4

FUNDAMENTALS OF PL/SQL

Unit Structure

4.0 Objectives

4.1 Overview of PL/SQL

Features of PL/SQL

Advantages of PL/SQL

PL/SQL Block Structure

4.2 PL/SQL Identifiers

Variable Declaration in PL/SQL

Constants

Literals

4.3 PL/SQL Expressions and Comparisons

PL/SQL Operators

PL/SQL Operator Precedence

CASE Expressions

Null Values in Comparisons, Conditional Statements

4.4. PL/SQL Data Types

Number Types

Character Types

Boolean Type

Date Time Types.

LOB Types

4.0 OBJECTIVES

This chapter makes you to understand the basic concepts in PL/SQL.

It guides you to write PL/SQL block on yourself for a given problem.

4.1 OVERVIEW OF PL/SQL

The PL/SQL programming language was developed by Oracle Corporation in the late 1980s and early 90's to enhance the capabilities of SQL as a procedural extension language for SQL and the Oracle relational database. This is a combination of SQL embedded with the procedural features of programming languages.

- PL/SQL is a high-performance transaction-processing language and completely portable.
- PL/SQL provides a integrated, interpreted and OS independent programming environment.
- This can also be called directly from the command-line SQL*Plus interface.
- It has an option from external programming language calls to the database.
- PL/SQL is also available in Times Ten in-memory database, IBM DB2 and so on.

Features of PL/SQL

PL/SQL is built-in with SQL.

- It provides error checking facility.
- It offers numerous data types.
- It offers different programming structures.
- It has structured programming through functions and procedures.
- It has object-oriented programming.
- It has the development of web applications and server pages.

Advantages of PL/SQL

- SQL is the standard database language and PL/SQL is strongly integrated with SQL and supports static SQL and dynamic SQL.
- Static SQL supports DML operations and transaction control from PL/SQL block. In Dynamic SQL, easy to embed DDL statements on PL/SQL program blocks.
- PL/SQL allows sending an entire block of statements to the database at one time. Reduces network traffic, provides high performance on applications.
- PL/SQL gives high productivity to programmers as it can query, transform, and update data in a database.
- PL/SQL saves time on design and debugging by strong features, such as exception handling, encapsulation, data hiding, and object-oriented data types.
- Portable Applications can be written in PL/SQL.
- High security level.

- Access to predefined SQL packages.
- Support for Object-Oriented Programming.
- Support for developing Web Applications and Server Pages.

PL/SQL Block Structure

PL/SQL programs are divided and written in logical blocks of code. The blocks ar also have two different types.

1. Anonymous Block : A Block of code without name
2. Named Block : A Block of code has a specific name such as function name, subprogram name like, any valid name for the block.

Each block has three sections.

S.No	Sections & Description
1	**Declarations** This section starts with the keyword **DECLARE**. It is an optional section and defines all variables, cursors, subprograms, and other elements to be used in the program.
2	**Executable Commands** This section is enclosed between the keywords **BEGIN** and **END.** It consists of the executable statements of the program. It should have at least one executable statement, NULL command is used to indicate that nothing should be executed.
3	**Exception Handling** This section starts with the keyword **EXCEPTION.** This optional section contains exception(s) that handle errors in the program.

Every PL/SQL statement ends with a semicolon (;). PL/SQL blocks can be nested within other PL/SQL blocks using BEGIN and END.

The syntax of PL/SQL block structure **DECLARE**

<declarations section>

BEGIN

<executable command(s)>

EXCEPTION

<exception handling>

END;

Example :

```
DECLARE
msg  varchar2(20):= 'Hello World';
BEGIN
dbms_output.put_line (msg);
END;
```

The end; line signals the end of the PL/SQL block. To run the code from the SQL command line, use / at the beginning of the first blank line after the last line of the code.This produces the following result

Hello World

The PL/SQL Comments

Program comments are explanatory statements that can be included in the PL/SQL code that you write and helps anyone reading its source code. All programming languages allow some form of comments.

The PL/SQL supports single-line and multi-line comments. All characters available inside any comment are ignored by the PL/SQL compiler.

- Single-line comments ,the line start by -- (double hyphen)
- Multi-line comments must enclosed by /* and */.

```
DECLARE
 -- variable declaration
 var1 varchar2(20):= 'Hello World';
BEGIN
   dbms_output.put_line(var1);
END;
/
```

When the above code is executed at the SQL prompt, it produces the following result

Hello World

4.2 THE PL/SQL IDENTIFIERS

PL/SQL identifiers are constants, variables, exceptions, procedures, cursors, and reserved words. This consists of a letter optionally followed by more letters, numerals, dollar signs, underscores, and number signs. This should not exceed 30 characters. By default, identifiers are not case-

sensitive. The identifier can be named integer or INTEGER to represent a numeric value. You cannot use a reserved keyword as an identifier.

Variable Declaration in PL/SQL

PL/SQL variables must be declared in the declaration section or in a package as a global variable. When you declare a variable, PL/SQL allocates memory for the variable's value and the storage location is identified by the variable name.

