Section – B

Introduction to PL/SQL :-

PL/SQL is a procedural language. It is extension to the non-procedural SQL. Oracle introduced the PL/SQL to overcome some limitations of SQL and to provide complete programming solution. PL/SQL is the block structur programming language. It is the superset of SQL with the use of SQL, user can perform only basic operations like data manipulation (insertion, deletion, updation, selection), but PL/SQL extend the capability of the SQL by adding control structure, functions, procedures, triggers, cursors etc. In other words PL/SQL combines the data manipulation power of SQL and procedural power of PL/SQL. PL/SQL provide different control structure like if then else, while loop, for loop, declaration of variable, declaration of constant etc. PL/SQL is not a case sensitive language.

Structure of PL/SQL :-

PL/SQL is a block structure programming language. It has many procedural capabilies like branching, looping, error handling, functions, procesures etc. PL/SQL programs are written in different blocks. Block is a ~~statement~~ smallest piece of SQL and PL/SQL statements. Related statements are written to a specific block. PL/SQL program consists basically four sections :- which are given below :-

```
DECLARE
                (Optional section)      ← used for variable and constant declaration
BEGIN
                (Compulsary section)    ← for SQL and PL/SQL statements.
EXCEPTION
                (optional section)
END;
```

Scanned with CamScanner

① DECLARE Section :- Declare section is used to declare the variables & constants. It is the optional section, it means this section can be omitted if there is no variable declaration in the program. Declare section begins with the DECLARE keyword. This section is also used to declare the functions, procedures etc. Once the variable, constant or other objects are declared, they can be used further in the BEGIN section.

② BEGIN Section :- Begin section is also called executable section. This section is the compulsary section. It mean user cannot omit this section from program, otherwise error will be produced. Begin section starts with the BEGIN Keyword. All the SQL and PL/SQL statements will be written in the BEGIN block. All the variable that are declared in DECLARE section are used in the BEGIN section. The definition of functions or procedures or Triggers are written in BEGIN section. All the looping, branching statements are written in BEGIN section.

③ EXCEPTION Section :- This section is also known as error handling section. It start with the EXCEPTION keyword. This section deals with the handling of errors that arise during the execution of the program. This is a optional section.

④ END Section :- END section is used to mark the ending of PL/SQL block. A semicolon (;) should be placed after the END Keyword. ~~The~~

Scanned with CamScanner

Example for describing the structure of PL/SQL is given below:-

```
DECLARE
        a number(10);
        b number(10);
        c number(10);
BEGIN
        a:= &Enter a;
        b:= &Enter b;
        c:= a+b;
        dbms_output.put_line('Sum of 2 numbers' || c);
END;
```

⇒ **Difference b/w SQL and PL/SQL**

SQL	PL/SQL
① SQL is the non-procedural language.	① PL/SQL is a procedural language.
② SQL does not support PL/SQL.	② PL/SQL support SQL.
③ SQL is the subset of the PL/SQL.	③ PL/SQL is the superset of the SQL.
④ SQL does not allow the user to create their own functions, procedure or packages.	④ PL/SQL allow the user to create their own functions, procedures, packages and triggers etc.
⑤ SQL does not have any procedural capability. It means in SQL there is no way to specify looping & controling statements	⑤ PL/SQL provide all the procedural capabilities
⑥ SQL statements are passed to Oracle engine one at a time. Therefore each time for each statement, a call is made to the Oracle engine and oracle engine	⑥ In PL/SQL the entire PL/SQL program are send to the Oracle engine therefore one call is made and all the statements in the program are executed. Result is fast execution or

Scanned with CamScanner

SQL	PL/SQL
open and close the resource every time so resulting in slow processing & increase in network traffic.	this reduce the network traffic.
(7) SQL does not allow to declare the variable to store the intermediate result.	(7) PL/SQL support the declaration of variables so that we can store the values or intermediate result for further use

Advantages of PL/SQL :-

(1) **Support for SQL :-** PL/SQL allow the use of all the SQL commands as well as all the SQL functions, operators and data types

(2) **Better Performance :-** PL/SQL improve the performance by reducing the number of call to the Oracle engine. In SQL Oracle engine execute SQL statements one at a time. Therefore each time a call is made to the oracle engine, result in slow processing but on the other hand with PL/SQL, entire program is sent to the Oracle engine at a time, result in better performance and faster execution.

