**PL/SQL Features and Programming Basics**

**PL/SQL features ---**

* PL/SQL is an extension of SQL
* It is an application development language containing procedural statements and commands along with SQL commands
* It bridges the gap between database technology and procedural programming languages
* It allows you to process data using flow control statements like iterative loops and conditional branching
* Uses procedural techniques of control, looping and branching
* Supports SQL i.e. cursor operations, exceptions, functions and transactional commands
* Variables and constants, robust error handling and functions
* Adds functionality to non-procedural tools such as SQL\*Forms
* Developers using SQL\*Forms can enter an entire PL/SQL block using a single trigger

**Structure of PL/SQL**

* **Standard PL/SQL code segment is called a Block**
* **A block consists of three parts or sections**
  + **Declaration Part**
  + **Executable Part**
  + **Exception Handling Part**

****

* Declaration Part
  + optional part where variables are defined
* Executable Part
  + mandatory part which consists of executable statements
* Exception Handling Part
  + optional part which consists of code for handling errors (runtime)

**Pl/SQL Files -🡪**

* PL/SQL programs can be written in any editor and saved as files with .sql extension
* Can also use “ED” command in SQL\*Plus to create a PL/SQL program file
* Use the “@ <filename>” command to execute a PL/SQL program file
  + **Variables --**
* Used to store results of a query for later processing, or to calculate values to be inserted into database tables
* Can be used anywhere in an expression, either in SQL or PL/SQL statements
* Must be declared before referencing it in other statements, including other declarative statements
* Are declared by specifying the name along with the datatype
* Can be declared to be of any datatype native to Oracle
* Examples

oldfare NUMBER(5);

m\_name VARCHAR(15);

***(Note – Set Serveroutput On has to be given when a session starts for displaying the output statements\_)***

dbms\_output.put\_line(*String as a parameter*)

**Till 9i we could have 255 characters per line.**

**Now from Oracle 10g the limit is drastically increased to 32767**

declare

x number;

begin

x := 67;

dbms\_output.put\_line(x);

dbms\_output.put\_line('The value of x is '|| x);

end;

**Declaring variable in declare block.**

**Assigning value in in begin block using := .**

**Output statement is dbms\_output.put\_line**

**Concatenation operator is ||**

**Command terminator is ; after end**

**Declaring and initializing variables together**

declare

y number := 100;

begin

dbms\_output.put\_line('The value of y is '|| y);

end;

**-------------------------------------------------------------------------------**

**Taking value from the user using &**

declare

z number;

a varchar2(10);

begin

z := &z;

a := '&a';

dbms\_output.put\_line('Z is '|| z);

dbms\_output.put\_line('A is '|| a);

end;

**-------------------------------------------------------------------------------**

**/\*Cannot declare or initialize more than one variable simultaneously\*/**

declare

a number;

b number;

c number;

begin

a := 67; b := 90; c := 87;

dbms\_output.put\_line(a);

dbms\_output.put\_line(b);

end;

**A constant number has to declared and initialized in the declare block only using**

**CONSTANT keyword. Value cannot be changed**

declare

r CONSTANT number :=100;

begin

/\* r := r + 100; Not possible\*/

dbms\_output.put\_line(r);

end;

**/\*Assigning value to variable from a column of a table using select into clause\*/**

declare

x number;

begin

Select sal Into x from emp

where ename = 'SMITH';

dbms\_output.put\_line('Salary of Smith is '|| x);

end;

**/\* Selecting ename,sal from emp**

**Use of more than one columns value with Into clause\*/**

declare

n varchar2(50);

s number;

begin

select **ename**, **sal** Into **n**, **s**

from emp

where ename = 'SMITH';

dbms\_output.put\_line(n);

dbms\_output.put\_line(s);

end;

**% Type Attribute –**

* **Provides datatype of a variable or column**
* **Useful when declaring a variable that refers to a column in a database**
  + **exact datatype of column need not be known**
  + **if column definition changes, variable datatype changes accordingly at runtime**
* **Example**

**oldfare fare.first\_fare%TYPE;**

**newfare oldfare%TYPE;**

declare

**a emp.ename%type;**

**b emp.sal%type;**

**c emp.deptno%type;**

/\*Using %TYPE attribute for variable data type\*/

begin

select ename,sal,deptno

into a,b,c

from emp

where ename = 'KING';

dbms\_output.put\_line(a ||'-'|| b ||'-' || c);

end;

**%RowType Attribute –**

* **Useful when declaring a record variable having same structure as a row in a table or view, or as a row fetched from a cursor**
* **Fields in the record have same names and datatypes as the columns in the table/view**
* **Example**

**emp\_rec employee%ROWTYPE;**

* **A specific field can be referenced using**

**emp\_rec.emp\_num;**

declare

E emp%rowtype;

/\*rowtype attribute holds the datatype of the columns of the

entire row\*/

begin

select \* INTO E

from emp

where ename = 'MARTIN';

dbms\_output.put\_line(E.sal);

dbms\_output.put\_line(E.ename);

dbms\_output.put\_line(e.deptno);

end;

**Conditional Statements**

**1) Case Construct**

Whenever Case is used in a PLSQL block then Case …. End Case has to be used.