The syntax for declaring a variable is

variable_name [CONSTANT] datatype [NOT NULL] [:= | DEFAULT initial_value]

Where, *variable_name* is a valid identifier in PL/SQL

sales number(10, 2);

pi CONSTANT double precision := 3.1415;

name varchar2(25);

address varchar2(100);

When you provide a size, scale or precision limit with the data type, it is called a **constrained declaration**. Constrained declarations require less memory than unconstrained declarations.

Example:

sales number(10, 2);

name varchar2(25);

address varchar2(100);

Initializing Variables in PL/SQL

DEFAULT

PL/SQL assigns it a default value of NULL. If you want to initialize a variable with a value other than the NULL value, you can do so during the declaration, using either of the following −

The **DEFAULT** keyword

The **assignment**(:=)operator

Example:

counter binary_integer := 0;

greetings varchar2(20) DEFAULT 'Have a Good Day';

You can also specify that a variable should not have a **NULL** value using the **NOT NULL** constraint. If you use the NOT NULL constraint, you must explicitly assign an initial value for that variable.

It is a good programming practice to initialize variables properly otherwise, sometimes programs would produce unexpected results

Constants

A constant holds a value that once declared, does not change in the program. A constant declaration specifies its name, data type, and value, and allocates storage for it. The declaration can also impose the NOT NULL constraint.

Declaring a Constant

A constant is declared using the CONSTANT keyword. It requires an initial value and does not allow that value to be changed.

Example:

PI CONSTANT NUMBER := 3.141592654;

PL/SQL Block:

```
DECLARE
  -- constant declaration
  pi constant number := 3.141592654;
  -- other declarations
  radius number(5,2);
  dia number(5,2);
  circumference number(7, 2);
  area number (10, 2);
BEGIN
  -- processing
  radius := 9.5;
  dia := radius * 2;
  circumference := 2.0 * pi * radius;
  area := pi * radius * radius;
  -- output
  dbms_output.put_line('Radius: ' || radius);
```

```
   dbms_output.put_line('Diameter: ' || dia);
   dbms_output.put_line('Circumference: ' || circumference);
   dbms_output.put_line('Area: ' || area);
END;
/
```

When the above code is executed at the SQL prompt, it produces the following result −

```
Radius: 9.5
Diameter: 19
Circumference: 59.69
Area: 283.53
```

Literals

A literal is an explicit numeric, character, string, or Boolean value not represented by an identifier. For example, TRUE, 786, NULL, 'tutorials point' are all literals of type Boolean, number, or string. PL/SQL, literals are case-sensitive. PL/SQL supports the following kinds of literals

Numeric Literals	050 78 -14 0 +32767 6.6667 0.0 -12.0 3.14159 +7800.00 6E5 1.0E-8 3.14159e0 -1E38 -9.5e-3
Character Literals	'A' '%' '9' ' ' 'z' '('
String Literals	'Hello, world!' 'Tutorials Point' '19-NOV-12'
BOOLEAN Literals	TRUE, FALSE, and NULL.
Date and Time Literals	DATE '1978-12-25'; TIMESTAMP '2012-10-29 12:01:01';

4.3 PL/SQL EXPRESSIONS AND COMPARISONS

Expressions are constructed using operands and operators. An operand may be a variable or constant that contributes value to an expression.

Simple arithmetic expression is:

```
-X / 2 + 3
```


Unary operators like negation operator (-) operate on one operand; binary operators like the division operator (/) operate on two operands.

PL/SQL OPERATORS

An operator is a symbol that tells the compiler to perform specific mathematical or logical operation.

Types of operators

Arithmetic Operators

Relational Operators

Comparison Operators

Logical Operators

Arithmetic Operators

Following table shows all the arithmetic operators supported by PL/SQL. Let us assume variable A holds 20 and variable B holds 15, then

Operator	Description	Example
+	Adds two operands	A + B = 35
-	Subtracts second one from the first one	A - B = 5
*	Multiplies both operands	A * B = 300
/	Divides numerator by de-numerator	A / B = 1
**	Exponentiation operator, raises one operand to the power of other	A ** 2 = 400

Relational Operators

Relational operators compare two expressions or values and return a Boolean result. Following table shows all the relational operators supported by PL/SQL. Consider the variable A has 10 and variable B has 20, then –

Operator	Description	Example
=	Checks if the values of two operands are equal or not, if yes then the condition is true.	(A = B) is not true.
!= <> ~=	Checks if the values of two operands are equal or not, if values are not equal then the condition is true.	(A != B) is true.
>	Checks if the value of the left operand is greater than the value of right operand, if yes then the condition is true.	(A > B) is not true.
<	Checks if the value of the left operand is less than the value of the right operand, if yes then the condition is true.	(A < B) is true.
>=	Checks if the value of left operand is greater than or equal to the value of right operand, if yes then condition is true.	(A >= B) is not true.
<=	Checks if the value of left operand is less than or equal to the value of right operand, if yes then condition is true.	(A <= B) is true.