(3) **Block Structure :-** PL/SQL is a block structure language. It mean the entire PL/SQL program is divided into logical blocks. Each block can contain any number of statements. PL/SQL program is basically divided into four blocks. Variables are declared into DECLARE Block. Executable statements are written in the BEGIN Block.

Scanned with CamScanner

Error handling statements are written in the execution block and END block specify the ending of the program.

(4) Control Structure :- Control structure are most important part of the PL/SQL. Control statements control the flow of the program. It means control statements decide which are the statements are to be executed and how many times particular statement will be executed. PL/SQL provide different types of control statements like branching and looping statements. (if else & for, while loop etc)

(5) Error Handling :- This is the main advantage of PL/SQL. PL/SQL handles errors or exceptions during the execution of a PL/SQL program. Once an exception is caught, specific actions can be taken depending upon the type of the exception. It allow the user to display the user friendly message when the errors are encountered.

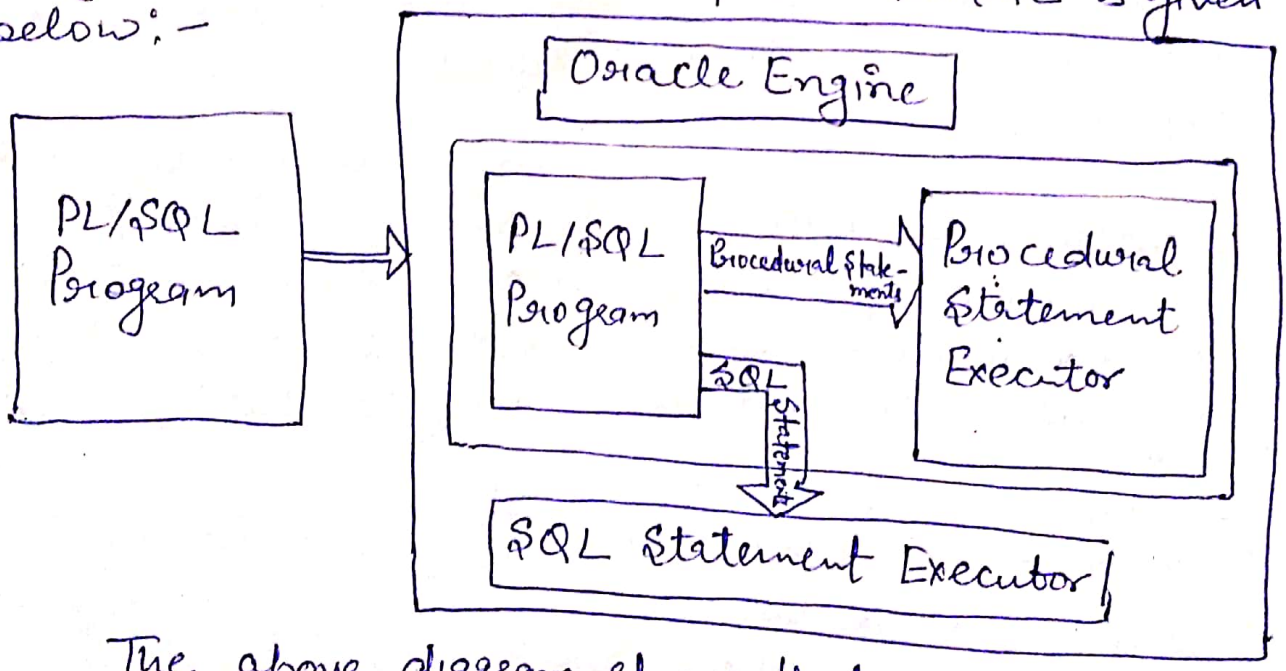
(6) Portability :- PL/SQL is portable language. Programs written in the PL/SQL can be run to any computer hardware or operating system on which Oracle runs. Once you write a program in PL/SQL, it can be used in any environment without any change at all.

