

CHAPTER-12

GETTING STARTED WITH PL/SQL

SQL Vs PL/SQL:

Limitations of SQL are:

- No procedural capabilities .
- Time Consuming Processing or Network traffic.
- No Error Handling Routines/Procedures.

Advantages of PL/SQL are:

- Procedural Capabilities.
- Reduced Network Traffic.
- Error Handling Procedures/Routines.
- Facilitates Sharing.
- Improved Transaction Performance.
- Portable Code.

ANCHORED DECLARATION:

It refers to a declaration where a variable is declared with another variable or a table column used as its anchor.

PL/SQL use %TYPE declaration attribute for anchoring.

Ex: num1 NUMBER(5);
 num2 num1%TYPE;
 empsal Emp.Salary%TYPE;

Note: Anchored types are evaluated at compile time. Thus, you need to recompile the change of underlying type in the anchored variable.

TYPES OF PL/SQL VARIABLES:

- Local Variables.
- Substitution Variables.
- Bind or Host Variables.

PL/SQL BLOCK STRUCTURES:

DECLARE

/* definitions of <constants>
<variables>

BEGIN

<PL/SQL statement here>

[EXCEPTION]

<Exception Handling>

END;

TYPES OF BLOCKS:

- Anonymous Blocks: Blocks without headers.
- Named Blocks: Blocks having headers or labels like procedure, functions, packages or triggers.

PL/SQL CONTROL STRUCTURES:

- Sequence
- Selection
- Iteration.

SELECTION CONSTRUCT: (Condition Testing or Decision Making Statements)

1. Simple IF:-

Syntax:

```
IF <condition> THEN
Statement
END IF;
```

Example:

```
DECLARE
a number;
BEGIN
a := &a;
if a > 100 THEN
dbms_output.put_line(a);
END IF;
```

2. IF...THEN...ELSE...END IF:-

Syntax:

```
IF <condition> THEN
Statement1;
ELSE
Statement2;
END IF;
```

Example:

```
DECLARE
a number;
b number;
BEGIN
a := &a;
b := &b;
if a > b THEN
dbms_output.put_line(a);
ELSE
dbms_output.put_line(b);
END IF;
```

3. NESTED IFELSE:-

```
IF <condition> THEN
Statement1;
ELSIF <condition>
Statement2;
THEN
.
.
.
ELSE
END IF;
```

Example:

DECLARE

```
a number;
b number;
c number;
BEGIN
a :=&a;
b :=&b;
c :=&c;
if a>b THEN
if a>c THEN
dbms_output.put_line(a);
ELSE
dbms_output.put_line(c);
END IF;
ELSE
if (b>c) THEN
dbms_output.put_line(b);
ELSE
dbms_output.put_line(c);
END IF;
END IF;
```

4. ELSIF LADDER:-

Example:

DECLARE

```
salary number;
BEGIN
salary :=&salary;
if salary >=10000 THEN
dbms_output.put_line("CLASS I OFFICER");
ELSIF salary <10000 AND salary>=8000 THEN
dbms_output.put_line("CLASS II OFFICER");
ELSIF salary <8000 AND salary>=5000 THEN
dbms_output.put_line("CLASS III OFFICER");
ELSE
dbms_output.put_line("YOU ARE NOT IN JOB");
END IF;
END IF;
```

Points to remember for using IF:

- Always match up an IF with an END IF.
- You must put a space between the keywords END and IF.
- The ELSIF keyword does not have an embedded “E”.
- Place a semicolon (;) only after the END IF keywords.

ITERATION CONSTRUCT : (LOOPS)

PL/SQL provides three different types of loops:

- The simple loop.
- The FOR loop.
- The WHILE loop.

A General Loop Structure:

A loop has two parts: the loop boundary and the loop body.

