# PL/SQL

# Overview..

- Introduction
- PL/SQL Block Structure
- PL/SQL Variables
- PL/SQL IF Statement
- PL/SQL Loop Statement:
  - PL/SQL WHILE Loop Statement PL/SQL FOR Loop Statement.
- PL/SQL Function ; PL/SQL Procedure
- Exceptions
- Cursors and Triggers

# About SQL

- The purpose of SQL is to provide an interface to a relational database such as Oracle Database, and all SQL statements are instructions to the database.

- The strengths of SQL provide benefits for all types of users, including application programmers, database administrators, managers, and end users.

# Why PL/SQL?

How can we have a chain of SQL statements which produce result according to the output of previous queries?

How can we take any smart decisions based on users input or based on output on previous queries..?

How can we automate any task using SQL?

# Introduction

- The PL/SQL programming language was developed by Oracle Corporation in the late 1980s as procedural extension language for SQL and the Oracle relational database.

- PL/SQL, supplement SQL with standard programming-language features like:

  - Block (modular) structure
  - Flow-control statements and loops
  - Variables, constants, and types
  - Structured data
  - Customized error handling

# Why PL/SQL?

- The purpose of PL/SQL is to combine database language and procedural programming language.
- By extending SQL, PL/SQL offers a unique combination of power and ease of use.
- PL/SQL fully supports all SQL data manipulation statements.
- We can write procedures and functions which can be invoked from different applications.

# PL/SQL Block

- PL/SQL is a block-structured language.
- Each program written in PL/SQL is written as a block.
- Blocks can also be nested.
- Each block is meant for a particular task.

# PL/SQL Block Structure

**Header** *(named blocks only)*

```
IS
```

**Declare Section**

```
BEGIN
```

**Execution Section**

```
EXCEPTION
```

**Exception Section**

```
END;
```

# Header Section

Relevant for named blocks only

- Stored procedures (used to perform repetitive code.)
- Stored functions (used to return value to the calling block),
- Packages (allows related objects to be stored together),
- Triggers (pl/sql block executed implicitly whenever the triggering event takes place).

Determines the way that the named block or program must be called.

Includes the name, parameter list, and RETURN clause (for a function only).

- Header Section: Example

  - Below is header section of stored procedure:

```
CREATE OR REPLACE PROCEDURE
    print ( p_num NUMBER ) …
```

  - Below is header section of stored function:

```
CREATE OR REPLACE FUNCTION
    add ( p_num1 NUMBER, p_num2 NUMBER )
    RETURN NUMBER …
```

# Declare Section

- Relevant for anonymous blocks.
- Contains declarations of variables, constants, cursors, user-defined exceptions and types.
- Optional section , but if you have one, it must come before the execution and exception sections.

```
DECLARE
    v_name VARCHAR2(35);
    v_id NUMBER := 0;
```

# Execution Section

- Mandatory section of PLSQL block
- Contains SQL statements to manipulate data in the database
- Contains PL/SQL statements to manipulate data in the block.
- Every block must have at least one executable statement in the execution section.

```
BEGIN
    SELECT ename
    INTO v_ename
    FROM emp
    WHERE empno = 7900 ;
    DBMS_OUTPUT.PUT_LINE
    ('Employee name :' || v_ename);
END;
```

