CSE2007 (DBMS) Lab-12

KHAN MOHD OWAIS RAZA
20BCD7138

1. Write a PL/SQL cursor

```
DECLARE
  CURSOR employees_cursor IS
    SELECT employee_id, first_name, last_name
    FROM employees
    WHERE department_id = 50;
  emp_id NUMBER(6);
  emp_first_name VARCHAR2(20);
  emp_last_name VARCHAR2(25);
BEGIN
  OPEN employees_cursor;
  LOOP
    FETCH employees_cursor INTO emp_id, emp_first_name,
emp_last_name;
    EXIT WHEN employees_cursor%NOTFOUND;
    DBMS_OUTPUT.PUT_LINE('Employee ID: ' || emp_id ||
', Name: ' || emp_first_name || ' ' || emp_last_name);
  END LOOP;
  CLOSE employees_cursor;
END;
```

2. Study of PL/SQL Conditional Statements

PL/SQL, which stands for "Procedural Language/Structured Query Language," is an extension of SQL that includes procedural constructs like conditional statements. In PL/SQL, there are three types of conditional statements: IF, CASE, and NULLIF.

IF Statements :-
The IF statement allows you to execute a block of code if a condition is true. The basic syntax for an IF statement in PL/SQL is as follows:

```
IF condition THEN
   -- code to execute if condition is true
END IF;
```

We can also include an ELSE clause to execute a block of code if the condition is false:

```
IF condition THEN
   -- code to execute if condition is true
ELSE
   -- code to execute if condition is false
END IF;
```

We can include multiple conditions using the ELSEIF clause:

```
IF condition1 THEN
   -- code to execute if condition1 is true
ELSIF condition2 THEN
   -- code to execute if condition2 is true
ELSIF condition3 THEN
   -- code to execute if condition3 is true
ELSE
   -- code to execute if all conditions are false
END IF;
```

CASE Statements :-
The CASE statement allows you to choose between multiple conditions and execute a block of code based on the selected condition. The basic syntax for a CASE statement in PL/SQL is as follows:

```
CASE expression
   WHEN value1 THEN
      -- code to execute if expression = value1
   WHEN value2 THEN
      -- code to execute if expression = value2
   WHEN value3 THEN
      -- code to execute if expression = value3
   ELSE
      -- code to execute if expression does not match
any value
END CASE;
```

We can also use the simple CASE format where the expression is repeated:

```
CASE expression
   WHEN value1 THEN
      -- code to execute if expression = value1
   WHEN value2 THEN
      -- code to execute if expression = value2
   WHEN value3 THEN
      -- code to execute if expression = value3
END CASE;
```

NULLIF Statements :-
The NULLIF statement allows you to compare two expressions and return NULL if they are equal. The basic syntax for a NULLIF statement in PL/SQL is as follows:

```
NULLIF(expression1, expression2);
```

If expression1 and expression2 are equal, then NULL is returned. Otherwise, expression1 is returned.

In summary, conditional statements in PL/SQL provide a way to control the flow of your code based on certain conditions. The IF statement allows you to execute code based on a single condition, while the CASE statement allows you to choose between multiple conditions. The NULLIF statement allows you to return NULL if two expressions are equal.

3. Study of PL/SQL Loops

a. Study of PL/SQL procedures and functions

In PL/SQL, a procedure is a named block of code that performs a specific task or set of tasks, while a function is a named block of code that returns a single value. Both procedures and functions are used to encapsulate business logic and are stored in the database for reuse.

PL/SQL Procedures:-
Procedures are created using the CREATE PROCEDURE statement and can include input and output parameters. Here is an example of a PL/SQL procedure:

```
CREATE OR REPLACE PROCEDURE update_salary
(
  p_employee_id IN employees.employee_id%TYPE,
  p_salary IN employees.salary%TYPE
)
IS
BEGIN
```

```
  UPDATE employees
  SET salary = p_salary
  WHERE employee_id = p_employee_id;
  COMMIT;
END;
```

In this example, the procedure is called "update_salary" and takes two input parameters: p_employee_id and p_salary. The procedure updates the salary of an employee with the specified employee_id and commits the transaction.

PL/SQL Functions:-
Functions are created using the CREATE FUNCTION statement and return a single value. Here is an example of a PL/SQL function:

```
CREATE OR REPLACE FUNCTION get_employee_name
(
  p_employee_id IN employees.employee_id%TYPE
)
RETURN employees.first_name%TYPE
IS
  v_employee_name employees.first_name%TYPE;
BEGIN
  SELECT first_name INTO v_employee_name
  FROM employees
  WHERE employee_id = p_employee_id;
  RETURN v_employee_name;
END;
```

In this example, the function is called "get_employee_name" and takes one input parameter: p_employee_id. The function retrieves the first name of the employee with the specified employee_id and returns it as a single value.

