By Maha Rana

CS405 Finalterm Subjective Solved

Short Questions:

1. **Table Record:**
   - A record is a collection of **related data items** stored as a **single unit**.
   - Example:

```
DECLARE
   emp_record emp%ROWTYPE;
BEGIN
   SELECT * INTO emp_record FROM emp WHERE empno = 1001;
END;
```

2. **Exception Types in PL/SQL:**

- **Predefined Exceptions:** Raised automatically (e.g., `NO_DATA_FOUND`).
- **User-Defined Exceptions:** Declared manually using `EXCEPTION`.
- **Example:**

```
DECLARE
   salary_error EXCEPTION;
BEGIN
   IF salary < 0 THEN
      RAISE salary_error;
   END IF;
EXCEPTION
   WHEN salary_error THEN
      DBMS_OUTPUT.PUT_LINE('Invalid Salary');
END;
```

By Maha Rana

1. **Continue When Statement Explanation:**
   - This skips the remaining code in the loop's **current iteration** when a condition is met.
   - Example:

```
BEGIN
    FOR i IN 1..10 LOOP
        CONTINUE WHEN i = 5;
        DBMS_OUTPUT.PUT_LINE(i);
    END LOOP;
END;
```

2. **Records in PL/SQL:**
   - Similar to **structs in C**, they store multiple fields in a single variable.
   - Example:

```
DECLARE
    emp_rec emp%ROWTYPE;
BEGIN
    SELECT * INTO emp_rec FROM emp WHERE empno
= 7369;
    DBMS_OUTPUT.PUT_LINE(emp_rec.ename || ' - '
|| emp_rec.sal);
END;
```

Long Questions:

1. **Nested Loop Syntax:**

```
BEGIN
    FOR i IN 1..3 LOOP
        FOR j IN 1..2 LOOP
            DBMS_OUTPUT.PUT_LINE('i=' || i || ',
j=' || j);
```

```
        END LOOP;
    END LOOP;
END;
```

2. **PL/SQL Procedure Code:**

```
CREATE OR REPLACE PROCEDURE get_emp_details(emp_id
NUMBER) IS
    emp_name emp.ename%TYPE;
BEGIN
    SELECT ename INTO emp_name FROM emp WHERE empno
= emp_id;
    DBMS_OUTPUT.PUT_LINE('Employee Name: ' ||
emp_name);
END;
```

3. **User-Defined Exception Syntax:**

```plsql
DECLARE
    invalid_salary EXCEPTION;
BEGIN
    IF salary < 0 THEN
        RAISE invalid_salary;
    END IF;
EXCEPTION
    WHEN invalid_salary THEN
        DBMS_OUTPUT.PUT_LINE('Invalid Salary');
END;
```

By Maha Rana

Nested Loop Code Example:

```
pl

DECLARE
    i NUMBER := 1;
BEGIN
    WHILE i <= 3 LOOP
        FOR j IN 1..2 LOOP
            DBMS_OUTPUT.PUT_LINE('i=' || i || ', j=' || j);
        END LOOP;
        i := i + 1;
    END LOOP;
END;
```

Short Questions Solutions:

1. Table Record:

A **record** is a collection of **related data items** that can be stored in a single unit. It is similar to a **struct** in C/C++.

Example in PL/SQL:

```
plsql

DECLARE
   emp_record emp%ROWTYPE;  -- Record variable based on table structure
BEGIN
   SELECT * INTO emp_record FROM emp WHERE empno = 1001;
   DBMS_OUTPUT.PUT_LINE('Employee Name: ' || emp_record.ename);
END;
```

By Maha Rana

Purpose: Records allow us to store and manipulate **multiple columns** of a table within a single variable.