(7) Modularity :- Modularity means divide an application or a process into ~~meaning~~ manageable, well defined modules. PL/SQL allows process to be divided into different modules such as procedures and functions, called subprograms

Scanned with CamScanner

Architecture of PL/SQL :-

When the PL/SQL technology is used, oracle engine accept the entire program and send it to PL/SQL engine. PL/SQL engine is the small program that reside in the oracle engine. PL/SQL engine execute the PL/SQL block. It only execute procedural statements and send the SQL statements to the SQL statement executer. The call to the oracle engine need to be made only once to execute any number of statements. This thing increase the speed of processing because only one call is made to the oracle engine to execute all the statements contain in the PL/SQL program. The architecture of the PL/SQL is given below:-



The above diagram shows that when the user will execute the PL/SQL program, one call is made to the oracle engine, which send this program to the PL/SQL engine. PL/SQL engine is the small engine reside in the Oracle engine. After receving the PL/SQL program, PL/SQL engine will filter the statements and send the procedural statements to the procedure statement executor and send SQL statements to the SQL statement executor then the result of the program is send back to the user by the Oracle engine.

Scanned with CamScanner

⇒ Fundamentals of PL/SQL :-

① Character Set :-

PL/SQL programs are written as lines of text using a specific set of characters. The PL/SQL character set is includes:-

⇒ The uper and lower case letters (A...Z, a..z)
⇒ The numbers (0...9)
⇒ Tabs, spaces, and carriage returns
⇒ The special symbols like : ( ) + - * / < > ! : ; { } [ ] etc.

PL/SQL is not case sensitive so lower case letters are equivalent to corresponding uper-case letters.

② Keywords :-

Keywords are the reserved words that have a special meaning and that meaning cannot be changed by the user. Keywords cannot be used as identifiers. All the Keywords have predefined meaning and any attempt to redefine the meaning of Keyword will produce an error. There are many keywords that are used in PL/SQL :-

BEGIN, DECLARE, END, EXCEPTION, WHILE, IF, ENDIF, LOOP, GOTO, ENDLOOP, FOR, DO etc.

Reserved words are generally written in upper case to promote readability, but it can be written in lower or mixed case.

③ Identifiers :-

Identifiers are the user defined names which are given to the variable name, function name, procedure name, trigger name etc. When define the identifiers, some rules should be follow :-

(a) It must start with a letter.
(b) Keywords cannot be used as a Identifier name.

Scanned with CamScanner

(c) Identifier names are not case sensitive.
(d) White spaces are not allowed.
(e) Underscore can be used to combine two words.
eg. Roll_no
(f) In PL/SQL identifiers can have 30 characters.

4. Comments :-

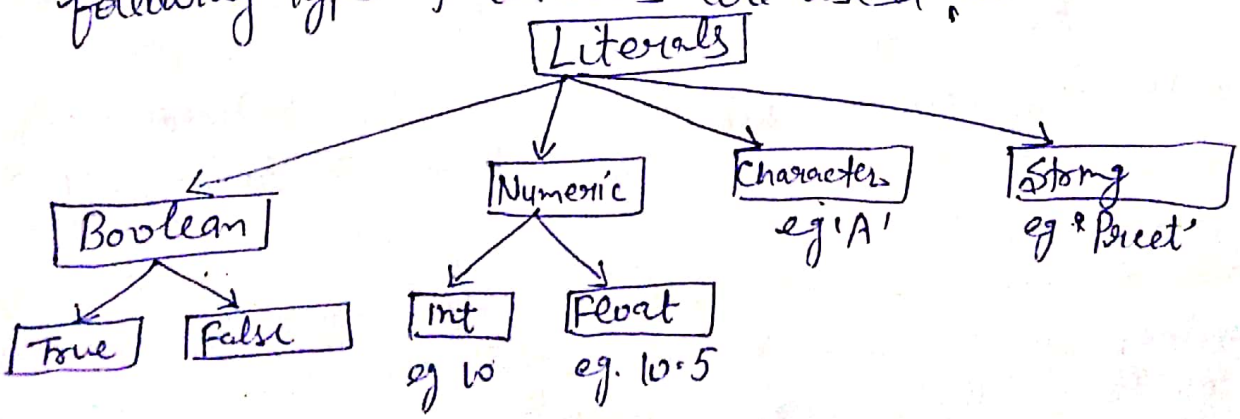
Comments are basically used to make the program understandable by other users. Comments provide the documentation to the program. In PL/SQL two types of comments are used :-

(a) Single line comments (--)
(b) Multiple line comments (/* ..... */)

Single line comment begins with the double hyphen (--) It means the entire line will be treated as a comment. The multiple line comment begin with /* and end with the */

5. Literals :-

Literals are the numeric or character values that is stored in the variable. This value can be manipulated using operators. In PL/SQL following types of literals are used :-



⇒ **Numeric Literals :-**

Numeric literals store the positive or negative value with or without decimal point. A value without a decimal point is called integer literals. For example 10, 20, -10 etc.

