23-6-25: MODULE - PL/SQL Programming :- 9:00 AM

→ PL/SQL (Procedural language / structured query language) is a block structured language developed by oracle.

→ It allow developers to combine the power of SQL with procedural programming constructs.

Advantages:-

* Execute SQL Queries
* Define variables.
* Create reusable program units
* Handle exceptions.

Structure of PL/SQL Block:-

PL/SQL Block

- Declare
- Begin
- Exception
- End.

```
DECLARE
    declaration statements;
BEGIN
    executable statements
EXCEPTIONS
    exception handling stmts
END
```

→ Among three different blocks (Declaration block, execution block & exception block). It is compulsory to specify execution block while other two block's are optional.

Types of blocks in PL/SQL:-

Anonymous Blocks:-

→ This are the blocks which don't have any associated with the block

→ The most disadvantage of the PL/SQL anonymous block is that this block can be used only once.

Named Block:-

→ This are generally the procedures & functions which can be executed & used again

→ we can also pass list of parameters.

→ Each of the parameter declared inside the parenthesis can be either OUT/IN or INOUT parameter

PL/SQL Identifiers:-

→ Identifiers are constants, variables, exceptions, procedures cursors, and reserved words

→ should not exceed 30 characters

[1] Variables in PL/SQL:-

→ variables must be declared prior to its use.

→ Variable is like container that holds data during program execution.

Syntax:- variable_name datatype [NOT NULL := value].

```
DECLARE
        var1 INTEGER;
        var2 REAL
        var3 varchar2(20).
```

```
message varchar2(20) := 'hello';
```

[2] Displaying output in PL/SQL:-

→ outputs are displayed by using DBMS_OUTPUT which built in package that enable the user to display output

```
DECLARE
        var varchar2(40) := 'Cognizent';
BEGIN
        dbms_output.put_line(var);
END
```

[3] Comments in SQL.

→ Comments can be put within the code which has no effect in the code.

Single Line Comment :- To create a single line comment the symbol -- is used

Multi line comment :- To create comments that span over several lines, the symbols "/*", "*/" is used

```
-- This is single line comment

/*
    This is multi
    line comment
*/
```

[1] Taking input from user :-

→ PL/SQL allows to take input from the user & store in a variable using substitution variable

→ These variables are preced by '&' symbol

```
DECLARE
    a INTEGE := &amp;a;
    b VARCHAR2(30) := &amp;b;
BEGIN
    DBMS_OUTPUT.PUT_LINE (a);
    :
END
```

→ &a, &b are substitution variables where the user will be prompted to provide values

→ The user is asked to enter values for a & b when code runs

Example :- PL/SQL code to print sum of 2 numbers taken from u

```
DECLARE
        a  INTEGER  := &a;
        b  INTEGR  := &b;
        c  INTEGER.
BEGIN
        c := a + b;
        dbms_output.PUT_line("Sum of "||a|| and "||b|| is = "|| c);
END;
```

Enter value for a : 2

Enter value for b : 3

Sum of 2 and 3 is = 5.

CONTROL STRUCTURES :- The selection structure test a condition, then excutes one sequence of statements instead of another, depending wheather the condition is true or false.

* A condition is any variable or expression that returns a boolean value.

Conditional Control :

24-6-25

IF AND CASE :- Statements :

IF - THEN :- The simplest form of IF statement associates a condition with a sequence of statements enclosed by keywords THEN and END IF.

```
IF   condition  THEN
        =
        =
END  IF.
```

IF - THEN - ELSE Statement :- The second form of IF statement adds the keyword ELSE followed by alternative sequence of statements, as follows.

```
IF condition THEN
   ___
ELSE
   ___
END IF
```

(Nested IF-ELSE)

```
IF condition THEN
   ___
ELSE
   IF condition THEN
      ___
   ELSE
      ___
   END IF
END IF
```

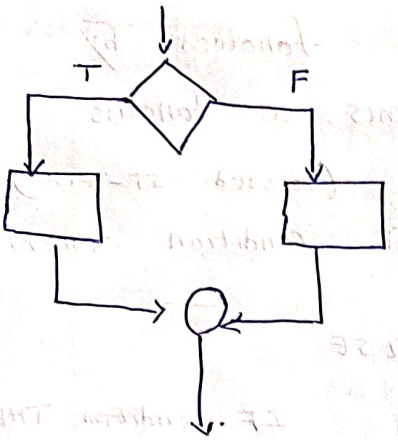
IF-THEN-ELSIF statement :- Sometimes we want to select an action from several mutually exclusive alternatives

```
IF condition1 THEN
   ___
ELSEIF condition 2 THEN
   ___
   :
ELSE
   ___
END IF.
```

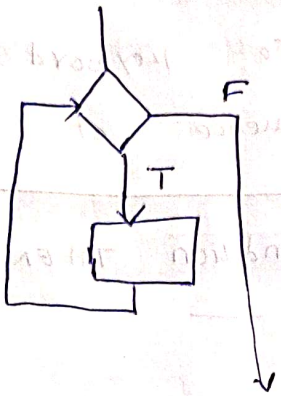
CASE steatement: Case statement uses selector rather than multiple boolean expressions

```
CASE grade
   WHEN 'A' THEN dbms_output.Put_line ('Excellent');
   WHEN 'B' THEN dbms_output.Put_line ('very good');
   :
   WHEN 'F' THEN dbms_output.Put_line ('Fail');
END CASE.
```