2. Exception Types in PL/SQL

PL/SQL **Exceptions** handle errors that occur during execution. There are **three types**:

1. **Predefined Exceptions** – Raised automatically by Oracle.
    - Example: `NO_DATA_FOUND`, `TOO_MANY_ROWS`
2. **User-Defined Exceptions** – Manually declared and raised in PL/SQL.
    - Example:

```plsql
DECLARE
    salary_error EXCEPTION;
BEGIN
    IF salary < 0 THEN
        RAISE salary_error;
    END IF;
EXCEPTION
    WHEN salary_error THEN
        DBMS_OUTPUT.PUT_LINE('Invalid Salary');
END;
```

1. **Named System Exceptions** – Oracle predefined exceptions that can be handled explicitly.
    - Example: `WHEN OTHERS THEN ...`

4. Continue When Statement

By Maha Rana

The **`CONTINUE WHEN`** statement in PL/SQL is used inside loops to **skip** certain iterations when a condition is met.

Example:

```
BEGIN
   FOR i IN 1..5 LOOP
      CONTINUE WHEN i = 3;  -- Skips iteration when i =
3
      DBMS_OUTPUT.PUT_LINE('Iteration: ' || i);
   END LOOP;
END;
```

 Output:

```
Iteration: 1
Iteration: 2
Iteration: 4
Iteration: 5
```

Notice that **Iteration 3 is skipped**.

---

5. Records in PL/SQL

A **PL/SQL record** is a collection of **fields (columns)** stored in a **single variable**.

Example:

plsql

```
DECLARE
   emp_rec emp%ROWTYPE;  -- Create a record variable based on EMP table
BEGIN
   SELECT * INTO emp_rec FROM emp WHERE empno = 7369;
   DBMS_OUTPUT.PUT_LINE(emp_rec.ename || ' - ' || emp_rec.sal);
END;
```

By Maha Rana

Purpose: Allows working with **multiple related columns** easily.

---

6. Loop Output Prediction

Since the **loop code** is missing, here's a **generic loop example** and its expected output:

```plsql
DECLARE
   i NUMBER := 1;
BEGIN
   WHILE i <= 5 LOOP
      DBMS_OUTPUT.PUT_LINE('i = ' || i);
      i := i + 1;
   END LOOP;
END;
```

□ Expected Output:

```
i = 1
i = 2
i = 3
i = 4
i = 5
```

Long Questions Solutions:

1. Nested Loop Syntax

By Maha Rana

A **nested loop** is when one loop is placed **inside another loop**.

Syntax:

Syntax:

```plsql
BEGIN
   FOR i IN 1..3 LOOP
      FOR j IN 1..2 LOOP
         DBMS_OUTPUT.PUT_LINE('i=' || i || ', j=' || j);
      END LOOP;
   END LOOP;
END;
```

This **executes the inner loop for each iteration of the outer loop.**

2. Procedure in PL/SQL

A **procedure** is a named PL/SQL block that can be called multiple times.

Example:

```plsql
CREATE OR REPLACE PROCEDURE get_emp_details(emp_id NUMBER) IS
   emp_name emp.ename%TYPE;
BEGIN
   SELECT ename INTO emp_name FROM emp WHERE empno = emp_id;
   DBMS_OUTPUT.PUT_LINE('Employee Name: ' || emp_name);
END;
```

Usage:

By Maha Rana

```
pl

BEGIN
   get_emp_details(1001);
END;
```

3. User-Defined Exception Syntax

PL/SQL allows developers to create **custom exceptions**.

Syntax:

```
plsql

DECLARE
   salary_error EXCEPTION;  -- Declare exception
BEGIN
   IF salary < 0 THEN
      RAISE salary_error;  -- Raise exception if salary is negative
   END IF;
EXCEPTION
   WHEN salary_error THEN
      DBMS_OUTPUT.PUT_LINE('Invalid Salary');
END;
```

Purpose: Allows **custom error handling** in PL/SQL.

4. Nested Loop Code Example

Here is a **nested loop example** that prints **all combinations of i and j**.

