Tasks on PL/SQL Basics with Database

Task 1: Write a PL/SQL block to insert a new employee into the `employees` table.

Table: `employees(emp_id, emp_name, salary, department)`

Insert an employee with `emp_id = 101`, `emp_name = 'John Doe'`, `salary = 5000`, `department = 'IT'`.

```
SQL> DECLARE
  2  BEGIN
  3      INSERT INTO employees (emp_id, emp_name, salary, department)
  4      VALUES (101, 'John Doe', 5000, 'IT');
  5
  6      COMMIT;
  7      DBMS_OUTPUT.PUT_LINE('Employee inserted successfully.');
  8  EXCEPTION
  9      WHEN OTHERS THEN
 10          ROLLBACK;
 11          DBMS_OUTPUT.PUT_LINE('Error in column ' || DBMS_UTILITY.FORMAT_ERROR_BACKTRACE || ': ' || SQLERRM);
 12  END;
 13  /
    INSERT INTO employees (emp_id, emp_name, salary, department)
```

Task 2: Create a PL/SQL block to retrieve and display all employee names from the `employees` table.

```
SQL> DECLARE
  2      -- Cursor to fetch all employee names
  3      CURSOR emp_cursor IS
  4          SELECT emp_name FROM employees;
  5
  6      -- Variable to hold each employee name
  7      v_emp_name employees.emp_name%TYPE;
  8  BEGIN
  9      -- Open the cursor
 10      OPEN emp_cursor;
 11
 12      -- Display header
 13      DBMS_OUTPUT.PUT_LINE('Employee Names:');
 14      DBMS_OUTPUT.PUT_LINE('---------------');
 15
 16      -- Fetch and display each employee name
 17      LOOP
 18          FETCH emp_cursor INTO v_emp_name;
 19          EXIT WHEN emp_cursor%NOTFOUND;  -- Exit loop when no more rows
 20
 21          DBMS_OUTPUT.PUT_LINE(v_emp_name);
 22      END LOOP;
 23
 24      -- Close the cursor
 25      CLOSE emp_cursor;
 26  EXCEPTION
 27      WHEN OTHERS THEN
 28          -- Error handling
 29          IF emp_cursor%ISOPEN THEN
 30              CLOSE emp_cursor;
 31          END IF;
 32          DBMS_OUTPUT.PUT_LINE('Error retrieving employee names: ' || SQLERRM);
 33  END;
 34  /
```

Task 3: Write a PL/SQL block to update the salary of an employee whose `emp_id = 101` by

increasing it by **10%**.

```
SQL> DECLARE
  2      v_emp_id NUMBER := 101;
  3      v_new_salary NUMBER;
  4      v_old_salary NUMBER;
  5  BEGIN
  6      -- Get current salary
  7      SELECT salary INTO v_old_salary
  8      FROM employees
  9      WHERE emp_id = v_emp_id;
 10
 11      -- Calculate new salary (10% increase)
 12      v_new_salary := v_old_salary * 1.10;
 13
 14      -- Update the salary
 15      UPDATE employees
 16      SET salary = v_new_salary
 17      WHERE emp_id = v_emp_id;
 18
 19      COMMIT;
 20
 21      -- Display results
 22      DBMS_OUTPUT.PUT_LINE('Salary updated for Employee ID: ' || v_emp_id);
 23      DBMS_OUTPUT.PUT_LINE('Old Salary: ' || v_old_salary);
 24      DBMS_OUTPUT.PUT_LINE('New Salary: ' || v_new_salary);
 25
 26  EXCEPTION
 27      WHEN NO_DATA_FOUND THEN
 28          DBMS_OUTPUT.PUT_LINE('Employee with ID ' || v_emp_id || ' not found.');
 29      WHEN OTHERS THEN
 30          ROLLBACK;
 31          DBMS_OUTPUT.PUT_LINE('Error updating salary: ' || SQLERRM);
 32  END;
 33  /
```

Task 4: Create a PL/SQL block to delete an employee whose `emp_id = 105`.

```
SQL> BEGIN
  2      DELETE FROM employees
  3      WHERE emp_id = 105;
  4
  5      COMMIT;
  6
  7      DBMS_OUTPUT.PUT_LINE('Employee with emp_id 105 deleted successfully.');
  8  EXCEPTION
  9      WHEN OTHERS THEN
 10          DBMS_OUTPUT.PUT_LINE('Error deleting employee: ' || SQLERRM);
 11  END;
 12  /
    DELETE FROM employees
```

Task 5: Display the count of employees in the `employees` table.

```
SQL> DECLARE
  2      v_emp_count NUMBER;
  3  BEGIN
  4      -- Retrieve the count of employees
  5      SELECT COUNT(*) INTO v_emp_count FROM employees;
  6
  7      -- Display the count
  8      DBMS_OUTPUT.PUT_LINE('Total number of employees: ' || v_emp_count);
  9  END;
 10  /
    SELECT COUNT(*) INTO v_emp_count FROM employees;
```

