CODE%TYPE)
RETURN NUMBER
AS
STATUS NUMBER(2) :=0;
BEGIN
SELECT COUNT(*) INTO STATUS FROM ITEMS WHERE P_CODE = INPUT_CODE;
IF STATUS > 0 THEN
RETURN 1;
ELSE
RETURN 0;
END IF;
END;
/

Function created.

BEGIN
IF CHECK_ITEM('&INPUT_CODE') = 1 THEN
DBMS_OUTPUT.PUT_LINE('RECORD FOUND');
ELSE
DBMS_OUTPUT.PUT_LINE('RECORD NOT FOUND');
END IF;
END;
/

Enter value for input_code: HC100
old   2: IF CHECK_ITEM('&INPUT_CODE') = 1 THEN
new   2: IF CHECK_ITEM('HC100') = 1 THEN
RECORD FOUND

PL/SQL procedure successfully completed.

SQL> /
Enter value for input_code: PP100
old   2: IF CHECK_ITEM('&INPUT_CODE') = 1 THEN
new   2: IF CHECK_ITEM('PP100') = 1 THEN
RECORD NOT FOUND

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
(P_CODE ITEMS.P_CODE%TYPE, DESCR ITEMS.DESCR%TYPE, IN_DATE
ITEMS.IN_DATE%TYPE, QTY
ITEMS.QTY%TYPE, MIN_QTY ITEMS.MIN_QTY%TYPE, PRICE ITEMS.PRICE%TYPE, V_CODE
ITEMS.V_CODE%TYPE)
AS
BEGIN
 INSERT INTO ITEMS VALUES
 (P_CODE,DESCR,IN_DATE,QTY,MIN_QTY,PRICE,V_CODE);
END;
/

Procedure created.

SELECT COUNT(*) FROM ITEMS;

COUNT(*)
----------
        22

BEGIN
ADD_ITEM('&P_CODE','&DESCR','&IN_DATE',&QTY,&MIN_QTY,&PRICE,&V_CODE);
END;
/



Enter value for p_code: RD304
Enter value for descr: Rat Tail File
Enter value for in_date: 29-SEP-20
Enter value for qty: 10
Enter value for min_qty: 2
Enter value for price: 98.02
```

```
Enter value for v_code: 23119
old   2:
ADD_ITEM('&P_CODE','&DESCR','&IN_DATE',&QTY,&MIN_QTY,&PRICE,&V_CODE);
new   2: ADD_ITEM('RD304','Rat Tail File','29-SEP-20',10,2,98.02,23119);

PL/SQL procedure successfully completed.
```

```
SELECT COUNT(*) FROM ITEMS;

COUNT(*)
----------
 23
```

****

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
(IN_P_CODE ITEMS.P_CODE%TYPE, IN_QTY ITEMS.QTY%TYPE, IN_PRICE
ITEMS.PRICE%TYPE)
AS
BEGIN
 IF(CHECK_ITEM(IN_P_CODE)=1) THEN
 UPDATE ITEMS
 SET QTY = IN_QTY,
 PRICE = IN_PRICE
 WHERE P_CODE = IN_P_CODE;
 DBMS_OUTPUT.PUT_LINE('Data updated');
 ELSIF(CHECK_ITEM(IN_P_CODE)=0) THEN
 RAISE NO_DATA_FOUND;
 END IF;
 EXCEPTION
 WHEN NO_DATA_FOUND THEN
 DBMS_OUTPUT.PUT_LINE('No Data Found');
END;
```

```
/

Procedure created.

SELECT QTY,PRICE FROM ITEMS WHERE P_CODE='RD304';


       QTY      PRICE
---------- ----------
         2      98.02

BEGIN
 UPDATE_ITEM ('&IN_P_CODE',&IN_QTY,&IN_PRICE);
END;
/

Enter value for in_p_code: RD304
Enter value for in_qty: 5
Enter value for in_price: 100
old   2:  UPDATE_ITEM ('&IN_P_CODE',&IN_QTY,&IN_PRICE);
new   2:  UPDATE_ITEM ('RD304',5,100);
Data updated

PL/SQL procedure successfully completed.

SELECT QTY,PRICE FROM ITEMS WHERE P_CODE='RD304';

       QTY      PRICE
---------- ----------
         5        100
```