Code:

By Maha Rana

```plsql
DECLARE
   i NUMBER := 1;
BEGIN
   WHILE i <= 3 LOOP
      FOR j IN 1..2 LOOP
         DBMS_OUTPUT.PUT_LINE('i=' || i || ', j=' || j);
      END LOOP;
      i := i + 1;
   END LOOP;
END;
```

Output:

```ini
i=1, j=1
i=1, j=2
i=2, j=1
i=2, j=2
i=3, j=1
i=3, j=2
```

This shows how **nested loops** work in **PL/SQL**.

MCQs Topics Covered:

- **CASE Statement**
- **Cursor Attributes**
- **Exception Types**
- **Loops (FOR, WHILE, SIMPLE, NESTED)**
- **Triggers**
- **Collections in PL/SQL**

By Maha Rana

 Tip: Revise **Cursor Attributes (`%FOUND`, `%NOTFOUND`, `%ROWCOUNT`, `%ISOPEN`)** and **Types of Exceptions (Predefined, User-Defined, Named System)**.

Short Questions Solutions:

1. What is a CASE statement and how is it executed?

A **CASE statement** in PL/SQL is an alternative to **IF-THEN-ELSE** statements. It allows multiple conditional checks in a structured way.

Syntax:

```
CASE selector
    WHEN expression1 THEN result1;
    WHEN expression2 THEN result2;
    ELSE default_result;
END CASE;
```

Example Execution:

```
DECLARE
    grade CHAR(1) := 'A';
BEGIN
    CASE grade
        WHEN 'A' THEN
DBMS_OUTPUT.PUT_LINE('Excellent');
        WHEN 'B' THEN DBMS_OUTPUT.PUT_LINE('Good');
        ELSE DBMS_OUTPUT.PUT_LINE('Needs Improvement');
    END CASE;
END;
```

 Execution: If `grade = 'A'`, it prints `Excellent`.

---

2. What is a User-Defined Exception?

By Maha Rana

A **User-Defined Exception** is an error condition created by the programmer to handle specific business logic errors in PL/SQL.

Syntax:

```
DECLARE
    my_exception EXCEPTION;  -- Declare Exception
BEGIN
    RAISE my_exception;  -- Raise Exception
EXCEPTION
    WHEN my_exception THEN
        DBMS_OUTPUT.PUT_LINE('Custom Exception
Triggered');
END;
```

 Purpose: It helps in handling **specific errors** not covered by Oracle's predefined exceptions.

---

3. What is a Collection? Name any two types of PL/SQL tables.

A **collection** is a **data structure** in PL/SQL that stores **multiple elements** of the same type (like arrays in other languages).

Types of PL/SQL Collections:

1. **Associative Arrays (Index-by Tables)**
    - Key-value pairs like a dictionary.
2. **Nested Tables**
    - Dynamic tables that can store a collection of data.

Example of a Collection:

```
DECLARE
    TYPE emp_table IS TABLE OF VARCHAR2(50);
    emp_names emp_table := emp_table('Ali', 'Ayesha',
'Bilal');
```

```
BEGIN
    DBMS_OUTPUT.PUT_LINE(emp_names(1)); -- Output: Ali
END;
```

---

4. What is a Trigger? Name any two types of Triggers.

A **trigger** is a **PL/SQL block** that executes automatically when a specific event occurs (like `INSERT`, `UPDATE`, `DELETE`).

Types of Triggers:

1. **Row-Level Trigger** – Executes for each row affected by DML.
2. **Statement-Level Trigger** – Executes once per SQL statement.

Example of a Trigger:

```
CREATE OR REPLACE TRIGGER emp_salary_check
BEFORE INSERT OR UPDATE ON employees
FOR EACH ROW
BEGIN
    IF :NEW.salary < 0 THEN
        RAISE_APPLICATION_ERROR(-20001, 'Salary cannot
be negative!');
    END IF;
END;
```

---

5. `RAISE_APPLICATION_ERROR` Syntax Completion

The `RAISE_APPLICATION_ERROR` function is used to **display custom error messages** in PL/SQL.