Comparison Operators

Comparison operators are used for comparing one expression to another. The result is from TRUE, FALSE or NULL.

Operator	Description	Example
LIKE	The LIKE operator compares a character, string, or CLOB value to a pattern and returns TRUE if the value matches the pattern else FALSE.	If 'Zara Ali' like 'Z%_i' returns true, whereas, 'Nuha Ali' like 'Z% A_i' returns false.
BETWEEN	The BETWEEN operator tests whether a value lies in a specified range. x BETWEEN a AND b means that x >= a & x <= b.	If x is 10 then, x between 15 and 20 returns true, x between 5 and 10 returns true, but between 11 and 20 returns false.

IN	The IN operator tests set membership. x IN (set) means that x is equal to any member of set.	If x = 'm' then, x in ('a', 'b', 'c') is false but x in ('m', 'n', 'o') is true.
IS NULL	The IS NULL operator returns the BOOLEAN value TRUE if its operand is NULL or FALSE if it is not NULL. Comparisons involving NULL values is always NULL.	If x = 'm', then 'x is null' is false.

Logical Operators

Following table shows the Logical operators supported by PL/SQL. All these operators work on Boolean operands and produce Boolean results. Let us consider variable A has true and variable B has false.

Operator	Description	Example
and	Called the logical AND operator. If both the operands are true then condition is true.	A and B is false.
or	Called the logical OR Operator. If any of the two operands is true then condition becomes true.	A or B is true.
not	Called the logical NOT Operator. Used to reverse the logical state of its operand. If a condition is true then Logical NOT operator will make this false.	not (A and B) is true.

PL/SQL Operator Precedence

Operator precedence determines the grouping of terms in an expression. This affects how an expression is evaluated. Some operators have higher precedence than others; for example, the multiplication operator has higher precedence than the addition operator.

For example, x = 7 + 5 * 2; here, x is assigned 17, not 24 because operator * has higher precedence than +, so it first gets multiplied with 5*2 and then adds into 7.

Here, operators with the highest precedence appear at the top of the table, those with the lowest appear in bottom. Higher precedence operators will be evaluated first.

The precedence of operators : =, <, >, <=, >=, <>, !=, ~=, ^=, IS NULL, LIKE, BETWEEN, IN.

Operator	Operation
**	exponentiation
+, -	identity, negation
+, -	multiplication, division
+, -, ||	addition, subtraction, concatenation
comparison	
NOT	logical negation
AND	conjunction
OR	inclusion

CASE Expressions

There are two types of expressions used in CASE statements: simple and searched. These expressions correspond to the type of CASE statement in which they are used.

Simple CASE expression

A simple CASE expression selects a result from one or more alternatives and returns the result. It contains a block that stretch over several lines, it really is an expression that forms part of a larger statement, like assignment or a procedure call. The CASE expression uses a selector, an expression whose value determines which alternative to return.

Searched CASE Expression

A searched CASE expression lets you test different conditions instead of comparing a single expression with different values. This expression has no selector. Each WHEN clause contains a search condition that yields a BOOLEAN value, so you can test different variables or multiple conditions in a single WHEN clause.

Null Values in Comparisons, Conditional Statements

When working with nulls, you can avoid some common mistakes by keeping in mind the following rules:

- Comparisons in null values provide always NULL only.
- Using the logical operator NOT with a null provides NULL
- In conditional statements, if condition is NULL, its associated sequence of statements is not executed.
- If the expression is a simple CASE statement or CASE expression is NULL, it cannot be matched by WHEN NULL in condition. Here, need to use the searched case syntax and test WHEN expression IS NULL.

4.4. PL/SQL DATA TYPES

The PL/SQL variables, constants and parameters must have a valid data type, which specifies a storage format, conditions with valid range of values.

S. No	Category & Description
1	**Scalar** Single values which has no internal components, like **NUMBER, DATE** or **BOOLEAN**.
2	**Large Object (LOB)** Pointers to large objects which are stored separately from other data items, such as text, graphic images, video clips, and sound waveforms.
3	**Composite** Data items that have internal components that can be accessed individually. For example, collections and records.
4	**Reference** Pointers to other data items.

Numeric Types

Following table shows the numeric data types and their sub-types –

S. No	Data Type & Description
1	**PLS_INTEGER** Signed integer in range -2,147,483,648 through 2,147,483,647, represented in 32 bits
2	**BINARY_INTEGER** Signed integer in range -2,147,483,648 through 2,147,483,647, represented in 32 bits
3	**BINARY_FLOAT** Single-precision floating-point number
4	**BINARY_DOUBLE** Double-precision floating-point number
5	**NUMBER(prec, scale)** Numeric values with fixed or floating-point number with absolute value in range 1E-130 to (but not including) 1.0E126. A NUMBER variable can also represent 0
6	**DEC(prec, scale)** Fixed-point type specified in ANSI with maximum precision of 38 decimal digits
7	**DECIMAL(prec, scale)** Fixed-point type specified in IBM with maximum precision of 38 decimal digits
8	**NUMERIC(pre, secale)** Floating type with 38 decimal digits
9	**DOUBLE PRECISION** Floating-point type specified in ANSI with maximum precision of 126 binary digits (approximately 38 decimal digits)
10	**FLOAT** Floating-point type specified in ANSI and IBM with maximum precision of 126 binary digits (approximately 38 decimal digits)

