Database Management System

EXPERIMENT 12 PROCEDURES AND FUNCTIONS

NAME : PRATHESHA J

ROLL NO : 241001172

1. Create a procedure to increment item price

```
CREATE OR REPLACE PROCEDURE increment_price(
p_item_id IN NUMBER,     p_increment IN
NUMBER
) IS    v_current_price
NUMBER;    null_price
EXCEPTION;
BEGIN
    SELECT actualprice INTO v_current_price
    FROM ititems
    WHERE itemid = p_item_id;

    IF v_current_price IS NULL THEN
        RAISE null_price;
    ELSE
        UPDATE ititems
        SET actualprice = actualprice + p_increment
        WHERE itemid = p_item_id;
        DBMS_OUTPUT.PUT_LINE('Price updated successfully');
    END IF;

EXCEPTION
```

```
    WHEN null_price THEN
        DBMS_OUTPUT.PUT_LINE('Error: Price is null');
    WHEN NO_DATA_FOUND THEN
        DBMS_OUTPUT.PUT_LINE('Error: Item not found');
END increment_price;
/
```

Execute the procedure:

```
EXEC increment_price(101, 500);
```

Expected Output:

```
Price updated successfully

PL/SQL procedure successfully completed.
```

2. Procedure with IN parameter

```
CREATE OR REPLACE PROCEDURE display_price(
p_item_id IN NUMBER
) IS
    v_price NUMBER;
BEGIN
    SELECT actualprice INTO v_price
    FROM ititems
    WHERE itemid = p_item_id;

    DBMS_OUTPUT.PUT_LINE('Actual price is: ' || v_price);
    IF v_price IS NULL THEN
```

```
        DBMS_OUTPUT.PUT_LINE('Price is null');
    END IF;
END display_price;
/
```

Execute:

```
EXEC display_price(103);
```

Expected Output:

```
Actual price is: 4000

PL/SQL procedure successfully completed.
```

3. Procedure with OUT parameter

```
CREATE OR REPLACE PROCEDURE check_order_status(
p_item_id IN NUMBER,     p_status OUT NUMBER
) IS
    v_order_id NUMBER;
BEGIN
    SELECT ordid INTO v_order_id
    FROM ititems
    WHERE itemid = p_item_id;

    IF v_order_id < 1000 THEN
p_status := 100;  -- Small order    ELSE
        p_status := 200;  -- Large order
    END IF;
END check_order_status;
```

```
/
```

Execute with OUT parameter:

```
DECLARE
    v_status NUMBER; BEGIN
check_order_status(101, v_status);
    DBMS_OUTPUT.PUT_LINE('Order status: ' || v_status);
END;
/
```

Expected Output:

```
Order status: 100

PL/SQL procedure successfully completed.
```

4. Procedure with IN OUT parameter

```
CREATE OR REPLACE PROCEDURE increment_value(
p_value IN OUT NUMBER
) IS BEGIN    p_value :=
p_value + 1;
END increment_value;
/
```

Execute:

```
DECLARE    v_number NUMBER :=
7; BEGIN
increment_value(v_number);
```

```
    DBMS_OUTPUT.PUT_LINE('Updated value: ' || v_number);
END;
/
```

Expected Output:

```
Updated value: 8

PL/SQL procedure successfully completed.
```

5. Function to get train fare

```
CREATE OR REPLACE FUNCTION get_train_fare(
p_train_number IN NUMBER ) RETURN NUMBER IS
v_fare ittrain.tfare%TYPE;
BEGIN
    SELECT tfare INTO v_fare
    FROM ittrain
    WHERE tno = p_train_number;

    RETURN v_fare;

EXCEPTION
    WHEN NO_DATA_FOUND THEN
        RETURN -1;  -- Indicate not found
END get_train_fare;
/
```

Execute function:

```
DECLARE    v_fare
NUMBER;
BEGIN
    v_fare := get_train_fare(1001);
    DBMS_OUTPUT.PUT_LINE('Train fare is: ' || v_fare);
END;
/
```

Expected Output:

```
Train fare is: 550


PL/SQL procedure successfully completed.
```

6. Function to calculate factorial

```
CREATE OR REPLACE FUNCTION calculate_factorial(
p_number IN NUMBER ) RETURN NUMBER IS    v_fact
NUMBER := 1;    v_counter NUMBER; BEGIN
v_counter := p_number;

    WHILE v_counter > 0 LOOP
v_fact := v_fact * v_counter;
v_counter := v_counter - 1;    END
LOOP;

    RETURN v_fact;
END calculate_factorial;
/
```

Execute factorial function:

```
DECLARE
    v_result NUMBER; BEGIN     v_result
:= calculate_factorial(5);
    DBMS_OUTPUT.PUT_LINE('Factorial of 5 is: ' || v_result);
END;
/
```

Expected Output:

```
Factorial of 5 is: 120

PL/SQL procedure successfully completed.
```

Additional Examples:

Function with exception handling

```
CREATE OR REPLACE FUNCTION get_employee_salary(
p_emp_id IN NUMBER ) RETURN NUMBER IS
    v_salary employees.salary%TYPE;
BEGIN
    SELECT salary INTO v_salary
    FROM employees
    WHERE employee_id = p_emp_id;

    RETURN v_salary;

EXCEPTION
    WHEN NO_DATA_FOUND THEN
RETURN 0;
```

```
    WHEN TOO_MANY_ROWS THEN
        RETURN -1;
END get_employee_salary;
/
```

Procedure with multiple parameters

```
CREATE OR REPLACE PROCEDURE update_employee_salary(
p_emp_id IN NUMBER,     p_new_salary IN NUMBER,
p_rows_updated OUT NUMBER
) IS
BEGIN
    UPDATE employees
    SET salary = p_new_salary
    WHERE employee_id = p_emp_id;
         p_rows_updated :=
SQL%ROWCOUNT;

    IF p_rows_updated = 0 THEN
        DBMS_OUTPUT.PUT_LINE('No employee found with ID: '
|| p_emp_id);
    ELSE
        DBMS_OUTPUT.PUT_LINE('Salary updated successfully');
    END IF;
END update_employee_salary;
/
```

Key Concepts Demonstrated:

1. **Procedures** - For performing actions, can have IN, OUT, IN OUT parameters

2. **Functions** - Must return a value, used in expressions 3.

   Parameter Modes:

   a. IN: Read-only (default)
   b. OUT: Write-only
   c. IN OUT: Read and write

4. **Exception Handling** - Using EXCEPTION block
5. **%TYPE Attribute** - For data type consistency
6. **DBMS_OUTPUT** - For displaying messages
7. **RETURN Statement** - Mandatory in functions