Complete Syntax:

```
RAISE_APPLICATION_ERROR(error_number, error_message);
```

 Example with Corrected Syntax:

```
RAISE_APPLICATION_ERROR(-20001, 'Invalid Input
Provided');
```

 Tip:

- The error number **must be between `-20000` and `-20999`**.
- Used inside `EXCEPTION` blocks or **triggers**.

Long Questions Solutions:

2. Exception Types and Their Explanation

PL/SQL **exceptions** are used to handle errors during runtime. There are **three main types**:

1. Predefined Exceptions (Built-in by Oracle)

- `NO_DATA_FOUND` – No rows found in `SELECT INTO`.
- `TOO_MANY_ROWS` – When `SELECT` returns **more than one row**.
- `ZERO_DIVIDE` – When trying to divide by **zero**.
- Example:

```
BEGIN
    SELECT sal INTO v_sal FROM employees WHERE
emp_id = 100;

EXCEPTION
    WHEN NO_DATA_FOUND THEN
        DBMS_OUTPUT.PUT_LINE('No employee found!');
END;
```

3. User-Defined Exceptions (Manually Declared)

- Example:

```
DECLARE
    salary_error EXCEPTION;



BEGIN
    IF salary < 0 THEN
        RAISE salary_error;
    END IF;
EXCEPTION
    WHEN salary_error THEN
        DBMS_OUTPUT.PUT_LINE('Invalid Salary');
END;
```

3. Named System Exceptions (Explicitly Handled)

- Example:

```
EXCEPTION
    WHEN DUP_VAL_ON_INDEX THEN
        DBMS_OUTPUT.PUT_LINE('Duplicate value
error!');
```

Short Questions:

1. Difference Between CASE and Searched CASE Statement

By Maha Rana

Feature	CASE Statement	Searched CASE Statement
Condition Type	Uses a **selector** (a single variable or expression)	Uses **boolean conditions** directly in `WHEN`
Syntax	`CASE var_name WHEN value THEN ... END CASE;`	`CASE WHEN condition THEN ... END CASE;`
Example	`plsql CASE grade WHEN 'A' THEN 'Excellent'; ELSE 'Fail'; END CASE;`	`plsql CASE WHEN marks > 50 THEN 'Pass'; ELSE 'Fail'; END CASE;`

✅ **CASE** is used when comparing a variable against values.
✅ **Searched CASE** is used when conditions involve comparisons like `>`, `<`, etc.

3. Error vs Exception

Feature	Error	Exception
Definition	A system-generated issue that **halts execution**	A runtime event that **can be handled**
Cause	Can occur at **compile-time** or **runtime**	Occurs at **runtime** only
Handling	Cannot be handled in PL/SQL	Handled using `EXCEPTION` block
Example	**Syntax Error** (e.g., missing `;`)	**NO_DATA_FOUND**, **ZERO_DIVIDE**, **User-Defined Errors**

✅ **Errors** stop execution immediately, while **exceptions** allow handling through an `EXCEPTION` block.

3. Arrays in PL/SQL (Collections)

PL/SQL does **not** support standard arrays like C/C++. Instead, it provides **Collections**:

✅Types of Arrays (Collections) in PL/SQL:

1. **Associative Arrays (Index-By Tables)** – Key-value pairs (like a dictionary).
2. **Nested Tables** – Expandable collections stored in database tables.