11	**INT** Integer type specified in ANSI with maximum precision of 38 decimal digits
12	**INTEGER** Integer type specified in ANSI and IBM with maximum precision of 38 decimal digits
13	**SMALLINT** Integer type specified in ANSI and IBM with maximum precision of 38 decimal digits
14	**REAL** Floating-point type with 63 binary digits as maximum precision (approximately 18 decimal digits)

Following is a valid declaration –

```
DECLARE
  num1 INTEGER;
  num2 REAL;
  num3 DOUBLE PRECISION;
BEGIN
  null;
END;
/
```

Character Types

Following Table shows the character data types and their sub-types

S. No	Data Type & Description
1	**CHAR** Fixed-length character string with maximum size of 32,767 bytes
2	**VARCHAR2** Variable-length character string with maximum size of 32,767 bytes

3	**RAW** Variable-length binary or byte string with maximum size of 32,767 bytes, not interpreted by PL/SQL
4	**NCHAR** Fixed-length national character string with maximum size of 32,767 bytes
5	**NVARCHAR2** Variable-length national character string with maximum size of 32,767 bytes
6	**LONG** Variable-length character string with maximum size of 32,760 bytes
7	**LONG RAW** Variable-length binary or byte string with maximum size of 32,760 bytes, not interpreted by PL/SQL
8	**ROWID** Physical row identifier, the address of a row in an ordinary table
9	**UROWID** Universal row identifier (physical, logical, or foreign row identifier)

Boolean Types

The **BOOLEAN** data type stores logical values that are used in logical operations. The logical values are the Boolean values **TRUE** and **FALSE** and the value **NULL**.

However, SQL has no data type equivalent to BOOLEAN. Therefore, Boolean values cannot be used in

- SQL statements
- Built-in SQL functions
- functions invoked SQL statements

Date Time Types

The **DATE** data type is used to store fixed-length date-times, which include the time of day in seconds. Valid dates from 1st January , 4712 BC to 31st December , 9999 AD.

The default date format is set by the Oracle initialization parameter NLS_DATE_FORMAT.

For example, 'DD-MON-YY ' is the default one, which includes a two-digit number for the day of the month, first three characters of the month name, and the last two digits of the year.

Eg. 01-SEP-12.

Each DATE includes the century, year, month, day, hour, minute, and second.

The table shows the Valid Date-Time Values and its Interval Types.

Name of the Field	Date-Time Values	Interval Values
YEAR	-4712 to 9999 (excluding year 0)	Any nonzero integer
MONTH	01 to 12	0 to 11
DAY	01 to 31 (limited by the values of MONTH and YEAR, according to the rules of the calendar for the locale)	Any nonzero integer
HOUR	00 to 23	0 to 23
MINUTE	00 to 59	0 to 59
SECOND	00 to 59.9(n), where 9(n) is the precision of time fractional Seconds	0 to 59.9(n), where 9(n) is the precision of interval fractional seconds
TIMEZONE_HOUR	-12 to 14 (range accommodates daylight savings time changes)	Not applicable
TIMEZONE_MINUTE	00 to 59	Not applicable
TIMEZONE_REGION	Found in the dynamic performance view V$TIMEZONE_NAMES	Not applicable
TIMEZONE_ABBR	Found in the dynamic performance view V$TIMEZONE_NAMES	Not applicable

Large Object (LOB) Data Types

Large Object (LOB) data types refer to large data items such as text, graphic images, video clips, and sound waveforms. LOB data types allow efficient, random, piecewise access to this data. Following are the predefined PL/SQL LOB data types –

Data Type	Description	Size
BFILE	store large binary objects in operating system files outside the database.	System-dependent. Cannot exceed 4 gigabytes (GB).
BLOB	store large binary objects in the database.	8 to 128 terabytes (TB)
CLOB	store large blocks of character data in the database.	8 to 128 TB
NCLOB	store large blocks of NCHAR data in the database.	8 to 128 TB

Summary

PL/SQL Blocks contain three sections: Declaration, Execution and Exception.

PL/SQL Expressions used to retrieve particular data for the database.

PL/SQL Operators: Arithmetic Operators, Relational Operators, Comparison Operators, Logical Operators.

PL/SQL Data types: Numeric, Character, Boolean and Date Types.

Review Questions

1. Explain PL/SQL Block Structure with simple example.
2. Discuss briefly on Fundamentals of PL/SQL.
3. How you declare variables and constants in PL/SQL?
4. List out the PL/SQL Operators and Explain.
5. Give brief note on PL/SQL Datatypes.