Scanned with CamScanner

Any value with a decimal point is called floating point literals. For example 10.5, 2.5, -2.5 etc

⇒ **Character Literals :-** Character literals consists only single character inclosed in a single quotes. eg 'A', 'B' etc.

⇒ **String Literals :-** String literals consist collection of one or more characters enclosed with in single quotes eg. 'Preet'

⇒ **Boolean Literals :-** Boolean literals consists the predefined value, TRUE, FALSE or NULL.

⑥ Constants :-

Constant refers to the fixed value that do not change during the execution of the program. Constant can be numeric, character, string, float etc. In PL/SQL constants are declared by using the CONSTANT keyword. Any attempt to change the value of constant will produce an error.

eg

```
DECLARE
        a constant number(10) := 10;
        c number(10);
BEGIN
        c := a+10;
dbms_output.put_line('Value of c is '||c);
END;
/
```

⑦ Variables :-

Variables are the names given to the memory location to store the particular value. Unlike constant variable does change their value during the execution of the program. It means variable take different value at different time during the execution of the program. Variable must be declared before they are used.

Scanned with CamScanner

The declaration of the variable includes the name of the variable ~~include the~~ followed by the datatype and size.

Syntax :- <variable name> datatype (size);
eg a number(10);

⇒ Values can be assigned to a variable through three ways :-

(a) With the use of assignment operator.
(b) With the use of substitution variable.
(c) With the use of select into clause.

⇒ (a) With the use of assignment operator :-

In this method assignment operator is used to assign a value to a variable. Initial value can be assigned to a variable by using the assignment operator (:=)

eg a number(10) := 20;

⇒ (b) With the use of substitution variable :-

In this method the value is assigned to a variable at a runtime by using the substitution variable.

eg.

```
DECLARE
        a number(10);
BEGIN
        a := &a;
```

⇒ (c) With the use of select into clause :-

In this method select into command is used to insert the value to a variable. This statements are basically used when user wants to retrieve the contents of the table and want to assign

Scanned with CamScanner

these contents to the variable.

eg.
```
DECLARE
        s number (10);
        e varchar2 (20);
BEGIN
    select sal, ename into s, e from emp
            where emp.no = 7788;
dbms_output.put_line('Salary of employee is "||s);
dbms_output.put_line('Name of employee is "||e);
END;
/
```

(8)

Attribute of variable :-

Attribute of a variable allow the user to refer to the datatype and size from the other table. There are two types of attribute supported by the PL/SQL :-

1) %type :- To avoid the datatype and size conflict b/w the variables and columns name %type attribute is used. In other words the variable that deals with the column of the table should have the same datatype & size as the column itself. %type attribute is used when declaring a variable that reffers to the column of the table. When using %type, user need to know the name of the column and name of the table to which the variable will refer. The advantage of using this attribute is that whenever the type & size of the column is changed it will automatically reflect to the variable declaration.

eg.
```
DECLARE
        s emp.sal%type;
BEGIN
    select sal into s from emp where empno=7788;
    dbms_output.put_line('Salary of Employee is' || s);
END;
```

Scanned with CamScanner

(2) % rowtype :-

%rowtype attribute represent a row in the table. %rowtype attribute refer to the entire row from the table. eg. emp table contain different columns like empno, ename, job, sal, deptno etc. And user want to manipulate the value in each column of a row by using one refrence variable. This refrence variable is declared by using %rowtype.

eg.

```
DECLARE
        a emp %rowtype;
BEGIN
  select * into a from emp where empno=7788;
      a.sal=a.sal+1000;
  insert into emp values (a.sal, a.ename);
  End;
```

(9)

Scope and Visibility of the variable :-

The scope of the variable is the portion of the program in which the variable can be access. In other words scope also refer to the life of the variable. For a PL/SQL variable, the scope of the variable is from the declaration to the end of the block. When the variable goes out of the scope it cannot be accessed. On the other hand visibility of the variable is the portion of the program where the variable can be accessed without having to qualify refrence. If the variable is outof scope it will not be visible. Visibility is also within the scope. The redeclaration of the variable in the local block will hide the globle declaration of the variable because local variable have higher priority than globle variable