By Maha Rana

Example of Associative Array:

```
pl

DECLARE
   TYPE emp_array IS TABLE OF VARCHAR2(50) INDEX BY PLS_INTEGER;
   emp_names emp_array;
BEGIN
   emp_names(1) := 'Ali';
   emp_names(2) := 'Ayesha';
   DBMS_OUTPUT.PUT_LINE(emp_names(1)); -- Output: Ali
END;
```

3. Difference Between RAISE and RAISE_APPLICATION_ERROR

Feature	`RAISE`	`RAISE_APPLICATION_ERROR`
Purpose	Used to **re-raise** an existing exception	Used to **generate a new error message**
Syntax	`RAISE exception_name;`	`RAISE_APPLICATION_ERROR(-20001, 'Error Message');`
Usage	Inside `EXCEPTION` block	Inside `BEGIN` or `EXCEPTION` block
Example	`plsql RAISE salary_error;`	`plsql RAISE_APPLICATION_ERROR(-20001, 'Invalid Input');`

✅ `RAISE` is for **existing exceptions**, while `RAISE_APPLICATION_ERROR` is for **custom error messages**.

Long Questions:

1. User-Defined Exception Syntax

✅PL/SQL Code for User-Defined Exception

By Maha Rana

```plsql
DECLARE
   invalid_salary EXCEPTION;  -- Declaring exception
BEGIN
   IF salary < 0 THEN
      RAISE invalid_salary;  -- Raising the exception
   END IF;
EXCEPTION
   WHEN invalid_salary THEN
      DBMS_OUTPUT.PUT_LINE('Error: Salary cannot be negative!');
END;
```

 Purpose: Used for **custom errors** in PL/SQL.

---

2. PL/SQL Procedure to Print "Good Morning"

✅Creating and Calling a Procedure

```plsql
CREATE OR REPLACE PROCEDURE greet_message IS
BEGIN
   DBMS_OUTPUT.PUT_LINE('Good Morning');
END;
```

To call the procedure:

```
BEGIN
   greet_message;
END;
```

 Output:

By Maha Rana

Good Morning

3. PL/SQL Code for First 10 Natural Numbers

✅Using Loops

```plsql
DECLARE
   i NUMBER := 1;
BEGIN
   WHILE i <= 10 LOOP
      DBMS_OUTPUT.PUT_LINE(i);
      i := i + 1;
   END LOOP;
END;
```

□ Output:

```
1
2
3
4
5
6
7
8
9
10
```

---

4. PL/SQL Code for Factorial Calculation

✅Factorial of First 10 Natural Numbers

By Maha Rana

```plsql
DECLARE
   n NUMBER := 5; -- Change for different numbers
   fact NUMBER := 1;
   i NUMBER;
BEGIN
   FOR i IN 1..n LOOP
      fact := fact * i;
   END LOOP;
   DBMS_OUTPUT.PUT_LINE('Factorial of ' || n || ' is: ' || fact);
END;
```

 Output for `n = 5`:

```
Factorial of 5 is: 120
```

Factorial is calculated using `n! = n * (n-1) * (n-2) ... * 1`.

Short Questions:

1. Syntax of CASE Statement

The **CASE statement** in PL/SQL is used as an alternative to **IF-THEN-ELSE** for conditional execution.

✅**Basic Syntax:**

```
CASE variable_name
    WHEN value1 THEN statement1;
    WHEN value2 THEN statement2;
    ELSE default_statement;
END CASE;
```

✅**Example:**

By Maha Rana

```
DECLARE
   grade CHAR(1) := 'A';
BEGIN
   CASE grade
      WHEN 'A' THEN DBMS_OUTPUT.PUT_LINE('Excellent');
      WHEN 'B' THEN DBMS_OUTPUT.PUT_LINE('Good');
      ELSE DBMS_OUTPUT.PUT_LINE('Needs Improvement');
   END CASE;
END;
```

 Execution: If `grade = 'A'`, it prints `"Excellent"`.

---

2. What is a Searched CASE Statement? How does it provide flexibility?

A **Searched CASE Statement** allows more flexible **comparison conditions** using Boolean expressions instead of direct value matching.