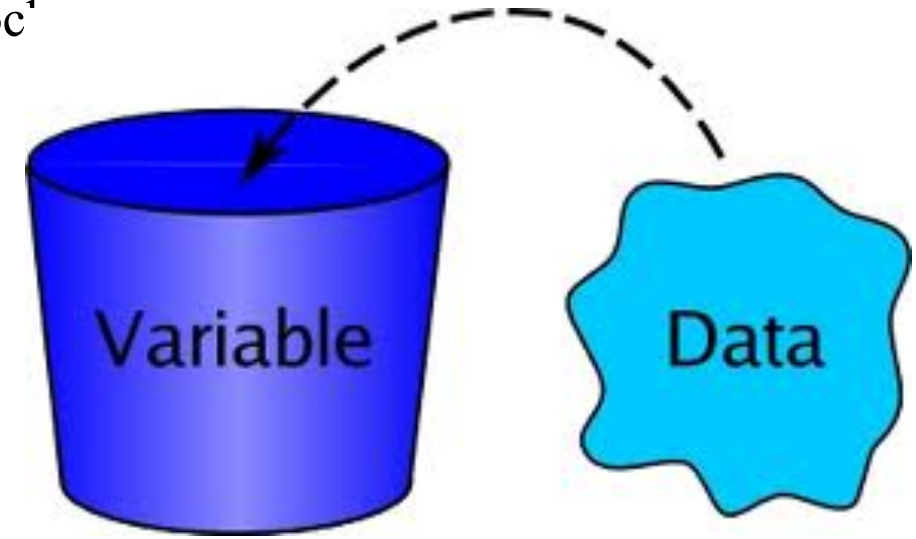
# Exception Section

- The last section of the PL/SQL block.
- It contains statements that are executed when a runtime error occurs within a block.
- An optional section.
- Control is transferred to this section when an run time error is encountered and it is handled

```
EXCEPTION
      WHEN NO_DATA_FOUND THEN
           DBMS_OUTPUT.PUT_LINE
           (' There is no employee with Employee no  7900 ');
END;
```

# PL/SQL Variables

- These are placeholders that store the values that can change through the PL/SQL Bloc




- PL/SQL lets you declare constants and variables, then use them in SQL and procedural statements anywhere an expression can be used

- A constant or variable  must be declared before referencing it in other statements

# Syntax for declaring variables

```
variable_name [CONSTANT] datatype
[NOT NULL] [:= expr | DEFAULT expr]
```

Note: Square brace indicates optional

- Variable_name is the name of the variable.
- Datatype is a valid PL/SQL datatype.
- NOT NULL is an optional specification on the variable.
- A value or DEFAULT value is also an optional specification, where you can initialize a variable.
- Each variable declaration is a separate statement and must be terminated by a semicolon.
- CONSTANT keyword is used to declare constants.

# PL/SQL variables

- Valid variable declarations

```
DECLARE
  v_Activedate DATE;
  V_cust_id NUMBER(2) NOT NULL := 10;
  V_Address VARCHAR2(13) := 'Pune';
  V_sr_id  NUMBER DEFAULT 201;
  V_Name VARCHAR2(20) DEFAULT 'Aditya'
```

- Valid constant declaration

```
c_constant CONSTANT NUMBER := 1400;
```

- Invalid Declarations

```
v_cust_id NUMBER(2) NOT NULL;
v_name VARCHAR2 DEFAULT 'Sachin';

c_constant CONSTANT NUMBER ;
c_constant NUMBER CONSTANT;
```

# Guidelines for Declaring PL/SQL Variables

- Follow the naming Rules
  - ✔ The variable name must be less than 31 characters
  - ✔ The starting of a variable must be an ASCII letter
  - ✔ It can be either lowercase or uppercase
  - ✔ A variable name can contain numbers, underscore, and dollar sign characters followed by the first character
- Follow the naming conventions
- Initialize variables designated as NOT NULL and CONSTANT
- Declare one identifier per line
- Initialize identifiers by using the assignment operator (:=) or the reserved word “DEFAULT”

# Declaring variable of record type : Example

```
DECLARE
  -- Type declaration
     TYPE EmpRec IS RECORD(
     empno   emp.empno%TYPE,
     ename   emp.ename%TYPE,
     salary  emp.sal%TYPE
     );
   -- Record type variable declaration
     V_emp_Rec emprec;
```

| V_emp_Rec | |
|---|---|
| empno | number(10) |
| ename | varchar2(25) |
| salary | varchar2(75) |

# Sample program of PL/SQL

Oracle SQL*Plus

File Edit Search Options Help

```
SQL> SET SERVEROUTPUT ON
SQL> BEGIN
  2   DBMS_OUTPUT.PUT_LINE('Hello! Welcome to PL/SQL');
  3  END;
  4  /
Hello! Welcome to PL/SQL

PL/SQL procedure successfully completed.
```

# Program using variables

```
DECLARE

   X   NUMBER(3) :=  10;
   Y   NUMBER(3) :=  20;

BEGIN

    DBMS_OUTPUT.PUT_LINE
   ('The value of variable X is : ' ||  X);

    DBMS_OUTPUT.PUT_LINE
   ('The value of variable Y is : ' ||  Y);

END;
```