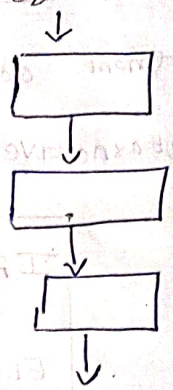
Selection



Iteration



Sequence



ITERATION:-

LOOP:- Loop statement is the simplest basic loop, which encloses a sequence of statements between the keyword LOOP and END LOOP.

```
LOOP
        seq_of_stmts
END LOOP
```

EXIT:- The exit stmt forces a loop to complete unconditionally. when EXIT stmt is encountered, the loop completes immediately & control passes to the next statement

```
LOOP
        IF condition THEN
            =
            EXIT
        END IF;
END LOOP
```

EXIT-WHEN:- Lets a loop complete conditionally

```
LOOP
        EXIT WHEN condition
END LOOP.
```

WHILE - LOOP :- The while-loop statement associates a condition with a seq of statements enclosed by the key words LOOP and END LOOP

```
WHILE condition LOOP
  ——
END LOOP
```

FOR - LOOP :- The number of iterations through a while loop is unknown whereas we specify the no. of iterations to be made.

```
FOR i IN 1..3 LOOP
  ==
END LOOP.
```

SEQUENTAL CONTROL.

GOTO statement :- Branches to a label unconditionally. the label must be unique within its scope & precede an executable statement. whe GOTO encounters control transfer to the labeled block

```
BEGIN
  ==
  GOTO insert_row;
  ==
  <<insert_row>>
  INSERT ——
END.
```

NULL statement :- Does nothing other than pass control to the next statement

```
WHEN c1 THEN.
  —
WHEN c2 THEN
  —
WHEN c3 Then
NULL.
```

Handling PL/SQL Errors :-

→ Run time errors arise from design faults, coding mistake, hardware failures and many other sources Although you cannot anticipate all possible errors, you can plan to handle certain kinds of errors meaning ful to PL/SQL Program.

→ we can handle errors using a mechanism called exception handling lets you "bulletProof" your program so that it continue operating in the presence of errors.

Predefined Exception:-

```
BEGIN
    SELECT 1/0 FROM dual;
EXCEPTION
    WHEN ZERO_DIVIDE THEN
        dbms_output.Put_line ('Division by zero error');
END;
```

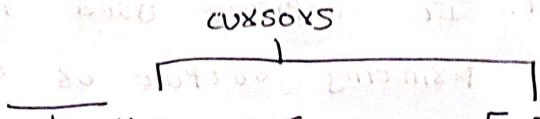
User-Defined Exception:-

```
DECLARE
    c_custom EXCEPTION;
    Value number := 5;
BEGIN
    If Value = 5 THEN
        RAISE c_custom;
    END IF;
EXCEPTION
    WHEN c_costom THEN
        dbms_output.Put_line ('exception raised');
END;
```

CURSORS :- A cursor is a pointer to this context area. PL/SQL controls the context are through a cursor.

→ A cursor holds the rows returned by a SQL stmt.

→ The set of rows the holds is referred as active set.



→ Automatically created by oracle whenever an SQL stmt is executed.

S.NO	Attribute & description
1	%FOUND :- returns TRUE if an INSERT, UPDATE or DELETE statement affected one or more rows
2	%NOTFOUND the logical opposite of %FOUND it returns TRUE if an INSERT, UPDATE or DELETE statement affected no rows
3	%ISOPEN Always returns FALSE for implicit cursors, because oracle closes the SQL cursors
4	%ROWCOUNT Returns the no.of rows affected by an INSERT, UPDATE or DELETE statement

Program:

```
DECLARE
        total_rows  number (2);
BEGIN
        UPDATE  customers
        SET     salary = salary + 500;
        IF      sql %  notfound  THEN
                dbms_output.put_line ('no customers selected')
        ELSEIF  sql % found THEN
                dbms_output.put_line (total_rows);
        END IF;
END;
```

PL/SQL Procedures & functions :-

→ A procedures is a named PL/SQL that performs a specific task but doesn't return a value directly to the caller. It typically used for actions like inserting data, printing output or updating records

```
CREATE OR REPLACE PROCEDURE greet_user (n IN VARCHAR2)
BEGIN
    DBMS_OUTPUT.PUT_LINE ('Hello, ' || name);
END;

EXEC greet_user ('cognizent');
```

Functions:

→ A function is named PL/SQL block that performs a task & returns a single value.

→ Functions are often used in calculations or to return data in SELECT query

```
CREATE OR REPLACE FUNCTION square (n NUMBER) RETURN NUMBER
BEGIN
    RETURN n * n;
END;

SELECT square (1) FROM dual;
```

PACKAGE:

Specification:

```
CREATE OR REPLACE PACKAGE math_PKG IS
    FUNCTION square (n NUMBER) RETURN NUMBER;
END;
```

Body:-

```
CREATE OR REPLACE PACKAGE BODY math_PKG IS
    FUNCTION square (n NUMBER) RETURN NUMBER IS
BEGIN
        RETURN n * n;
END;
END;
```

Benifits of PACKAGE:-

→ Modularize code
→ Group procedures/functions.
→ Improve performance
→ Encapsulation

Triggers:

A triggers is a stored procedure that oracle database invokes automatically whenever a specified event occurs.

```
CREATE OR REPLACE TRIGGER instead_of_trigger
INSTEAD OF INSERT ON emp_view
BEGIN
        -- logic
END
```