Feature	CASE Statement	Searched CASE Statement
Condition Type	Uses a **selector variable**	Uses **Boolean conditions** directly
Usage	`WHEN value THEN result;`	`WHEN condition THEN result;`
Flexibility	**Limited** to direct value comparison	**More flexible** with `<, >, BETWEEN, LIKE`

✅Syntax of Searched CASE Statement

```
CASE
   WHEN condition1 THEN statement1;
   WHEN condition2 THEN statement2;
   ELSE default_statement;
END CASE;
```

✅Example

```
DECLARE
   marks NUMBER := 75;
BEGIN
   CASE
```

```
      WHEN marks >= 80 THEN
DBMS_OUTPUT.PUT_LINE('Grade: A');
      WHEN marks >= 60 THEN
DBMS_OUTPUT.PUT_LINE('Grade: B');
      ELSE DBMS_OUTPUT.PUT_LINE('Fail');
   END CASE;
END;
```

 Flexibility: Unlike normal `CASE`, **this works with comparisons like >, <, `BETWEEN`**.

---

3. How Many Types of Exceptions? Write Their Names.

PL/SQL exceptions are divided into **three types**:

✅1. Predefined Exceptions (Built-in by Oracle)

- **`NO_DATA_FOUND`** → Raised when a `SELECT INTO` returns no data.
- **`ZERO_DIVIDE`** → Division by zero error.
- **`TOO_MANY_ROWS`** → `SELECT INTO` returns more than one row.

✅2. User-Defined Exceptions (Manually Declared by the Programmer)

- Declared using `EXCEPTION` keyword.
- Example:

```
DECLARE
   invalid_salary EXCEPTION;
BEGIN
   IF salary < 0 THEN
      RAISE invalid_salary;
   END IF;
EXCEPTION

   WHEN invalid_salary THEN
      DBMS_OUTPUT.PUT_LINE('Salary cannot be
negative!');
END;
```

By Maha Rana

✅3. Named System Exceptions (Explicitly Declared by Oracle)

- **Handled using `WHEN OTHERS` clause.**
- Example:

```
EXCEPTION
   WHEN DUP_VAL_ON_INDEX THEN
      DBMS_OUTPUT.PUT_LINE('Duplicate value
error!');
```

---

4. Loop Execution Count in Given Code

here is a **generic example of loop execution count**:

```
DECLARE
   count_var NUMBER := 0;
BEGIN
   FOR i IN 1..5 LOOP
      count_var := count_var + 1;
      DBMS_OUTPUT.PUT_LINE('Iteration: ' || count_var);
   END LOOP;
END;
```

 Output:

```
Iteration: 1
Iteration: 2
Iteration: 3
Iteration: 4
Iteration: 5
```

✅**Tip:** Count is the **number of times the loop runs before termination.**

5. Two Important Points About UDR (User-Defined Records)

By Maha Rana

User-Defined Records (UDR) in PL/SQL allow **multiple fields** to be stored in a single variable.

✅Two Key Points:

1. **Custom Data Structure**
   - UDR allows defining a record structure that **does not exist in the database**.
2. **Multiple Data Types in One Variable**
   - A single UDR can hold different data types like `NUMBER`, `VARCHAR2`, `DATE`.

✅Example of UDR:

```
DECLARE
   TYPE emp_record IS RECORD (
      emp_name VARCHAR2(50),
      emp_salary NUMBER
   );
   employee emp_record;
BEGIN
   employee.emp_name := 'Ali';
   employee.emp_salary := 50000;

   DBMS_OUTPUT.PUT_LINE(employee.emp_name || ' earns '
|| employee.emp_salary);
END;
```

✅Output:

```
Ali earns 50000
```

---

6. When is `COUNT` Used in PL/SQL Tables?

By Maha Rana

The `COUNT` function is used in **PL/SQL tables** (collections) to get the **total number of elements**.

✅Example of COUNT in PL/SQL Tables

```
DECLARE
   TYPE emp_table IS TABLE OF VARCHAR2(50) INDEX BY
PLS_INTEGER;
   emp_names emp_table;
BEGIN
   emp_names(1) := 'Ali';
   emp_names(2) := 'Ayesha';
   DBMS_OUTPUT.PUT_LINE('Total Employees: ' ||
emp_names.COUNT);
END;
```

✅Output:

```
Total Employees: 2
```

✅**Tip:** `COUNT` is useful for checking how many elements are stored.

