**PLSQL TEST-1 ASSIGNMENT**

**1. PL/SQL**

* PL/SQL (Procedural Language/Structured Query Language) is Oracle Corporation's procedural extension for SQL.
* It allows you to write code that combines SQL statements with procedural constructs like loops, conditionals, and exception handling.
* PL/SQL is used to write complex database applications, such as stored procedures, functions, triggers, and packages.

Syntax:

* DECLARE - This is where variables, constants, and cursors are declared
* BEGIN - Assign values to variables
* EXCEPTION - This is where exceptions (errors) are handled.
* END - End Of the Program

**2.DATATYPES IN PL/SQL**

These data types hold a single value

Datatypes in PL/SQL

* Char
* Varchar2
* Number
* Date

**3. MAXIMUM STORAGE**

**🡪** 32,767

**4.%TYPE AND %ROWTYPE**

**%TYPE-** The attribute is used to declare a variable with the same data type as a column in a table or a variable that has already been declared.

DECLARE

v\_emp\_name employees.first\_name%TYPE

BEGIN

SELECT first\_name INTO v\_emp\_name FROM employees WHERE employee\_id = 100;

DBMS\_OUTPUT.PUT\_LINE('Employee Name: ' || v\_emp\_name);

END;

**%ROWTYPE –** The attribute is used to declare a record variable that can hold an entire row of data from a table or a cursor. This attribute creates a composite data type that consists of all the columns in the table or cursor, with each field in the record corresponding to a column in the row.

DECLARE

v\_emp\_record employees%ROWTYPE;

BEGIN

SELECT \* INTO v\_emp\_record FROM employees WHERE employee\_id = 100;

DBMS\_OUTPUT.PUT\_LINE('Employee Name: ' || v\_emp\_record.first\_name || ' ' || v\_emp\_record.last\_name);

END;

**5.BLOB**

BLOB – Binary Large Object

PL/SQL, a **BLOB** (Binary Large Object) is a data type that stores large binary data, such as images, multimedia files, or any data that is not in a character format. BLOBs can store up to 4 gigabytes (GB) of binary data in Oracle databases.

**6.Cursors**

A cursor is a pointer that points to a result of a query.

PL/SQL has two types of cursors:

* Implicit cursors
* Explicit cursors.

Cursor Attributes

**%FOUND**

* Returns **TRUE** if the last fetch or DML operation affected one or more rows.
* Returns FALSE if no rows were affected.

**%NOTFOUND**

* Returns TRUE if the last fetch did not return any row (i.e., the cursor has reached the end of the result set).
* Returns FALSE if it successfully fetched a row.

**%ROWCOUNT**

* Returns the number of rows fetched so far by the cursor.

**%ISOPEN**

* Returns TRUE if the cursor is currently open.
* Returns FALSE if the cursor is closed.
* Usage: Cursor\_name % Isopen.

**7. Implicit Cursor and Explicit Cursor**

**Implicit Cursor**

**Definition:** An implicit cursor is automatically created by Oracle whenever an SQL statement (like SELECT, INSERT, UPDATE, or DELETE) is executed within a PL/SQL block. You do not need to declare or manage an implicit cursor; Oracle handles it internally.

**Attributes:**

* SQL%FOUND
* SQL%NOTFOUND
* SQL%ISOPEN
* SQL%ROWCOUNT

**Explicit Cursor**

An explicit cursor is defined and controlled by the programmer. It is used when you need to handle a query that returns multiple rows or when you need more control over the query processing.

**Attributes :**

* C1%FOUND
* C1%NOTFOUND
* C1%ISOPEN
* C1%ROWCOUNT

**8. Closing a Cursor**

Closing a Cursor will do Releasing a Allocating a Memory

**9. Error Instances?**

1. **When you Open a Cursor Which is already Opened –** Cursor Already Opened
2. **When you Close a Cursor Which is already Opened –** Invalid Cursor
3. **When you Close a Cursor Which is not yet Opened –** Invalid Cursor

**10.Parameterized Cursors**

A **parameterized cursor** in PL/SQL is an explicit cursor that accepts parameters, allowing you to pass values to the cursor at runtime. This makes parameterized cursors more flexible and reusable, as they can be used with different input values without needing to declare multiple cursors.

**Syntax**

CURSOR cursor\_name (parameter\_name datatype, ...)

IS

SELECT\_statement;

**11.Output**

Output :

Cursor Already Opened.

**12.Correct Syntax**

**iii)**

OPEN C1;

LOOP

FETCH C1 INTO CR1;

NULL;

EXIT WHEN C1%NOTFOUND;

END LOOP;

CLOSE C1;

**13.Output**

SQL> DECLARE

2 X NUMBER:= 10;

3 Y NUMBER;

4 BEGIN

5 DBMS\_OUTPUT.PUT\_LINE(X+Y);

6 END;

7 /

PL/SQL procedure successfully completed.

**14. CASE STATEMENT**

The CASE statement in PL/SQL is a control structure that allows you to execute a sequence of statements based on the evaluation of different conditions. It's similar to the IF-THEN-ELSE statement but can be more concise and easier to read when dealing with multiple conditions

**EXAMPLE**

DECLARE

v\_marks NUMBER := 85;

v\_grade CHAR(1);

BEGIN

CASE

WHEN v\_marks >= 90 THEN

v\_grade := 'A';

WHEN v\_marks >= 80 THEN

v\_grade := 'B';

WHEN v\_marks >= 70 THEN

v\_grade := 'C';

WHEN v\_marks >= 60 THEN

v\_grade := 'D';

ELSE

v\_grade := 'F';

END CASE;

DBMS\_OUTPUT.PUT\_LINE('Marks: ' || v\_marks || ', Grade: ' || v\_grade);

END;

**OUTPUT**

Marks: 85, Grade: B

PL/SQL procedure successfully completed.