The Simple Loop:

Syntax:

```
LOOP
    <executable statement>
END LOOP;
```

Example:

```
DECLARE
n :=0;
LOOP
n:=n+1;
Dbms_output.put_line(n);
END LOOP;
```

NOTE: Simple loop does not terminate by itself. So EXIT and EXIT WHEN statements are used with it to terminate the loop.

```
Ex:  DECLARE
      count number :=0;
      BEGIN
      LOOP
      count :=count +1;
      dbms_output.put_line('value of count is'||count);
      IF count >=10 THEN
      EXIT;
      END IF;
      END LOOP;
      dbms_output.put_line('Hi,I m out of the loop');
      END;
```

```
Ex:  DECLARE
      count number :=0;
      BEGIN
      LOOP
      count :=count +1;
      dbms_output.put_line('value of count is'||count);
      EXIT WHEN count>=10 ;
      END LOOP;
      dbms_output.put_line('Hi,I m out of the loop');
      END;
```

THE NUMERIC FOR LOOP:

The FOR LOOP provided by PL/SQL comes in two forms:

- a) Numeric For loop.
- b) Cursor For loop.

NUMERIC FOR LOOP:

Syntax:

```
FOR <loop index> IN [REVERSE] <lowest number>..<highest number>
LOOP
<executable statements>
END LOOP;
```

Ex:

```
BEGIN
FOR num IN 1..20
LOOP
n := num*2;
dbms_output.put_line(n);
END LOOP;
END;
```

Ex:

```
BEGIN
FOR num IN REVERSE 1..20
LOOP
n := num*2;
dbms_output.put_line(n);
END LOOP;
END;
```

Characteristics of Numeric For Loop:

- a) Loop index is automatically declared.
- b) Expressions in range scheme are evaluated only once.
- c) Loop index is not modifiable.

THE WHILE LOOP:

Syntax:

```
WHILE <condition>
LOOP
<executable statement>
END LOOP;
```

NOTE: WHILE loop tests the condition before executing the loop.

Ex:

```
DECLARE
n number;
BEGIN
WHILE n<=10
LOOP
```

```
n := n+1;
dbms_output.put_line(n);
END LOOP;
END;
```

Variations of WHILE Loop:

```
WHILE TRUE
LOOP
<executable statement>
END LOOP;
```

The Nested Loops:

The nesting of loops or nested loops mean that a loop resides within another loop. A loop can nest any type of loop.

Ex:

```
DECLARE
i number :=0;
BEGIN
WHILE i<10
LOOP
i :=i+1;
dbms_output.put_line(i);
END LOOP;
END;
```

LABELLING LOOPS:

Loops can be labeled to enhance readability.

Syntax:

```
<<outer loop>>
LOOP
```

```
.
.
```

```
EXIT WHEN condition;
END LOOP outer loop;
```

DATABASE INTERACTION IN PL/SQL:

We can use following SQL statements in PL/SQL code.

SELECT,INSERT,UPDATE,DELETE.

SELECT INTO statement:

This statement is used to store the resultant data of SELECT query into PL/SQL variables.

Syntax:

```
SELECT <select list> INTO <variable_list>
FROM <table>[WHERE <condition>];
```

The above syntax is used when we want to store some particular fields or columns of SQL into PL/SQL variables.

But what if we wish to store entire row of data into PL/SQL variable, in that situation the concept of records is used.

USING RECORDS:

A PL/SQL record is a group of multiple pieces of information, related to one another, called fields.

Types of Records:

- a. Table based records.
- b. Programmer based records.
- c. Cursor based records.

Table based records:

It represents each field in the table. For this anchored declaration %ROWTYPE is used.

Syntax:

<record name> <table name>%ROWTYPE;

Programmer Defined Records:

It is an explicitly defined record in PL/SQL. It is defined with TYPE statement as per the following syntax.

Syntax:

TYPE <typename> IS RECORD (field_declaration[, field_declaration]...);

Here, RECORD TYPE declared is treated as a data type, which can not hold values. For which we need to declare a variable of that type.

Syntax:

Variablename RECORD type;

This variable can now be used to access individual columns or fields.

EXCEPTION HANDLING IN PL/SQL:

EXCEPTIONS are some unwanted or undesired situations, which terminate the PL/SQL script unexpectedly.

Types Of EXCEPTIONS:

1. Predefined Exceptions.
2. Undefined Exceptions.
3. User-defined Exceptions.

Predefined Exceptions are not needed to be declared and raised while Userdefined Exceptions are to be declared, raised and handled in EXCEPTION handling section.

* * *