Long Questions

1. Explanation of Package and Its Parts

A **PL/SQL package** is a **collection of related procedures, functions, and variables** stored together.

✅Parts of a Package:

1. **Package Specification** → Declares procedures/functions (like an interface).
2. **Package Body** → Defines the actual implementation of procedures/functions.

✅Example of Package Specification

By Maha Rana

```
CREATE OR REPLACE PACKAGE employee_pkg AS
   PROCEDURE display_emp(emp_id NUMBER);
END employee_pkg;
```

✅Example of Package Body

```
CREATE OR REPLACE PACKAGE BODY employee_pkg AS
   PROCEDURE display_emp(emp_id NUMBER) IS
   BEGIN
      DBMS_OUTPUT.PUT_LINE('Employee ID: ' || emp_id);
   END display_emp;
END employee_pkg;
```

□ To Call a Procedure from a Package:

```
BEGIN
   employee_pkg.display_emp(101);
END;
```

✅Advantages of Packages:

- **Encapsulation** (Groups related functions together)
- **Performance Improvement** (Loads only once in memory

Short Questions:

1. Syntax of IF-THEN-ELSE Statement

The `IF-THEN-ELSE` statement in PL/SQL is used for **conditional execution**.

✅Syntax:

```
IF condition THEN
   -- Execute this block if condition is TRUE
ELSE
   -- Execute this block if condition is FALSE
```

```
END IF;
```

✅Example:

```
DECLARE
   salary NUMBER := 5000;
BEGIN
   IF salary > 4000 THEN
      DBMS_OUTPUT.PUT_LINE('High Salary');
   ELSE
      DBMS_OUTPUT.PUT_LINE('Low Salary');
   END IF;
END;
```

 Output (if salary = 5000):

```
High Salary
```

---

2. Types of Records in PL/SQL

PL/SQL supports **two types of records**

1. **Table-Based Record (`%ROWTYPE`)**
   - Uses a table structure as a record.
   - Example:

     ```
     DECLARE
        emp_rec emp%ROWTYPE;
     BEGIN
        SELECT * INTO emp_rec FROM emp WHERE empno =
     1001;
        DBMS_OUTPUT.PUT_LINE(emp_rec.ename || '
     earns ' || emp_rec.sal);
     END;
     ```

2. **User-Defined Record (`TYPE`)**
   - Created manually to store multiple values.
   - Example

```
DECLARE
   TYPE emp_record IS RECORD (
      emp_name VARCHAR2(50),
      emp_salary NUMBER
   );
   employee emp_record;
BEGIN
   employee.emp_name := 'Ali';
   employee.emp_salary := 50000;
   DBMS_OUTPUT.PUT_LINE(employee.emp_name || '
earns ' || employee.emp_salary);

END;
```

%ROWTYPE is used when working with database tables, while TYPE RECORD is useful for custom structures.

3. **Identifying the Error in Given Code**

   ✕Code with Error:

```
DECLARE
   salary NUMBER;
BEGIN
   salary := 'Five Thousand'; -- ✕Error: Assigning
string to NUMBER datatype
END;
```

   □ Common errors:
   ✕Mismatched data types
   ✕Using SELECT without INTO in PL/SQL
   ✕Forgetting END IF; in conditional statements

4. **Why is COUNT Attribute Used in PL/SQL Tables?**

By Maha Rana

The `COUNT` attribute is used in **PL/SQL collections** to **get the number of elements** stored.

✅**Example:**

```plsql
DECLARE
   TYPE emp_table IS TABLE OF VARCHAR2(50) INDEX BY PLS_INTEGER;
   emp_names emp_table;
BEGIN
   emp_names(1) := 'Ali';
   emp_names(2) := 'Ayesha';
   DBMS_OUTPUT.PUT_LINE('Total Employees: ' || emp_names.COUNT);
END;
```

👉 **Output:**

```yaml
Total Employees: 2
```

COUNT is used for **checking how many elements are stored**.