**15. IF AND ELSE**

**IF AND ELSE STATEMENT**

The IF ... ELSE statement in PL/SQL is used to execute a block of code conditionally based on whether a condition is true or false. It allows you to implement branching logic in your PL/SQL programs.

**EXAMPLE**

DECLARE

mark NUMBER := 75;

result VARCHAR2(10);

BEGIN

IF mark >= 60 THEN

result := 'Pass';

ELSE

result := 'Fail';

END IF;

DBMS\_OUTPUT.PUT\_LINE('Marks: ' || marks || ', Result: ' || result);

END;

**16. TO FIND ODD OR EVEN**

SQL> DECLARE

2 N NUMBER := 20;

3 BEGIN

4 IF MOD(N,2)=0 THEN

5 DBMS\_OUTPUT.PUT\_LINE('ENTERED NUM IS EVEN : ' || N);

6 ELSE

7 DBMS\_OUTPUT.PUT\_LINE('ENTERED NUM IS ODD : ' || N);

8 END IF;

9 END;

10 /

ENTERED NUM IS EVEN : 20

PL/SQL procedure successfully completed.

**17. Write a pl/sql block to insert the records from the table Employee to Employee temp. Delete all the**

**records in Employee temp table before inserting it.**

BEGIN

DELETE FROM EMPLOYEE\_TEMP\_SRP;

INSERT INTO SUR\_EMPLOYEE\_TEMP (ET\_BU, ET\_PLANT, ET\_ID, ET\_NAME, ET\_POS, ET\_STATUS)

SELECT EMP\_BU, EMP\_PLANT, EMP\_ID, EMP\_NAME, EMP\_POS, EMP\_STATUS FROM SUR\_EMP;

IF SQL%ROWCOUNT > 0 THEN

DBMS\_OUTPUT.PUT\_LINE(SQL%ROWCOUNT||' RECORDS ARE INSERT');

END IF;

END;

**18. Write a pl/sql block to insert the records from Employee table to Employee hist.**

**Update the emp status as 'M' after inserting into Employee hist table.**

BEGIN

INSERT INTO EMPLOYEE\_HIST\_SRP (EH\_BU, EH\_PLANT, EH\_ID, EH\_NAME, EH\_POS, EH\_STATUS)

SELECT EMP\_BU, EMP\_PLANT, EMP\_ID, EMP\_NAME, EMP\_POS, EMP\_STATUS FROM SUR\_EMP;

IF SQL%ROWCOUNT > 0 THEN

DBMS\_OUTPUT.PUT\_LINE(SQL%ROWCOUNT||' RECORDS ARE INSERT');

END IF;

UPDATE SUR\_EMP SET EMP\_STATUS = 'M';

IF SQL%ROWCOUNT > 0 THEN

DBMS\_OUTPUT.PUT\_LINE(SQL%ROWCOUNT||' RECORDS ARE UPDATED IN SUR\_EMP

TABLE AFTER INSERTING SUR\_EMPLOYEE\_HIST TABLE');

END IF;

END;

**19. Write a pl/sql block to print the particular employee's salary (Use Paramaterized cursor)**

DECLARE

CURSOR C1 (E\_Position Varchar2(10))

IS

SELECT \* FROM SALARY;

CR1 C1%ROWTYPE;

BEGIN

OPEN C1('Trainee');

LOOP

FETCH C1 INTO CR1;

EXIT WHEN C1%NOTFOUND;

DBMS\_OUTPUT.PUT\_LINE('Salary:' || CR1.SALARY || '---->' ||'Position :' || CR1.POSITION );

END LOOP;

END;

**20. Write a pl/sql block to insert the records from SUPLR PRODUCTS table to SUPLR PROD TEMP table.**

Note: Seq no is NULL in supir products table, Ako insert seq no into suplr\_prod\_temp while inserting it.

BEGIN

INSERT INTO SUPLR\_PROD\_TEMP\_SRP (SPT\_BU, SPT\_SUPLR\_ID, SPT\_PROD\_ID, SPT\_PROD\_REV,

SPT\_UOM, SPT\_TCF\_ID, SPT\_SEQ\_NO)

SELECT SUP\_BU, SUP\_SUPLR\_ID, SUP\_PROD\_ID, SUP\_PROD\_REV, SUP\_UOM, SUP\_TCF\_ID, NULL FROM SUR\_SUPLR

IF SQL%ROWCOUNT > 0 THEN

DBMS\_OUTPUT.PUT\_LINE(SQL%ROWCOUNT||' RECORDS ARE INSERT');

END IF;

END;

**21. Write a pl/sql block to insert the records from products table to prod temp.**

**Eliminate the duplicate rows and insert the records.**

BEGIN

INSERT INTO SUR\_PROD\_TEMP

SELECT DISTINCT \* FROM SUR\_PRODUCTS;

IF SQL%ROWCOUNT > 0 THEN

DBMS\_OUTPUT.PUT\_LINE(SQL%ROWCOUNT||' RECORDS ARE INSERT');

END IF;

END;