Scanned with CamScanner

```
DECLARE
        a number(10) := 100;
        b number(10);
BEGIN
        <<inner>>
        DECLARE
                a number(10) := 200;
                c number(10);
        BEGIN
                c := a+20;
                dbms_output.put_line('Inner Block value'||c)
        END;
        b := a+20;
        dbms_output.put_line('Outer Block value'||b);
END;
```

Here b, c and local a is visible

Here b and globle a is visible

Scope of Globle a & b

Scope of local a & c

Scope & Visibility of Variable

(10) Operators :-

Operators are the special symbol that specify what operations to be performed on one or more operands. Some operations use two operands called binary operator. Some operators use ~~and~~ one operands called unary operator. Oracle provide various operator in PL/SQL

(a) Arithmetic Operator :-

Arithmetic operators are used to perform addition, subtraction, multiplication division etc. PL/SQL provide another arithmetic operator exponential used as a power function

Operator	Meaning
+	Addition
−	Subtraction
/	Division
*	Multiplication
**	Exponential

Scanned with CamScanner

(b) Relational Operator :-

Oftenly two or more variables are compared with another and depending upon their relation some decision are taken. The comparison will be done with the help of relational operator. The expression that contain the relational operator is called relational expression. The value of the relational expression is either True or False.

Operator	Meaning
>	Greater than
<	Less than
>=	Greater than equal
<=	Less than equal
=	Equality
!=	Not equal

(C) Logical Operator :-

If the user want to test more than one condition then the logical operators are used. The expression that contain the logical operator is called logical expression. The value of logical expression will be true or false

Operator	Meaning
AND	Logical and
OR	Logical or
NOT	Logical not

(d) Other Operator :-

PL/SQL provide some other operator that are also used in PL/SQL. These are :-

Operator	Meaning
Between	For ranging
In	for set of values
||	to concatenate two strings
Is NULL	for checking missing value

Scanned with CamScanner

SQL within PL/SQL :-

PL/SQL is the database language that support SQL. It is also called the super set of SQL. Whenever any data manipulation have to perform in the PL/SQL, then the SQL commands are used. All the SQL commands are not supported by PL/SQL. It support only DML statements & TCL statements of SQL. The insert, update, delete statements are used with any restriction but when the select command is used, into clause must be specified with the select command. The select command can return only one row at one time. If it will return more than one row the error will be generated. When the select statement is used in PL/SQL to retrieve the row from a table it will use following syntax :-

select <columnname> into <variablename> from <tablename> where <condition>;

eg.
select sal into s from emp where empno =7788;

In the above syntax into clause is used to store the value of the column from a table into the PL/SQL variables.

⇒ The insert statement is used to add new row into the table. The syntax for writing the insert statement in PL/SQL is similar to the syntax in SQL

⇒ Similarly delete or update commands are used to delete the rows or update the data and these commands have similar syntax as in SQL.

⇒ All functions of SQL except group functions are used in PL/SQL.

⇒ PL/SQL also support the datatypes of SQL.

⇒ So we can say that PL/SQL is the superset of the SQL.

Scanned with CamScanner

Writing a PL/SQL code :-

PL/SQL program is written by using text editor. The PL/SQL program is written into logical block. The DECLARE block is used to declare the variables. BEGIN block is used to write the executable statements. END Block specify the ending of the program. Comments can be placed within PL/SQL program by using (--) for single line comment and /* - - - - */ for multiple lines comments.

⇒ dbms_output.put_line function is used to produce the output

⇒ when the program is completed @ sign is used to execute the program.

eg. SQL> @ filename.SQL

⇒ To enable the server output, the set serveroutput on command must be given at the SQL prompt before the execution of any program.

eg. SQL> set serveroutput on;

eg

```
DECLARE
        a number (10);
        b number (10);
        c number(10);
BEGIN
        a:=&a;
        b:=&b;
        c:=a+b;
        dbms_output.put_line('Sum of two no's '||c);
END;
/
```

Output :- SQL > set serveroutput on;
SQL> @ filename;
Enter value of a := 5
Enter value of b = 6
Sum of two no's = 11

Scanned with CamScanner

Control Statements in PL/SQL :-

PL/SQL is a set of statements which are normally executed sequentialy ~~by~~ but there can be a number of situations where a user want to change the flow of the execution based to certain conditions or repeat a group of statements again and again. until certain conditions are specify. All these can be possible with the use of control statements. PL/SQL has three types of control statements :-

