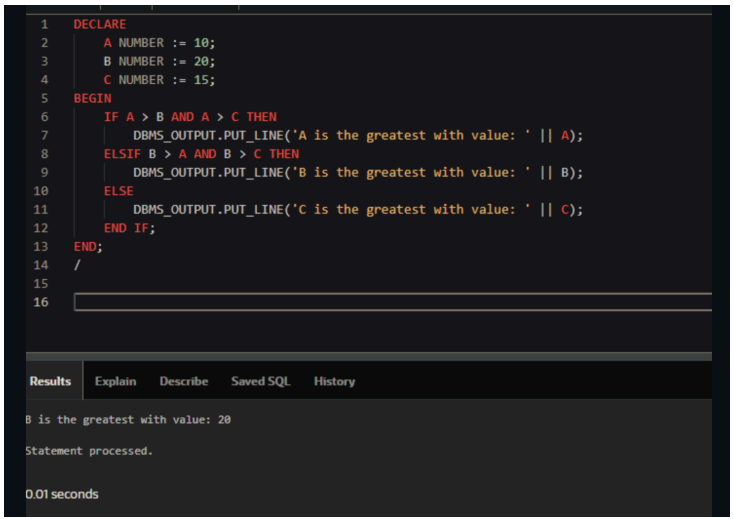
EXPERIMENT-14

Title: To understand the concepts of function and procedure in PL/SQL.

Objective: Students will be able to implement the Pl/SQL programs using function and procedure.

1) Implement the above experiments of PL/SQL using functions and procedures.



```sql
DECLARE
    i NUMBER := 1;
BEGIN
    WHILE i <= 20 LOOP
        DBMS_OUTPUT.PUT_LINE('Welcome to PL/SQL Programming');
        i := i + 1;
    END LOOP;
END;
/
```

Results Explain Describe Saved SQL History

```
Welcome to PL/SQL Programming
Welcome to PL/SQL Programming
Welcome to PL/SQL Programming
Welcome to PL/SQL Programming
Welcome to PL/SQL Programming
Welcome to PL/SQL Programming
Welcome to PL/SQL Programming
Welcome to PL/SQL Programming
Welcome to PL/SQL Programming
Welcome to PL/SQL Programming
Welcome to PL/SQL Programming
Welcome to PL/SQL Programming
Welcome to PL/SQL Programming
Welcome to PL/SQL Programming
Welcome to PL/SQL Programming
Welcome to PL/SQL Programming
Welcome to PL/SQL Programming
Welcome to PL/SQL Programming
Welcome to PL/SQL Programming
Welcome to PL/SQL Programming

Statement processed.
```

0.01 seconds

```sql
DECLARE
    num NUMBER := 5;
    fact NUMBER := 1;
    i NUMBER := 1;
BEGIN
    FOR i IN 1..num LOOP
        fact := fact * i;
    END LOOP;
    DBMS_OUTPUT.PUT_LINE('Factorial of ' || num || ' is: ' || fact);
END;
/
```

Results Explain Describe Saved SQL History

```
Factorial of 5 is: 120

Statement processed.
```

```sql
DECLARE
    n NUMBER := 10;
    a NUMBER := 0;
    b NUMBER := 1;
    temp NUMBER;
    i NUMBER := 1;
BEGIN
    DBMS_OUTPUT.PUT_LINE('Fibonacci Series:');
    DBMS_OUTPUT.PUT_LINE(a);
    DBMS_OUTPUT.PUT_LINE(b);
    WHILE i <= n - 2 LOOP
        temp := a + b;
        DBMS_OUTPUT.PUT_LINE(temp);
        a := b;
        b := temp;
        i := i + 1;
    END LOOP;
END;
/
```

Results Explain Describe Saved SQL History

```
Fibonacci Series:
0
1
1
2
3
5
8
13
21
34

Statement processed.
```

```
DECLARE
    N NUMBER := 10;
    sum_result NUMBER := 0;
    i NUMBER := 1;
BEGIN
    WHILE i <= N LOOP
        sum_result := sum_result + i;
        i := i + 1;
    END LOOP;

    DBMS_OUTPUT.PUT_LINE('Sum of the first ' || N || ' numbers is: ' || sum_result);
END;
/
```

Results | Explain | Describe | Saved SQL | History

```
Sum of the first 10 numbers is: 55

Statement processed.
```

EXPERIMENT-15

Title: To understand the concepts of implicit and explicit cursor.

Objective: Students will be able to implement the concept of implicit and explicit cursor.

1. Using implicit cursor update the salary by an increase of 10% for all the records in EMPLOYEES table, and finally display how many records have been updated. If no records exist display the message "**No Change**".