6. Definition of User-Defined Exception

A **User-Defined Exception** is a **custom error handler** in PL/SQL.

By Maha Rana

✅Definition:

A **user-defined exception** is an error condition explicitly declared and raised by the programmer to **handle specific business logic errors**.

✅Example:

```
DECLARE
   invalid_salary EXCEPTION;
BEGIN
   IF salary < 0 THEN
      RAISE invalid_salary;
   END IF;
EXCEPTION
   WHEN invalid_salary THEN
      DBMS_OUTPUT.PUT_LINE('Error: Salary cannot be negative!');
END;
```

Long Questions:

1. PL/SQL Procedure to Calculate Square of a Number

✅Creating and Calling a Procedure

```plsql
CREATE OR REPLACE PROCEDURE square_number(num IN NUMBER) IS
   result NUMBER;
BEGIN
   result := num * num;
   DBMS_OUTPUT.PUT_LINE('Square of ' || num || ' is: ' || result);
END;
```

By Maha Rana

👉 To Call the Procedure:

```
pl

BEGIN
   square_number(5);
END;
```

👉 Output for `5`:

```
csharp

Square of 5 is: 25
```

2. PL/SQL Procedure to Print Student Name by ID (Handle Exception if Not Found)

✅Procedure to Fetch Name Using Student ID

```
pl

CREATE OR REPLACE PROCEDURE get_student_name(student_id IN NUMBER) IS
   student_name VARCHAR2(50);
BEGIN
   SELECT name INTO student_name FROM students WHERE id = student_id;
   DBMS_OUTPUT.PUT_LINE('Student Name: ' || student_name);
EXCEPTION
   WHEN NO_DATA_FOUND THEN
      DBMS_OUTPUT.PUT_LINE('Error: Student ID not found!');
END;
```

By Maha Rana

👉 To Call the Procedure:

```plsql
BEGIN
   get_student_name(101);
END;
```

👉 If ID `101` exists, Output:

```yaml
Student Name: Ali
```

👉 If ID `101` exists, Output:

```yaml
Student Name: Ali
```

👉 If ID does not exist, Output:

```javascript
Error: Student ID not found!
```

3. PL/SQL Case Statement to Print Grades

✅Using CASE to Assign Grades

By Maha Rana

```plsql
DECLARE
   marks NUMBER := 75;
BEGIN
   CASE
      WHEN marks >= 80 THEN DBMS_OUTPUT.PUT_LINE('Grade: A');
      WHEN marks >= 60 THEN DBMS_OUTPUT.PUT_LINE('Grade: B');
      ELSE DBMS_OUTPUT.PUT_LINE('Grade: Fail');
   END CASE;
END;
```

👉 Output for `marks = 75`:

```makefile
Grade: B
```

4. Multiply First 10 Natural Numbers by 10

✅Loop to Multiply Numbers by 10

```
DECLARE
   i NUMBER;
BEGIN
   FOR i IN 1..10 LOOP
      DBMS_OUTPUT.PUT_LINE(i || ' x 10 = ' || (i * 10));
   END LOOP;
END;
```

By Maha Rana

👉 Output:

```
1 x 10 = 10
2 x 10 = 20
3 x 10 = 30
4 x 10 = 40
5 x 10 = 50
6 x 10 = 60
7 x 10 = 70
8 x 10 = 80
9 x 10 = 90
10 x 10 = 100
```

----------THE END----------

By Maha Rana

May Allah grant you success, ease your efforts, and bless you with wisdom and perseverance. Keep your faith strong, for with prayer, every challenge becomes an opportunity. I hope this file helps you a lot, and please remember me in your prayers.

Best wishes for Finals!
Maha ☺