❖❖❖❖

5

CONTROL STRUCTURES

Unit Structure

5.0 Objectives

5.1 Conditional Control

IF-THEN-ENDIF Statement,

IF-THEN-ELSE-ENDIF Statement,

IFTHEN-ELSIF-ENDIF Statement,

CASE Statement

5.2 Iterative Control:

LOOP

WHILE-LOOP

FOR-LOOP

LOOP Control Statements

5.3 Sequential Control:

GOTO Statement

NULL Statement

5.0 OBJECTIVES

This chapter makes you to understand the concepts in PL/SQL control structures

Helps to improve your coding with the help of decision making and iterative statements.

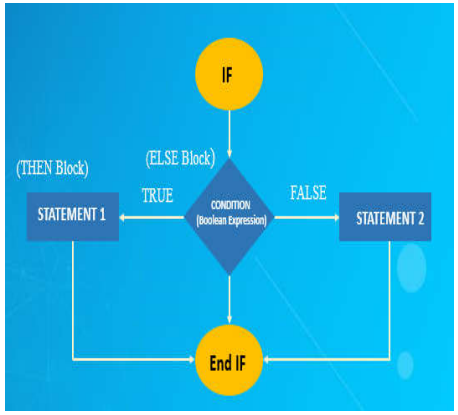
5.1 CONDITIONAL CONTROL

PL/SQL has conditional or selection statements for decision-making.

- IF....THEN....END IF.
- IF....THEN....ELSE....END IF.
- IF....THEN....ELSIF....END IF.
- CASE....END CASE.
- Searched CASE.

IF....THEN....END IF

The IF...THEN....END IF statement is also known as a simple IF statement. A simple IF statement performs action statements if the result of the condition is TRUE. Otherwise the condition is FALSE, action statements not performed and the program continues with the next statement in the block.



Syntax

IF condition(s) then

Action statements

END IF;

Above Statement show a simple IF statement with an output statement which will be performed if the day is 'SUNDAY'. The statement is skipped, if the day is not 'SUNDAY'

Example:

SQL> Declare

V_ Day Varchar2(a) := '& Day';

Begin

IF(V_ DAY ='SUNDAY') then

DBMS_ OUTPUT PUT_ LINE(SUNDAY is A HOLIDAY!);

End if;

End;

/

Enter value for day: SUNDAY

SUNDAY IS A HOLIDAY

IF...THEN...ELSE...END IF

The IF...THEN...ELSE...END IF statement is an extension of the simple IF statement. It provides action statements for the TRUE outcome as well as for the FALSE outcome.

Syntax

IF condition(s) then

Action statements 1;

Else

Action statements 2;

End if;

If the condition TRUE, action statements 1 are performed. If the condition is FALSE, action statements 2 in is ELSE part are performed. One set of statements is skipped in any case. Figure show if the entered age is 18 or older, age is displayed with string ADULT, otherwise, age is displayed with string MINOR.

Example:

SQL> Set server output on

SQL> Declare

2 V_ age number(2) := '& Age;

3 Begin

4 IF (V_ Age >=18) Then

5 DBMS_ OUTPUT. PUT_LINE('Age:'|| V_ age||-Adult');

6 else

7 DBMS_OUTPUT. PUT_LINE('Age: || V_age||. Minor')

8 End if;

9 End;

10 /

Enter value for age: 21

Age : 21-Adult

SQL>/

Enter value for age :12

Age :12-Minor

IF...THEN...ELSIF...END IF

This statement is the form of another if statement where the conditions can continue by multiple conditions in single if statement.

Syntax

IF condition(s)1 Then

Action statements 1

ElSIF condition(s)2 Then

Action statements

.....

ELSE IF condition(s) N then

Action statement N

[ELSE

Else action statements]

End if;

the word ELSIF, which does not have the last E in ELSE. ELSIF is a single word, but END IF uses the words.

Example:

SQL> Declare

2 V_ pos number (1) :=& position;

3 Begin

4 IF V_ Pos=1 then

5 DBMS_OUTPUT.PUT_LINE (‘20% increase’);

6 Elsif V_ Pos=2 then

7 DBMS_ OUTPUT.PUT_LINE (‘15% increase’);

8 Elsif V_Pos =3 then

9 DBMS_OUTPUT.PUT_LINE (‘10% increase’);

10 Elsif V_Pos=4 then

11 DBMS_OUTPUT.PUT_LINE (‘%% increase’);

12 Else

13 DBMS_OUTPUT.PUT_LINE (‘NO increase’);

14 End if;

15 End;

16 /

Enter value for position :2

15% increase

CASE

In Previous chapter, the features case and searched case comes under expressions topic. Now we see the syntax and how to use in PL/SQL Block.

- The CASE Statement is an alternative to the IF...THEN...ELSIF...END IF statement.
- This statement starts with keyword CASE and ends with the keywords END CASE.
- The body of the statement contains WHEN clauses with values or conditions and action statements.
- When a WHEN condition evaluates to TRUE its actions statements are executed.

Syntax

CASE[Variable- name]

WHEN condition1/value THEN action- statement1;

WHEN condition1/value 2 THEN action- statement 2;

.....