****
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
(IN_P_CODE ITEMS.P_CODE%TYPE, IN_QTY ITEMS.QTY%TYPE, IN_PRICE
ITEMS.PRICE%TYPE)
AS
BEGIN
 IF(CHECK_ITEM(IN_P_CODE)=1) THEN
 UPDATE ITEMS
 SET QTY = IN_QTY,
 PRICE = IN_PRICE
 WHERE P_CODE = IN_P_CODE;
 DBMS_OUTPUT.PUT_LINE('Data updated');
 ELSIF(CHECK_ITEM(IN_P_CODE)=0) THEN
 ADD_ITEM(IN_P_CODE,'NEW ITEM',SYSDATE,IN_QTY/8,IN_QTY,IN_PRICE,NULL);
 DBMS_OUTPUT.PUT_LINE('New Item Added');
 END IF;
END;
/

Procedure created.

BEGIN
UPDATE_ITEM_ADD_WHEN_NOT_FOUND ('&IN_P_CODE',&IN_QTY,&IN_PRICE);
END;
/

Enter value for in_p_code: RD200
Enter value for in_qty: 1
```

```
Enter value for in_price: 200
old   2: UPDATE_ITEM_ADD_WHEN_NOT_FOUND ('&IN_P_CODE',&IN_QTY,&IN_PRICE);
new   2: UPDATE_ITEM_ADD_WHEN_NOT_FOUND ('RD200',1,200);
New Item Added

PL/SQL procedure successfully completed.
```

```
SELECT * FROM ITEMS WHERE P_CODE = 'RD200';
```

```
P_COD DESCR     IN_DATE      MIN_QTY        QTY      PRICE     V_CODE
----- --------- --------- ---------- ---------- ---------- ----------
RD200 NEW ITEM  05-NOV-20          0          1        200
```

****

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.

****

```
CREATE OR REPLACE PROCEDURE SHOW_ITEM
(IN_P_CODE ITEMS.P_CODE%TYPE, ITEM_REC IN OUT ITEMS%ROWTYPE)
AS
BEGIN
 IF(CHECK_ITEM(IN_P_CODE)=1) THEN
 SELECT * INTO ITEM_REC FROM ITEMS WHERE P_CODE = IN_P_CODE;
 DBMS_OUTPUT.PUT_LINE('ITEM INFO: '||ITEM_REC.P_CODE||' '||ITEM_REC.DESCR
 ||' '||ITEM_REC.IN_DATE||' '||ITEM_REC.MIN_QTY||' '||
 ITEM_REC.QTY||' '||ITEM_REC.PRICE||' '||ITEM_REC.V_CODE);
 ELSIF (CHECK_ITEM(IN_P_CODE)=0) THEN
 DBMS_OUTPUT.PUT_LINE('No Data Found');
 END IF;
END;
/
```

```
Procedure created.
```

```
DECLARE
 ITEM_REC ITEMS%ROWTYPE;
BEGIN
 SHOW_ITEM('&IN_P_CODE',ITEM_REC);
END;
/

Enter value for in_p_code: SM48X
old   4:  SHOW_ITEM('&IN_P_CODE',ITEM_REC);
new   4:  SHOW_ITEM('SM48X',ITEM_REC);
ITEM INFO: SM48X Steel Malting Mesh 17-JAN-20 5 18 62.95 25595

PL/SQL procedure successfully completed.