2. Using explicit cursor fetch the employee name, employee_id and salary of all the records from EMPLOYEES table.

3. Using explicit cursor Insert the records from EMPLOYEES table for the columns employee_id, Last_Name and salary for those records whose salary exceeds 2500 into a new table TEMP_EMP

```
CREATE TABLE EMPLOYEES (
    employee_id NUMBER PRIMARY KEY,
    employee_name VARCHAR2(50),
    last_name VARCHAR2(50),
    salary NUMBER(10, 2)
);
```

Results | Explain | Describe | Saved SQL | History

```
Table created.

0.05 seconds
```

```
INSERT ALL
    INTO EMPLOYEES (employee_id, employee_name, last_name, salary) VALUES (1, 'John', 'Doe', 2000)
    INTO EMPLOYEES (employee_id, employee_name, last_name, salary) VALUES (2, 'Jane', 'Smith', 3000)
    INTO EMPLOYEES (employee_id, employee_name, last_name, salary) VALUES (3, 'Bob', 'Johnson', 4000)
SELECT 1 FROM DUAL;
```

Results | Explain | Describe | Saved SQL | History

```
3 row(s) inserted.
```

```
CREATE TABLE TEMP_EMP (
    employee_id NUMBER,
    last_name VARCHAR2(50),
    salary NUMBER(10, 2)
);
```

Results | Explain | Describe | Saved SQL | History

```
Table created.

0.03 seconds
```

```plsql
DECLARE
    v_count NUMBER := 0;
BEGIN
    UPDATE EMPLOYEES
    SET SALARY = SALARY * 1.10;
    v_count := SQL%ROWCOUNT;

    IF v_count > 0 THEN
        DBMS_OUTPUT.PUT_LINE(v_count || ' records have been updated.');
    ELSE
        DBMS_OUTPUT.PUT_LINE('No Change');
    END IF;
END;
```

Results | Explain | Describe | Saved SQL | History

```
3 records have been updated.

1 row(s) updated.
```

0.03 seconds

```plsql
DECLARE
    CURSOR emp_cursor IS
        SELECT employee_name, employee_id, salary FROM EMPLOYEES;
    emp_rec emp_cursor%ROWTYPE;
BEGIN
    OPEN emp_cursor;
    LOOP
        FETCH emp_cursor INTO emp_rec;
        EXIT WHEN emp_cursor%NOTFOUND;
        DBMS_OUTPUT.PUT_LINE('Name: ' || emp_rec.employee_name ||
                             ', ID: ' || emp_rec.employee_id ||
                             ', Salary: ' || emp_rec.salary);
    END LOOP;
    CLOSE emp_cursor;
END;
```

Results | Explain | Describe | Saved SQL | History

```
Name: John, ID: 1, Salary: 2200
Name: Jane, ID: 2, Salary: 3300
Name: Bob, ID: 3, Salary: 4400

Statement processed.
```

0.01 seconds

```plsql
DECLARE
    CURSOR emp_cursor IS
        SELECT employee_id, last_name, salary
        FROM EMPLOYEES
        WHERE salary > 2500;
    emp_rec emp_cursor%ROWTYPE;
BEGIN
    OPEN emp_cursor;

    LOOP
        FETCH emp_cursor INTO emp_rec;
        EXIT WHEN emp_cursor%NOTFOUND;

        INSERT ALL
            INTO TEMP_EMP (employee_id, last_name, salary)
            VALUES (emp_rec.employee_id, emp_rec.last_name, emp_rec.salary)
        SELECT 1 FROM DUAL;

    END LOOP;

    CLOSE emp_cursor;

    COMMIT;
END;
```

Results | Explain | Describe | Saved SQL | History

```
1 row(s) inserted.
```

0.05 seconds

EXPERIMENT-16

Title: To understand the concepts of Trigger.

Objective: Students will be able to implement the concept of trigger.

CUSTOMER Table:

1) Create a row level trigger for the customers table that would fire for INSERT or UPDATE or DELETE operations performed on the CUSTOMERS table. This trigger will display the salary difference between the old values and new values.

```
CREATE TABLE STUDENT (
    ID NUMBER PRIMARY KEY,
    NAME VARCHAR2(50),
    AGE NUMBER,
    ADDRESS VARCHAR2(100),
    SALARY NUMBER(10, 2)
);
```