1. Branching Control Statements
2. Looping & Iterative Control Statements
3. Jumping Control Statements

1. Branching Control Statements :-

Branching control statements are used to change the flow of execution based on certain conditions. PL/SQL has a different types of branching control statements :-

(a) IF-THEN-END IF
(b) IF-THEN-ELSE-END IF
(c) IF-THEN-ELSEIF-END IF
(d) Nested IF-THEN-ELSE-END IF

(a) IF-THEN-END IF :-

Simple IF-THEN-END IF statement is a decision making statement that is used to change the flow of the execution. The syntax is :-

```
IF condition THEN
    statement set;
END IF;
```

Initialy the condition will be evaluated, if the condition is evaluated to True only then the statement set will be executed. If the condition become false, nothing will happen. The statement set can contain a single statement or a group of statement.

Scanned with CamScanner

eg.

```
DECLARE
        a number (10);
        b number (10);
BEGIN
        a:=&a;
        b:=&b;
    IF a>b THEN
    dbms_output.put_line ('A is greater');
END; ENDIF;
```

(b) IF-THEN-ELSE-END IF :-

IF-THEN-ELSE-END IF is powerful decision making statement. It is basically a two way decision making statement. It allow the Oracle to evaluate the condition first and depending upon their value transfer the control to a particular block. The syntax for IF-THEN-ELSE-ENDIF is :-

```
IF condition THEN
        Statement set 1;
ELSE
        Statement set 2;
END IF;
```

In the above syntax if the condition is True the statement set 1 will be executed otherwise the statement set 2 will be executed.

eg

```
DECLARE
        a number(10);
        b number(10);
BEGIN
        a:=&a;
        b:=&b;
IF a>b THEN
dbms_output.put_line ('A is greater');
ELSE
dbms_output.put_line ('B is greater');
END IF;
END;
```

Scanned with CamScanner

(c) IF-THEN-ELSE IF-ENDIF :-

ELSEIF ladder is the another way of putting the If's together when multipal decisions are involved. In multiple decision a chain of ifs are associated with each other. It has the following syntax :-

```
IF condition1 THEN
        Statement set 1;
ELSE IF condition2 THEN
        Statement set 2;
    :
ELSE
        Default statement;
END IF;
```

In the above syntax the conditions are evaluated from top to downward. As soon as the true condition is found, the statement associated with it will be executed. Initially condition1 is evaluated if it become false then condition2 is evaluated. If it become false then condition3 is evaluated. When all the conditions become false, the final else block will be executed.

eg.

```
DECLARE
        p number(10);
BEGIN
        p:=&p;
    IF p>=60 THEN
    dbms_output.putline ('A Grade');
    ELSEIF (p>=50) and (p<60) THEN
    dbms_output.put_line ('B Grade');
    ELSEIT (p>=40) and (p<50) THEN
    dbms_output.put_line ('C Grade');
    ELSE
     dbms_output.put_line ('D Grade');
    ENDIF;
  END;
```

Scanned with CamScanner

(d) Nested IF-THEN-ELSE-END IF :-

Nested IF-THEN-ELSE-END IF statement means when the entire if else statement are written either in IF block or in ELSE block or in both. Syntax is :-

```
IF condition1 THEN
        IF condition2 THEN
            Statement set 1;
        ELSE
            Statement set 2;
        END IF;
ELSE
        IF condition3 THEN
            Statement set 3;
        ELSE
            Statement set 4;
        END IF;
END IF;
```

Initially in the above syntax the condition1 will be checked. If the condition1 is true then the condition2 will be checked. If the condition2 become true then the Statement set 1 otherwise statement set 2 will be executed. If the initially condition1 become false then control goes to the else block and condition3 is evaluated. If the condition3 become true then the statement set 3 will be executed otherwise statement set 4 will be executed.

eg.

```
DECLARE
        a number (10) := 10;
        b number (10) = 20;
        c number(10) = 5;
BEGIN
        IF a > b THEN
            IF a > c THEN
                dbms_output.put_line ('A is greatest');
            ELSE
                dbms_output.put_line ('C is greatest');
        ELSE
            IF b > c THEN
                dbms_output.put_line('B is greatest');
            ELSE
                dbms_output.put_line ('C is greatest');
END; END IF;
```

Scanned with CamScanner

2) Looping Control Statement :-

The iterative control statement in PL/SQL are used to execute the group of statements or single statement repeatedly. It means looping control statement enable the PL/SQL to execute the same code again and again until the specified condition is satisfied. In PL/SQL there are three types of control statements are used.