Tasks on Conditional Statements with Database

Task 6: Write a PL/SQL block that checks if an employee's salary is above **5000**. If yes, print `"High Salary"`; otherwise, print `"Low Salary"`.

```
SQL> DECLARE
  2      v_salary NUMBER;
  3      v_emp_id NUMBER := 101; -- Change this to the employee ID you want to check
  4  BEGIN
  5      -- Retrieve the employee's salary
  6      SELECT salary INTO v_salary
  7      FROM employees
  8      WHERE emp_id = v_emp_id;
  9
 10      -- Check salary and print message
 11      IF v_salary > 5000 THEN
 12          DBMS_OUTPUT.PUT_LINE('High Salary');
 13      ELSE
 14          DBMS_OUTPUT.PUT_LINE('Low Salary');
 15      END IF;
 16  EXCEPTION
 17      WHEN NO_DATA_FOUND THEN
 18          DBMS_OUTPUT.PUT_LINE('Employee not found.');
 19      WHEN OTHERS THEN
 20          DBMS_OUTPUT.PUT_LINE('Error: ' || SQLERRM);
 21  END;
 22  /
```

Task 7: Fetch the department of an employee based on `emp_id` and print:

- `"IT Department"` if in **IT**,
- `"HR Department"` if in **HR**,
- `"Other Department"` otherwise.

```
SQL> DECLARE
  2      v_department employees.department%TYPE;
  3      v_emp_id NUMBER := 101; -- Change this to the employee ID you want to check
  4  BEGIN
  5      -- Retrieve the employee's department
  6      SELECT department INTO v_department
  7      FROM employees
  8      WHERE emp_id = v_emp_id;
  9
 10      -- Check department and print message
 11      IF v_department = 'IT' THEN
 12          DBMS_OUTPUT.PUT_LINE('IT Department');
 13      ELSIF v_department = 'HR' THEN
 14          DBMS_OUTPUT.PUT_LINE('HR Department');
 15      ELSE
 16          DBMS_OUTPUT.PUT_LINE('Other Department');
 17      END IF;
 18  EXCEPTION
 19      WHEN NO_DATA_FOUND THEN
 20          DBMS_OUTPUT.PUT_LINE('Employee not found.');
 21      WHEN OTHERS THEN
 22          DBMS_OUTPUT.PUT_LINE('Error: ' || SQLERRM);
 23  END;
 24  /
```

Task 8: Use a `CASE` statement to categorize employees based on salary:

- **Above 8000 → "Senior Level"**
- **5000-8000 → "Mid Level"**
- **Below 5000 → "Junior Level"**

```
SQL> DECLARE
  2      v_salary employees.salary%TYPE;
  3      v_emp_id NUMBER := 101; -- Change this to the employee ID you want to check
  4      v_category VARCHAR2(20);
  5  BEGIN
  6      -- Retrieve the employee's salary
  7      SELECT salary INTO v_salary
  8      FROM employees
  9      WHERE emp_id = v_emp_id;
 10
 11      -- Categorize employee based on salary
 12      v_category := CASE
 13                      WHEN v_salary > 8000 THEN 'Senior Level'
 14                      WHEN v_salary BETWEEN 5000 AND 8000 THEN 'Mid Level'
 15                      ELSE 'Junior Level'
 16                    END;
 17
 18      -- Display the category
 19      DBMS_OUTPUT.PUT_LINE('Employee Salary Category: ' || v_category);
 20  EXCEPTION
 21      WHEN NO_DATA_FOUND THEN
 22          DBMS_OUTPUT.PUT_LINE('Employee not found.');
 23      WHEN OTHERS THEN
 24          DBMS_OUTPUT.PUT_LINE('Error: ' || SQLERRM);
 25  END;
 26  /
```

Task 9: If an employee's department is **Sales**, increase their salary by **5%**.

Task 10: Check if an employee with `emp_id = 110` exists. If not, insert a new record.

Tasks on Loops with Database

Task 11: Use a **FOR LOOP** to print all employees' names from the `employees` table.

Task 12: Write a **LOOP** to insert **5 new employees** into the `employees` table.

Task 13: Use a **WHILE LOOP** to increase the salary of all employees earning less than **4000** by **20%**.

Task 14: Create a **FOR LOOP** that prints the first **3 departments** from the `departments` table.

Task 15: Write a **LOOP** to delete employees who have not updated their records in the last **5 years** (assuming there's a `last_updated` column).

Task 16: Use a **LOOP** to find the employee with the highest salary in the `employees` table.

Task 17: Fetch and display all employees in a specific department using a **WHILE LOOP**.

Task 18: Write a **LOOP** to insert **10 new customers** into a `customers` table.

Task 19: Use a **FOR LOOP** to display the top **5 highest-paid employees** from the `employees` table.

Task 20: Write a **LOOP** to find and delete duplicate employee records in the `employees` table.