Results | Explain | Describe | Saved

```
Table created.
```

```
CREATE OR REPLACE TRIGGER Salary_Diff_Trigger
AFTER INSERT OR UPDATE OR DELETE
ON STUDENT
FOR EACH ROW
DECLARE
    v_diff NUMBER;
BEGIN
    -- Handle UPDATE operation
    IF UPDATING THEN
        v_diff := :NEW.SALARY - :OLD.SALARY;
        DBMS_OUTPUT.PUT_LINE('UPDATE operation: Salary changed from ' || :OLD.SALARY || ' to ' || :NEW.SALARY || ' (Difference: ' || v_diff || ').');
    END IF;

    -- Handle INSERT operation
    IF INSERTING THEN
        DBMS_OUTPUT.PUT_LINE('INSERT operation: New Salary is ' || :NEW.SALARY || '.');
    END IF;

    -- Handle DELETE operation
    IF DELETING THEN
        DBMS_OUTPUT.PUT_LINE('DELETE operation: Old Salary was ' || :OLD.SALARY || '.');
    END IF;
END;
/
```

Results | Explain | Describe | Saved SQL | History

```
Trigger created.
```

```
INSERT INTO STUDENT VALUES (7, 'Nikhil', 30, 'Pune', 5000.00);
```

Results | Explain | Describe | Saved SQL | History

```
INSERT operation: New Salary is 5000.

1 row(s) inserted.
```

```
UPDATE STUDENT SET SALARY = 3000.00 WHERE ID = 1;
```

Results | Explain | Describe | Saved SQL | History

```
UPDATE operation: Salary changed from 2000 to 3000 (Difference: 1000).

1 row(s) updated.
```

```
DELETE FROM STUDENT WHERE ID = 2;
```

Results | Explain | Describe | Saved SQL | Hi

```
DELETE operation: Old Salary was 1500.

1 row(s) deleted.
```

EXPERIMENT-17
Title: To understand the concepts of Trigger.
Objective: Students will be able to implement the concept of trigger.

1. CREATE TRIGGER SALARY_VIOLATION BEFORE INSERT OR UPDATE OF SALARY, SUPERVISOR_SSN ON EMPLOYEE of experiment 3

```
CREATE OR REPLACE TRIGGER SALARY_VIOLATION
BEFORE INSERT OR UPDATE OF SALARY, SUPER_SSN ON EMPLOYEE
FOR EACH ROW
DECLARE
    min_salary CONSTANT NUMBER := 30000; -- Minimum allowable salary
    max_salary CONSTANT NUMBER := 200000; -- Maximum allowable salary
    supervisor_count NUMBER; -- To hold the count of supervisors
BEGIN
    -- Check if the salary is within the valid range
    IF :NEW.SALARY < min_salary OR :NEW.SALARY > max_salary THEN
        RAISE_APPLICATION_ERROR(-20001, 'Salary must be between 30,000 and 200,000.');
    END IF;

    -- Check if the supervisor's SSN is valid (exists in EMPLOYEE table or is NULL)
    IF :NEW.SUPER_SSN IS NOT NULL THEN
        SELECT COUNT(*)
        INTO supervisor_count
        FROM EMPLOYEE
        WHERE SSN = :NEW.SUPER_SSN;

        IF supervisor_count = 0 THEN
            RAISE_APPLICATION_ERROR(-20002, 'Supervisor SSN must reference an existing employee or be NULL.');
        END IF;
    END IF;
END;
/
```

```
INSERT INTO EMPLOYEE (Fname, Minit, Lname, Ssn, Bdate, Address, Sex, Salary, Super_ssn, Dno)
VALUES ('Alice', 'A', 'Doe', '111223333', DATE '1990-01-01', '123 Main St', 'F', 50000, '333445555', 5);
```

Results | Explain | Describe | Saved SQL | History

```
1 row(s) inserted.
```

```
INSERT INTO EMPLOYEE (Fname, Minit, Lname, Ssn, Bdate, Address, Sex, Salary, Super_ssn, Dno)
VALUES ('Bob', 'B', 'Smith', '444556666', DATE '1992-02-02', '456 Elm St', 'M', 25000, '333445555', 5);
```

Results | Explain | Describe | Saved SQL | History

```
ORA-20001: Salary must be between 30,000 and 200,000.
ORA-06512: at "WKSP_AKSHATBALTI.SALARY_VIOLATION", line 8
ORA-04088: error during execution of trigger 'WKSP_AKSHATBALTI.SALARY_VIOLATION'
```

```
UPDATE EMPLOYEE
SET SUPER_SSN = '999999999'
WHERE Ssn = '111223333';
```

Results | Explain | Describe | Saved SQL | History

```
ORA-04091: table WKSP_AKSHATBALTI.EMPLOYEE is mutating, trigger/function may not see it
ORA-06512: at "WKSP_AKSHATBALTI.SALARY_VIOLATION", line 13
ORA-04088: error during execution of trigger 'WKSP_AKSHATBALTI.SALARY_VIOLATION'


1. UPDATE EMPLOYEE
2. SET SUPER_SSN = '999999999'
3. WHERE Ssn = '111223333';
```