# Accepting variables from users

```
DECLARE
   v_num1 NUMBER(3):=&n1;
   v_num2 NUMBER(3):=&n2;
IS
BEGIN
   DBMS_OUTPUT.PUT_LINE
   ('The value of variable v_num1 is : ' ||  v_num1);

   DBMS_OUTPUT.PUT_LINE
   ('The value of variable v_num2 is : ' ||  v_num2);
END;
```

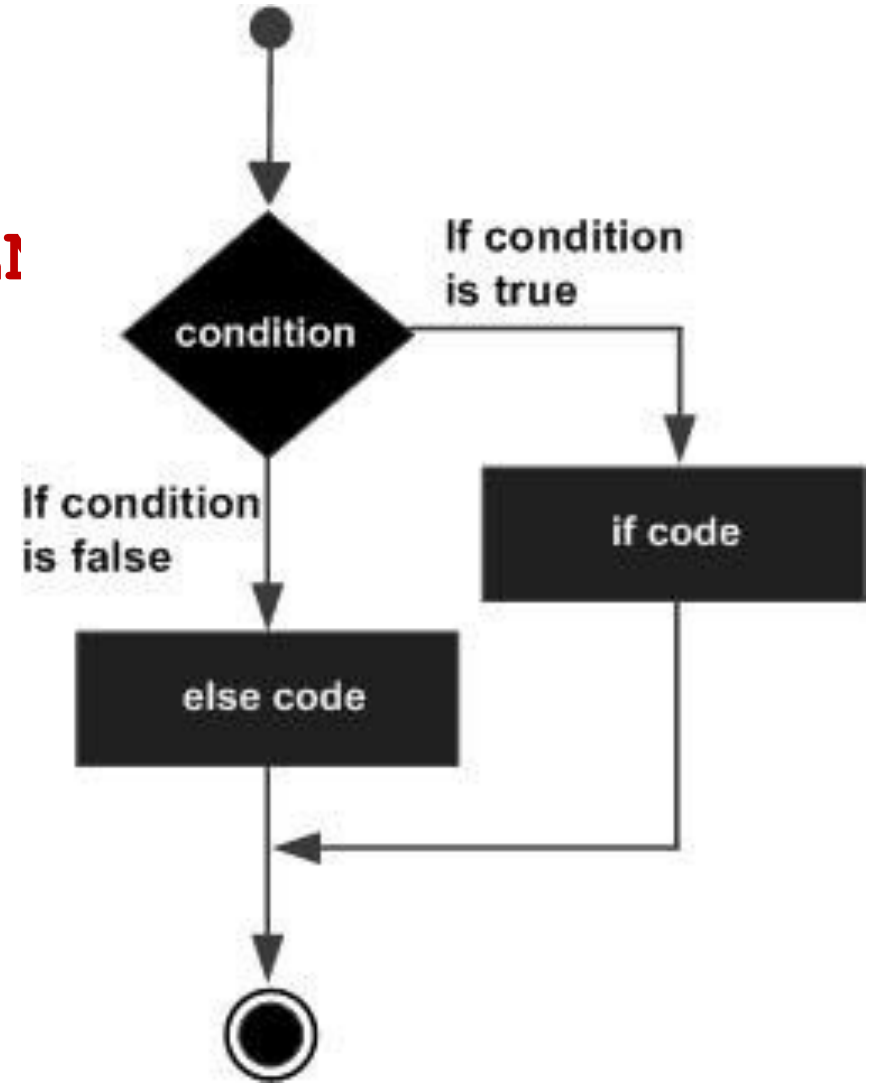
# PL/SQL Control Structures

- PL/SQL, like other 3GL has a variety of control structures which include
  - Conditional statements
    - IF
    - CASE
  - Loops
    - Simple loop
    - While loop
    - For loop

# IF .. End if

- Syntax

```
IF condition THEN
  Statements;
ELSE
  Statements;
END IF;
```



# Simple example

```
-- Block to demonstrate IF...ELSE...END IF
DECLARE
  v_empno emp.empno%TYPE;
  v_comm  emp.comm%TYPE;

BEGIN
  SELECT comm INTO v_comm FROM emp
   WHERE empno = v_empno;

  IF v_comm IS NULL THEN
        DBMS_OUTPUT.PUT_LINE
    ('This employee doesnt get commission');
   ELSE
    DBMS_OUTPUT.PUT_LINE
        ('This employee gets commission');
  END IF;

  DBMS_OUTPUT.PUT_LINE
   ('This line executes irrespective of the condition');
END;
```