Procedures and functions can be called from other PL/SQL code, SQL statements, and even other programming languages. They can also be used to define custom data types and exception handling logic.

b. Study of PL/SQL cursors

In PL/SQL, a cursor is a database object that enables the traversal and manipulation of result sets returned by a SQL query. Cursors are used to process individual rows returned by a SQL statement, and they provide a way to perform database operations on a row-by-row basis. Cursors can be either implicit or explicit.

Implicit Cursors:-
Implicit cursors are created by default when you execute a SQL statement in PL/SQL. They are used to process result sets returned by SELECT, INSERT, UPDATE, DELETE, and other SQL statements. Implicit cursors are automatically opened and closed by PL/SQL, and they are defined by the SQL statement that is executed. Here is an example of using an implicit cursor to retrieve data from the employees table:

```
DECLARE
  v_employee_id employees.employee_id%TYPE;
  v_first_name employees.first_name%TYPE;
BEGIN
  SELECT employee_id, first_name INTO v_employee_id,
v_first_name
  FROM employees
  WHERE employee_id = 100;
  DBMS_OUTPUT.PUT_LINE('Employee ID: ' || v_employee_id
|| ', First Name: ' || v_first_name);
END;
```

In this example, the SELECT statement retrieves the employee_id and first_name of the employee with the employee_id of 100. The result set is stored in two variables: v_employee_id and v_first_name. The DBMS_OUTPUT.PUT_LINE statement then displays the values of these variables.

Explicit Cursors:-
Explicit cursors are created by the developer and provide more control over the result set processing than implicit cursors. Explicit cursors are defined using the

CURSOR declaration, and they must be opened, fetched, and closed explicitly. Here is an example of using an explicit cursor to retrieve data from the employees table:

```
DECLARE
  CURSOR employees_cursor IS
    SELECT employee_id, first_name, last_name
    FROM employees;
  v_employee_id employees.employee_id%TYPE;
  v_first_name employees.first_name%TYPE;
  v_last_name employees.last_name%TYPE;
BEGIN
  OPEN employees_cursor;
  LOOP
    FETCH employees_cursor INTO v_employee_id,
v_first_name, v_last_name;
    EXIT WHEN employees_cursor%NOTFOUND;
    DBMS_OUTPUT.PUT_LINE('Employee ID: ' ||
v_employee_id || ', Name: ' || v_first_name || ' ' ||
v_last_name);
  END LOOP;
  CLOSE employees_cursor;
END;
```

In this example, the CURSOR declaration defines a cursor named "employees_cursor" that retrieves the employee_id, first_name, and last_name from the employees table. The cursor is then opened using the OPEN statement, and a loop is used to fetch each row from the cursor one by one. The values of each row are stored in variables: v_employee_id, v_first_name, and v_last_name. The loop continues until all rows have been fetched, and the cursor is then closed using the CLOSE statement.

c. Write a program using PL/SQL to raise Triggers

```
CREATE OR REPLACE TRIGGER employees_trigger
AFTER INSERT OR UPDATE OR DELETE
ON employees
FOR EACH ROW
BEGIN
  DBMS_OUTPUT.PUT_LINE('A row has been inserted,
updated, or deleted from the employees table');
END;
```

d. Write a PL/SQL program to handle Exceptions

```
DECLARE
  v_num1 NUMBER := 10;
  v_num2 NUMBER := 0;
  v_result NUMBER;
BEGIN
  BEGIN
    v_result := v_num1 / v_num2;
    DBMS_OUTPUT.PUT_LINE('Result: ' || v_result);
  EXCEPTION
    WHEN ZERO_DIVIDE THEN
      DBMS_OUTPUT.PUT_LINE('Error: Division by zero');
  END;
END;
```

In this program, we declare two variables v_num1 and v_num2, and initialize v_num1 to 10 and v_num2 to 0. We then try to divide v_num1 by v_num2, which would result in a division by zero error. However, we use a nested BEGIN...EXCEPTION block to catch the exception that is raised and handle it gracefully.

The EXCEPTION section specifies which exception to catch, in this case the ZERO_DIVIDE exception. When this exception is caught, the program will execute the code within the EXCEPTION block, which in this example simply prints an error message to the console using the DBMS_OUTPUT.PUT_LINE function.