WHEN condition1/value N THEN action- statement N;

ELSE action- statement;

END CASE

Example:

SQL> Declare /* Example of case */

2 V_ num number := & Any-num;

3 V_Res number;

4 Begin

5 V_Res := Mod (V_ num ,2);

6 CASE V_Res

7 When 0 then DBMS_OUTPUT.PUT_LINE (V_num||’is even’);

8 ELSE DBMS_OUTPUT.PUT_LINE (V_ num|| ‘is odd’);

```
9 end case;
10 End;
11 /
Enter value for any- num : 5
5 is odd
```

SEARCHED CASE

A statement with a value is known as a CASE statement and a statement with condition is known as a searched CASE statement. This statement does not use variable- name as a selector but a CASE uses variable- name as a selector.

Example:

```
SQL>
Declare
2 V_ num number :=& Any- num;
3 Begin
4 case
5 When mod (V_ num2) =0 Then
6 DBMS_OUTPUT.PUT_LINE (V_ num|| 'is odd');
7 else
8 DBMS_OUTPUT.PUT_LINE (V_ num||' is odd');
9 End case;
10 End;
11 /
Enter value for any num :5
5 is odd.
```

NESTED IF

The nested IF statement contains an IF statement within another IF statement. If the condition of the outer IF statement is TRUE, then the corresponding IF statement is performed.

Consider the following conditions.

- Male 25 or over
- Male under 25

- Female 25 or over
- Female under 25.

Example:

```
SQL> Declare
2 V_ Gender char :='& sex';
3 V_age number (2) :='& Age';
4 V_char number(3,2);
5 Begin
6 IF (V_ Gender ='19') then /* Male*/
7 IF (V_age>=25) then
8 V_charge:=0.05;
9 Else
10 V_charge :=0.01;
11 End if;
12 Else /* Female */
13 IF (V_age>=25) then
14 V_charge :=0.06;
17 End if;
18 End if;
19 DBMS_OUTPUT. PUT_LINE (Gender :|| V_Gender);
20 DBMS_ OUTPUT.PUT_LINE ('Age:='To-char (V_age));
21 DBMS_OUTPUT.PUT_LINE ('SURCHARGE:'|| To-
char(V_charge));
22 End;
23 /
Enter value for sex:F
Enter value for age :18
Gender: F
Age: 18
Surcharge: 06
```

5.2 ITERATIVE CONTROL

In general, statements are executed sequentially: The first statement in a function is executed first, then followed by the second, next and so on. Some situation when you need to execute a block of code several times. For this execution programming languages provide control structures that allow for more complicated execution paths.

A loop statement used to execute a statement or group of statements multiple times. A loop repeats a statement or a series of statements a specific number of times as defined by the programmer.

Types of Looping Statements

- Basic loop
- WHILE loop
- FOR loop

Each loop has their own syntax and works differently.

BASIC LOOP

A basic loop is a loop that is performed repeatedly. Once a loop is entered all statements in the loop are executed. Once the bottom of the loop is reached control shift back to the top of the loop. The loop will continue infinitely is a logical error in programming. The only way to terminate a basic loop is by adding an EXIT statement inside the loop.

Syntax

Loop

Looping statement 1;

Looping statement 2;

.....

Looping statement N;

EXIT [When condition];

End loop;

The EXIT statement in a loop could be independent statement. We can also add a condition with the optional WHEN clause that will end the loop when the condition becomes true.

Example:

EXIT WHEN V_count>10;

The condition is not checked at the top of the loop, but it is checked inside the body of loop. The loop is performed at least once, because the

condition is tested after entering the body of the loop is known as Post_ test loop.

Example:

SQL>Set Serveroutput on

SQL> Declare

2 V_ count number(2);

3 V_ sum number(2):=0;

4 V_Avg number(3,1);

5 Begin

6 V_ count : =1;

7 Loop

8 V_sum := V_sum + V_count

9 V_count := V_count +1;

10 Exit when V_ count >10;

11 End loop;

12 V_Avg := V_sum 1(V_ count -1);

13 DBMS-OUTPUT.PUT-LINE (Average of 1 to 10 1&|| To- char (V_Avg));

14

15 End;

16 /

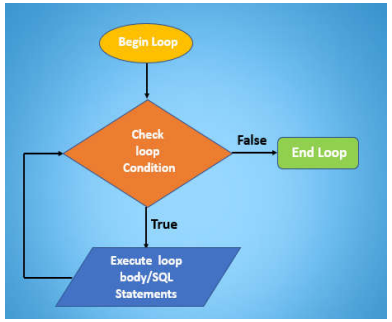
Average of 1 to 10 1& 5.5

SQL>

WHILE LOOP

The WHILE loop is an alternative to the basic loop. It is performed as long as the condition is true. This terminates when the condition become false. If the condition is false in beginning, then the loop is not performed at all.

The WHILE loop does not need an EXIT statement to terminate.



Syntax

WHILE condition loop

Looping statement 1;

Looping statement 2;

.....