# PL/SQL Loop Control Structures

- LOOP Statements
  - Simple Loops
  - WHILE Loops
  - FOR Loops

# LOOP Statements

- Simple Loops

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

Note: Add EXIT statement to exit from the loop

- WHILE Loops

```
WHILE  condition
LOOP
        Statements;
END LOOP;
```

Note: Condition is evaluated before each iteration of the loop

# Loop Example

```
DECLARE
    v_i NUMBER(2) := 1;
  BEGIN
    LOOP
        DBMS_OUTPUT.PUT_LINE('Value : '|| v_i);
        EXIT WHEN v_i = 10;
        v_i:=v_i+1;
    END LOOP;
  END;
```

# While loop example

```
DECLARE
   v_i  NUMBER(2)  := 1;
  BEGIN
   WHILE ( v_i <= 10 )
   LOOP
      DBMS_OUTPUT.PUT_LINE('Value : '|| v_i);
      v_i:=v_i+1;
   END LOOP;
  END;
```

# For loop

- The number of iterations for simple loops and WHILE loops is not known in advance, it depends on the loop condition. Numeric FOR loops, on the other hand, have defined number of iterations.

```
FOR  counter  IN  [REVERSE] low_bound  ..   high_bound
LOOP
   Statements;
END
```

- Where:
  - counter: is an implicitly declared integer whose value automatically increases or decreases by 1 on each iteration
  - REVERSE: causes the counter to decrement from upper bound to lower bound
  - Low_bound: specifies the lower bound for the range of counter values
  - High_bound: specifies the upper bound for the range of counter values

# For Loop example

```
BEGIN
    FOR v_i IN 1..10
 /* The LOOP VARIABLE v_i of type
BINARY_INTEGER is   declared
automatically */
     LOOP
    DBMS_OUTPUT.PUT_LINE('Value : '|| v_i);
     END LOOP;
END;
```

# For Loop with EXIT condition

```
DECLARE
    myNo NUMBER(5):= &myno;
    counter NUMBER(5):=1;
BEGIN
    FOR i IN 2..myNo-1
    LOOP
        counter:= counter+1;
        EXIT WHEN myNo mod i = 0;
    END LOOP;

    IF counter = myNo-1 THEN
    DBMS_OUTPUT.PUT_LINE( 'The given
        number is prime' );
    ELSE
        DBMS_OUTPUT.PUT_LINE('The given number is not
                a prime number' );
    END IF;
END;
```

# Procedures

- A Procedure is a subprogram unit that consists of a group of PL/SQL statements. Each procedure in Oracle has its own unique name by which it can be referred. This subprogram unit is stored as a database object. Below are the characteristics of this subprogram unit.
- CREATE OR REPLACE PROCEDURE
- <procedure_name>
- (
- <parameterl IN/OUT <datatype>
- )
- [ IS | AS ]
- <declaration_part>
- BEGIN
- <execution part>
- EXCEPTION
- <exception handling part>
- END;

# What is Function

- Functions is a standalone PL/SQL subprogram. Like PL/SQL procedure, functions have a unique name by which it can be referred. These are stored as PL/SQL database objects. Below are some of the characteristics of functions.
- CREATE OR REPLACE FUNCTION <procedure_name>(<parameterl IN/OUT <datatype>)RETURN <datatype>[ IS | AS ]<declaration_part>BEGIN<execution part> EXCEPTION<exception handling part>END;

```
1.  CREATE OR REPLACE FUNCTION welcome_msg_func ( p_name IN VARCHAR2)
2.  RETURN VARCHAR2
3.  IS
4.  BEGIN
5.  RETURN ('Welcome '|| p_name);
6.  END;
7.  /
```

**Output:**

Function created

Function Created

```
8.  DECLARE
9.  lv_msg VARCHAR2(250);
10. BEGIN
11. lv_msg := welcome_msg_func ('Guru99');
12. dbms_output.put_line(lv_msg);
13. END;
```

Calling function with 'Guru99' as parameter

**Output:**

Welcome Guru99

```
14. SELECT welcome_msg_func('Guru99') FROM DUAL;
```

**Output:**

Welcome Guru99