SQL> /
Enter value for in_p_code: SM40X
old   4:  SHOW_ITEM('&IN_P_CODE',ITEM_REC);
new   4:  SHOW_ITEM('SM40X',ITEM_REC);
No Data 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

****

Before executing Query 9

```
ALTER TABLE ITEMS DROP PRIMARY KEY;

Table altered.

EXEC ADD_ITEM('HH15P','NEW ITEM-2',SYSDATE,150,TRUNC(150/8),25,NULL);
```

```
PL/SQL procedure successfully completed.

SELECT * FROM ITEMS WHERE P_CODE = 'HH15P' AND DESCR = 'NEW ITEM-2';

P_COD DESCR          IN_DATE      MIN_QTY        QTY      PRICE     V_CODE
----- -------------- --------- ---------- ---------- ---------- ----------
HH15P NEW ITEM-2     05-NOV-20        150         18         25


CREATE OR REPLACE PROCEDURE SHOW_ITEM_TMR_E (
 ITEM_CODE ITEMS.P_CODE%TYPE)
 IS
 CNT NUMBER(2);
 IRECORD ITEMS%ROWTYPE;
 TOO_MANY_ROWS EXCEPTION;
 NO_ROW EXCEPTION;
BEGIN
 IF CHECK_ITEM(ITEM_CODE)=0 THEN
 RAISE NO_ROW;
 ELSE
 SELECT COUNT(*) INTO CNT FROM ITEMS WHERE P_CODE=ITEM_CODE;
 IF CNT>1 THEN
 RAISE TOO_MANY_ROWS;
 ELSIF CNT=1 THEN
 SELECT * INTO IRECORD FROM ITEMS WHERE P_CODE=ITEM_CODE;
 DBMS_OUTPUT.PUT_LINE ('P_CODE: '||IRECORD.P_CODE);
 DBMS_OUTPUT.PUT_LINE ('DESCRIPT: '||IRECORD.DESCR);
 DBMS_OUTPUT.PUT_LINE ('IN_DATE: '||IRECORD.IN_DATE);
 DBMS_OUTPUT.PUT_LINE ('MIN_QTY: '||IRECORD.MIN_QTY);
 DBMS_OUTPUT.PUT_LINE ('QTY: '||IRECORD.QTY);
 DBMS_OUTPUT.PUT_LINE ('PRICE: '||IRECORD.PRICE);
 DBMS_OUTPUT.PUT_LINE ('V_CODE: '||IRECORD.V_CODE);
 END IF;
 END IF;
EXCEPTION
 WHEN TOO_MANY_ROWS THEN
 DBMS_OUTPUT.PUT_LINE ('MULTIPLE RECORDS.................');
 WHEN NO_DATA_FOUND THEN
 DBMS_OUTPUT.PUT_LINE ('INVALID ITEM_CODE');
```

```
 WHEN NO_ROW THEN
 DBMS_OUTPUT.PUT_LINE ('ITEM IS NOT PRESENT ');
END;
/

Procedure created.

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

P_CODE: HH15P
DESCRIPT: NEW ITEM-2
IN_DATE: 05-NOV-20
MIN_QTY: 150
QTY: 18
PRICE: 25
V_CODE:
ITEM IS NOT PRESENT

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 (
 ITEM_CODE ITEMS.P_CODE%TYPE)
 IS
 CNT NUMBER(2);
 IRECORD ITEMS%ROWTYPE;
 TOO_MANY_ROWS EXCEPTION;
 NO_ROW EXCEPTION;
 IRECORE ITEMS%ROWTYPE;
```

```
BEGIN
 IF CHECK_ITEM(ITEM_CODE)=0 THEN
 RAISE NO_ROW;
 ELSE
 SELECT COUNT(*) INTO CNT FROM ITEMS WHERE P_CODE=ITEM_CODE;
 IF CNT>1 THEN
 RAISE TOO_MANY_ROWS;
 ELSIF CNT=1 THEN
 SELECT * INTO IRECORD FROM ITEMS WHERE P_CODE=ITEM_CODE;
 DBMS_OUTPUT.PUT_LINE ('P_CODE: '||IRECORD.P_CODE);
 DBMS_OUTPUT.PUT_LINE ('DESCRIPT: '||IRECORD.DESCR);
 DBMS_OUTPUT.PUT_LINE ('IN_DATE: '||IRECORD.IN_DATE);
 DBMS_OUTPUT.PUT_LINE ('MIN_QTY: '||IRECORD.MIN_QTY);
 DBMS_OUTPUT.PUT_LINE ('QTY: '||IRECORD.QTY);
 DBMS_OUTPUT.PUT_LINE ('PRICE: '||IRECORD.PRICE);
 DBMS_OUTPUT.PUT_LINE ('V_CODE: '||IRECORD.V_CODE);
 END IF;
 END IF;
EXCEPTION
 WHEN TOO_MANY_ROWS THEN
 DBMS_OUTPUT.PUT_LINE ('MULTIPLE RECORDS.................');
 DBMS_OUTPUT.PUT_LINE (CHR(10));
 FOR I IN (SELECT * FROM ITEMS) LOOP
 IF I.P_CODE=ITEM_CODE THEN
 DBMS_OUTPUT.PUT_LINE ('P_CODE: '||I.P_CODE);
 DBMS_OUTPUT.PUT_LINE ('DESCRIPT: '||I.DESCR);
 DBMS_OUTPUT.PUT_LINE ('IN_DATE: '||I.IN_DATE);
 DBMS_OUTPUT.PUT_LINE ('MIN_QTY: '||I.MIN_QTY);
 DBMS_OUTPUT.PUT_LINE ('QTY: '||I.QTY);
 DBMS_OUTPUT.PUT_LINE ('PRICE: '||I.PRICE);
 DBMS_OUTPUT.PUT_LINE ('V_CODE: '||I.V_CODE);
 DBMS_OUTPUT.PUT_LINE (CHR(10));
 END IF;
 END LOOP;
 WHEN NO_DATA_FOUND THEN
 DBMS_OUTPUT.PUT_LINE ('INVALID ITEM_CODE');
 WHEN NO_ROW THEN
 DBMS_OUTPUT.PUT_LINE ('ITEM IS NOT PRESENT ');
END;
```