(a) Simple loop statement / Basic loop
(b) While loop
(c) For loop

(a) Simple Loop :-

The most basic type of loop statement is simple loop statement. It include sequence of statement b/w the LOOP & END LOOP keyword. This loop is also useful to guarantee that the body will be executed atleast one time, when the exit condition is written at the end. The basic syntax of simple loop is given below :-

```
LOOP
    EXIT when <condition>
        Statement set;
END LOOP;
```

In the above syntax, with each iteration of the loop, the sequence of statements are executed, then the control goes to the top of the loop where the exit condition will be checked again. If the exit condition satisfy the loop complete, immediately control passes to the next statement after the END LOOP. On the other hand if the EXIT condition fail, the sequence of statements will be executed again :-

eg. -- Sum of first 10 no's.

```
DECLARE
        i number(10) := 1;
        s number(10) := 0;
BEGIN
    LOOP
        EXIT WHEN i > 10;
        s := s + i;
        i := i + 1;
    END LOOP;
  dbms_output.put_line('Sum of first 10 no's', s);
END;
```

Scanned with CamScanner

(b) WHILE LOOP :-

In the WHILE LOOP statement condition is checked firstly & if the condition become true then all the statements that are written b/w the loop and end loop Keyword will be executed. The while loop will continue as long as the condition is true. When the condition become false the control jump out from the loop.

Syntax for WHILE LOOP is given below :-

```
WHILE <condition>
LOOP
    Statement set ;
    increment / decrement ;
END LOOP;
```

In the above syntax with each iterative, the specified condition is checked. If the condition is true, then the statements are executed & if the condition evaluate to false. then the loop terminates.

eg. -- Print a Table

```
DECLARE
        a number(10);
        i number (10) := 1;
        d number (10);
BEGIN
        a := &a;
        WHILE i <= 10;
        LOOP
            d := a * i;
            dbms_output.put_line(a||'*'||i||'='||d);
            i := i + 1;
        END LOOP;
END;
/
```

(c) FOR LOOP :-

In the WHILE LOOP, the number of iterations are not known in advance but in case of FOR LOOP, Number of iterations are known before the loop is entered. FOR LOOP statement repeat the number of

Scanned with CamScanner

statements over a specifing range. The syntax for FOR LOOP is given below :-

```
FOR (counter variable) IN start .. end
LOOP
    Statement set;
END LOOP;
```

In the above syntax the range is evaluated before the loop entered. The double dots are used to specify the range. Here start represent the starting value of the range and End represent the ending value of the range.

Eg. -- Print a table by using for loop

```
DECLARE
        a number(10);
        d number(10);
BEGIN
        a:=&a;
        FOR i in 1..10
        LOOP
            d:= a*i;
            dbms_output.put_line(a||'*'||i||'='||d);
        END LOOP;
END;
/
```

(3) Jumping Statements :-

Jumping statements are used to change the sequence of the execution of the statements. They are basically used to perform unconditional branching. PL/SQL provide goto statement as a jumping statement :-

(a) GOTO Statement :-

GOTO statement perform the unconditional branching to another executable statement within the same PL/SQL program. In other words GOTO statement transfer the control to the lable. The lable must be unique in the PL/SQL block. Lable is marked by using double angular brackets (<< >>) eg. << lable >>

Scanned with CamScanner

The GOTO statement is used with this lable name to jump from the block to that level. The syntax of GOTO is given below:-

```
Statement set 1;
  -  -  -
  -  -  -
GOTO Lable;
  -  -  -
  -  -  -
<< Lable >>
```

eg:-

-- Prime number

```
DECLARE
        n number(10);
        rem number(10);
        i number(10);
BEGIN
        n:=&n;
        i:= 2;
        WHILE i<n
        LOOP
                rem:= mod(n,i);
        IF (rem=0) THEN
                GOTO abc;
        ENDIF;
                i:= i+1;
        END LOOP;
        <<abc>>
        IF n=i THEN
        dbms_output.put_line('No is Prime');
        ELSE
        dbms_output.put_line('Not Prime');
        ENDIF;
END;
/
```

Scanned with CamScanner