Looping statement n;

End loop;

Example:

```
SQL> Declare
2  V_ count        number(2);
3  V_ sum          number(2):=0;
4  V_Avg           number(3,1);
5  Begin
6  V_ count : =1;
7  While V_ count < = 10 Loop
8  V_sum := V_sum + V_count
9  V_count := V_count +1;
10  End loop;
11 V_Avg := V_sum 1( V_ count -1);
12  DBMS-OUTPUT.PUT_LINE (Average of 1 to 10 1&|| To- char (V_Avg));
13  End;
14  /
```

Average of 1 to 10 1& 5.5

Basic loop & While Loop

Basic Loop	**While Loop**
It is performed as long as the condition is false.	It is performed as long as the condition is true.
It testes the condition inside the loop (Post-test loop).	It checks condition before entering the loop (Pre-test Loop).
It is Performed at least one time.	It is performed zero or more times.
It needs the EXIT statement to terminate	no need for an Exit statement.

FOR LOOP

The For loop is the simplest loop. We do not have to initialize, test and increment/ decrement the loop control variable separately. The counter used here is implicitly declared as an integer and it is destroyed on the loop's termination. It may be used within the loop body in an assignment statement as a target variable.

Syntax

FOR Counter W(Reservse) lower...upper loop

Looping statement 1

Looping statement 2

.....

Looping statement N

End loop;

Counter Increment/ Decrement

The counter varies from the lower value to the upper value incrementing by one with every loop execution. The counter starts with higher value and decrementing by one with every loop execution. To reverse the order the keyword Reverse is used to make higher to lower value.

```
SQL> Declare
2  V_ count  number(2);
3  V_ sum  number(2):=0;
```

```
4  V_Avg   number(3,1);
5  Begin
6  For V_ count in 1.. Loop
7  V_ sum :=V_ sum +V_ sum;
8  End loop;
9  V_ Avg := V_sum/10;
10  DBMS-OUTPUT.PUT-LINE (Average of 1 to 10 1&|| To- char (V_Avg));
11
12  End;
13  /
```

Average of 1 to 10 1& 5.5

NESTED LOOP

We can use a loop within another loop. Loop can be nested to many levels, when the inner loop ends it does it does not automatically end the outer loop enclosing it. We can quit the outer loop by label each loop inside the inner loop and then using the EXIT statement. The loop labels use the same naming rules as those used for identifies.

The label is enclosed using << and >> two pairs of angel brackets.

Eg

```
<< outer- loop>>
Loop
EXIT WHEN condition;
<< inner- loop>>
Loop
.....
EXIT outer- loop WHEN condition; /* exit outer-loop*/
EXIT WHEN condition        /* exit inner- loop*/
.....
End Loop inner- loop   /* label optional*/
.....
End loop outer-loop     /* label optional */
```

Loop Control Statements

Loop control statements change execution from its normal sequence.

PL/SQL supports the following control statements.

S.No	Control Statement & Description
1	**EXIT** statement The Exit statement completes the loop and control passes to the statement immediately after the END LOOP.
2	**CONTINUE** statement Causes the loop to skip the remainder of its body and immediately retest its condition prior to reiterating.
3	**GOTO** statement Transfers control to the labelled statement. Though it is not advised to use the GOTO statement in your program.

5.3 SEQUENTIAL CONTROL

GOTO Statement

The GOTO statement allows you to transfer control to a labeled block or statement.

The syntax of the GOTO statement:

GOTO label_name;

The label_name is the name of a label that identifies the target statement. In the program, you surround the label name with double enclosing angle brackets as shown below:

<<label_name>>;

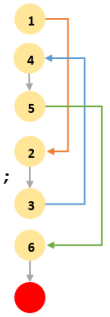
When PL/SQL executes a GOTO statement, it passes control to the first executable statement after the label.

```
BEGIN
   GOTO second_message;

  <first_message>
   DBMS_OUTPUT.PUT_LINE('Hello');
   GOTO the_end;

   <second_message>;
   DBMS_OUTPUT.PUT_LINE('PL/SQL GOTO Demo');
   GOTO first_message;

   <the_end>
   DBMS_OUTPUT.PUT_LINE('and good bye...');
END;
```



This code will execute like

- GOTO second_message statement is encountered firsr, therefore, the control is passed to the statement after the second_message label.
- GOTO first_message is encountered in second, so the control is transferred to the statement after the first_message label.
- Next , GOTO the_end is reached, hence the control is passed to the statement after the the_end label.

The output is:

PL/SQL GOTO Demo

Hello

and good Bye...

NULL Statement

The NULL statement is a NULL keyword followed by a semicolon (;). The NULL statement does nothing except that it passes control to the next statement.

The NULL statement is useful to:

- **Improve code readability**
- **Give target for a GOTO statement**
- **provide placeholders for subprograms**

Improving code readability

This code sends an email to employees whose job titles are SalesRepresentative.

```
IF jobtitle = 'SalesRepresentative' THEN

   send_email;

END IF;
```

If the employees job title are not SalesRepresentative then do nothing, this logic is not explicitly mentioned in the code.