/

Procedure created.

EXEC SHOW_ITEM_TMR_HANDLED('HH15P');

MULTIPLE RECORDS.................

P_CODE: HH15P

DESCRIPT: NEW ITEM2

IN_DATE: 30-SEP-20

MIN_QTY: 2

QTY: 150

PRICE: 25

V_CODE:

P_CODE: HH15P

DESCRIPT: NEW ITEM..

IN_DATE: 28-MAR-17

MIN_QTY: 12

QTY: 100

PRICE: 5.8

V_CODE:

PL/SQL procedure successfully completed.

VIVA-VOCE

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

Answer - Stored procedures provide several advantages

1. To help you build powerful database applications
2. Better performance
3. Higher productivity
4. Ease of use
5. Increased scalability

Question 2. Explain about IN,OUT and IN OUT variables in PL/SQL Procedures

Answer-

IN: A variable passed in this mode is of read only nature. This is to say, the value cannot be changed and its scope is restricted within the procedure. The procedure receives a value from this argument when the procedure is called.

OUT: In this mode, a variable is write only and can be passed back to the calling program. It cannot be read inside the procedure and needs to be assigned a value.

INOUT: This procedure has features of both IN and OUT mode. The procedure can also read the variables value and can also change it to pass it to the calling function.

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

Answer-

- The function must return a value but in **Stored Procedure** it is optional. Even a procedure can return zero or n values.
- Functions can have only input parameters for it whereas Procedures can have input or output parameters.
- Functions can be called from Procedure whereas Procedures cannot be called from a Function.

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

Answer-

The procedure RAISE_APPLICATION_ERROR() allows you to issue an user-defined error from a code block or stored program.

By using this procedure, you can report errors to the callers instead of returning unhandled exceptions.

The RAISE_APPLICATION_ERROR() has the following syntax:

```
raise_application_error(
   error_number,
   message
   [, {TRUE | FALSE}]
);
```

INFERENCES

- We learnt to execute stored functions and procedures.
- We also learnt to handle exceptions and about to how declare user defined exceptions.