An ELSE clause that consists of a NULL statement to clearly state that no action is needed for other employees.

```
IF jobtitle = 'SalesRepresentative' THEN

      send_email;

ELSE

      NULL;

END IF;
```


Give target for a GOTO statement

When using a GOTO statement, you need to specify a label followed by at least one executable statement.

This example has a GOTO statement to quickly move to the end of the program if no further processing is required:

```
DECLARE

 b_status BOOLEAN

BEGIN

 IF b_status THEN

      GOTO end_of_program;

 END IF;

 -- further processing here

 -- ...

 <<end_of_program>>

 NULL;

END;
```

Error will occur, if there is no NULL statement after the end_of_program label.

Provide placeholders for subprograms

The following example creates a procedure named apprreq - request_for_approval that doesn't have the code in the body. PL/SQL requires one executable statement in the body of the procedure to compile successfully. we add a NULL statement to the body as a placeholder. Later you can fill the real code.

```
CREATE PROCEDURE apprreq ( cusmer_id NUMBER )

AS

BEGIN

 NULL;

END;
```

Now, you have a good understanding of PL/SQL NULL statement and how to apply it in your daily programming tasks.

Summary

PL/SQL Control Structures has three type: Condition Control, Iterative Control and Sequence Control.

In Condition Control if and case statements are used to make decisions and perform executions.

In Iterative Control, three types loops used with different syntaxes.

In Sequence Control GOTO and NULL Statements are performed.

Review Questions

1. Write in detail about Conditional Control Structures.
2. Discuss the different Loop statements available in PL/SQL.
3. Explain the usage of sequence control statements with simple example.
4. How do you write a PL/SQL block for decision making purpose? Give Example.
5. How you come out of infinite Loop?

Unit 2: Simple PL/SQL Programs

1. Add Two Numbers

```
Declare
Var1 integer;
Var2 integer;
Var3 integer;
Begin
Var1:=&var1;
Var2:=&var2;
Var3:=var1+var2;
Dbms_output.put_line(var3);
end;
```

2. Prime Number

```
Declare
n number;
i number;
flag number;
 begin
i:=2;
flag:=1;
n:=&n;
for i in 2..n/2 loop
  if mod(n,i)=0 then
flag:=0;
exit;
end if;
      end loop;
      if flag=1 then
dbms_output.put_line('prime');
else
dbms_output.put_line('not prime');
            end if;
         end;
```


3. Factorial Number

```
declare
n number;
fac number:=1;
i number;
         begin
n:=&n;
for i in 1..n
loop
fac:=fac*i;
end loop;
dbms_output.put_line('factorial='||fac);
           end;
```

4. Print a Table of Number

```
declare
n number;
i number;
begin
n:=&n;
for i in 1..10
loop
dbms_output.put_line(n||' x '||i||' = '||n*i);
end loop;
end;
```

5. Reverse of a number

```
declare
n number;
i number;
rev number:=0;
r number;
begin
n:=&n;
while n>0
loop
r:=mod(n,10);
rev:=(rev*10)+r;
n:=trunc(n/10);
end loop;
dbms_output.put_line('reverse is '||rev);
end;
```


6. Fibonacci Series

```
declare
first number:=0;
second number:=1;
third number;
n number:=&n;
i number;
begin
dbms_output.put_line('Fibonacci series is:');
dbms_output.put_line(first);
dbms_output.put_line(second);
for i in 2..n
loop
third:=first+second;
first:=second;
second:=third;
dbms_output.put_line(third);
end loop;
end;
```

7. Check number is odd or even

```
declare
n number:=&n;
begin
if mod(n,2)=0
then
dbms_output.put_line('number is even');
else
dbms_output.put_line('number is odd');
end if;
end;
```

8. Palindrome Number

```
declare
   n number;
   m number;
   rev number:=0;
   r number;
begin
   n:=12321;
   m:=n;
   while n>0
   loop
      r:=mod(n,10);
      rev:=(rev*10)+r;
      n:=trunc(n/10);
   end loop;
   if m=rev
   then
      dbms_output.put_line('number is palindrome');
   else
      dbms_output.put_line('number is not palindrome');
   end if;
end;
```

9. Swap Two Numbers

```
declare
   a number;
   b number;
   temp number;
begin
   a:=5;
```



```
   b:=10;
   dbms_output.put_line('before swapping:');
   dbms_output.put_line('a='||a||' b='||b);
   temp:=a;
   a:=b;
   b:=temp;
   dbms_output.put_line('after swapping:');
   dbms_output.put_line('a='||a||' b='||b);
end;
```

10. Greatest of three numbers

```
declare
a number:=10;
b number:=12;
c number:=5;
begin
dbms_output.put_line('a='||a||' b='||b||' c='||c);
if a>b AND a>c
then
dbms_output.put_line('a is greatest');
else
if b>a AND b>c
then
dbms_output.put_line('b is greatest');
else
dbms_output.put_line('c is greatest');
end if;
end if;
end;
/
```