Declare

X number;

Begin

X := &x;

**Case X**

When 1 then Dbms\_output.put\_line('A');

Else

Dbms\_output.put\_line('B');

**End Case;**

End;

**2) IF**

* The selection structure tests a condition, then executes one sequence of statements instead of another, depending on the condition
* There are three forms of statements
  + IF-THEN
  + IF-THEN-ELSE
  + IF-THEN-ELSIF
* Sequence of statements is executed only if the condition evaluates to TRUE
* If condition evaluates to FALSE or NULL, it does nothing
* In either case control passes to next statement after the IF-THEN structure

IF <condition> THEN

statements;

END IF;

* Sequence of statements in the ELSE clause is executed only if the condition evaluates to FALSE or NULL

IF <condition> THEN

statements;

ELSE

statements;

END IF;

--------------------------------------------------------------------------------

declare

/\*Simple if condition \*/

x number;

begin

x := &x;

if x >= 35 then

dbms\_output.put\_line('Passed');

else

dbms\_output.put\_line('Failed');

end if;

end;

-----------------------------------------------------

**IF-THEN-ELSIF Structure**

* This construct allows selection of action from several mutually exclusive alternatives
* The IF statement can have any number of ELSIF clauses
* The final ELSE is optional
* Conditions are evaluated one by one from top to bottom

**Syntax**

IF <condition1> THEN

statements;

ELSIF <condition2> THEN

statements;

ELSIF <condition3> THEN

statements;

ELSE

statements;

END IF;

**Example 1 –**

Declare

y number;

/\*Multiple ifs \*/

Begin

y := &y;

if y >= 70 then

dbms\_output.put\_line('Distinction');

elsif y >= 60 then

dbms\_output.put\_line('First class');

elsif y >= 50 then

dbms\_output.put\_line('Second class');

elsif y >= 35 then

dbms\_output.put\_line('Passed');

else

dbms\_output.put\_line('Failed');

end if;

end;

**Example 2**

create table adm

(Name varchar2(30),

Marks number(3),

College varchar2(30),

Fees number(5));

**/\*Use of multiple if's**

**Accept name and marks from user.**

**Depending upon marks entered the college and fees should be decided**

**and the record should be entered in the adm table.\*/**

Declare

n adm.name%type;

m adm.marks%type;

c adm.college%type;

f adm.fees%type;

Begin

n := '&n';

m := &m;

if m >= 95 then

c := 'COEP';

f := 10000;

elsif m >= 90 then

c := 'MIT';

f := 15000;

elsif m >= 85 then

c := 'VIT';

f := 22000;

elsif m >= 80 then

c := 'D Y Patil';

f := 27000;

elsif m >= 75 then

c := 'Pune Vidyarthi';

f := 33000;

else

dbms\_output.put\_line('Cannot get admission');

end if;

if c is not null and f is not null then

dbms\_output.put\_line('Your College is '|| c || ' and fees are ' || f);

**Insert into adm**

**values(n,m,c,f);**

commit;

end if;

end;

**Nested IF –**

declare

x number;

y number;

z number;

begin

x := 1;

y := 6;

z := 7;

**if x >= 1 then**

**if** y >= 100 **then**

dbms\_output.put\_line('A');

**else**

dbms\_output.put\_line('B');

**end if;**

**else**

**if** z >= 50 **then**

dbms\_output.put\_line('C');

**else**

dbms\_output.put\_line('D');

**end if;**

**end if;**

end;

/

Nested Blocks

**declare**

**x number;**

**begin**

**x := 80;**

**dbms\_output.put\_line('abc');**

**declare**

**y number;**

**begin**

**y := 90;**

**dbms\_output.put\_line('Inner Block variable value ' || y);**

**end;**

**dbms\_output.put\_line('Outer Block variable value ' || x);**

**end;**

# Scope of variables

**A variable declared in the outer block is accessible in the inner block. But a variable declared in the inner block is accessible only in the inner block.**

**declare**

**outer number;**

**begin**

**outer := 80;**

**declare**

**inner number;**

**begin**

**inner := 90;**

**dbms\_output.put\_line('Inner Block variable value ' || inner);**

**dbms\_output.put\_line('Outer block variable is accessible in the inner**

**block’);**

**dbms\_output.put\_line('Outer block variable value ' || outer);**

**end;**

**dbms\_output.put\_line('Outer Block variable value ' || outer);**

**dbms\_output.put\_line('Inner Block variable value ' || inner);**

**end;**

**/**

# Labels

If the variables names of the outer and inner blocks are same then labels have to be used within the inner block to avoid ambiguity.

|  |
| --- |
| **<<outer\_block>>**  **declare**  **x number;**  **begin**  **declare**  **x number := 100;**  **begin**  **dbms\_output.put\_line('Value of the inner block x is ' || x);**  **-- Giving value of x of the inner block to the outer block x**  **outer\_block.x := x;**  **end;**  **x := x + 500;**  **dbms\_output.put\_line('Value of the outer block x is ' || x);**  **end;**  / |

**LOOPS**

* The ability to repeat or skip sections of a block can be achieved with the usage of LOOP or GOTO statements
* There are three forms of the LOOP statement
  + LOOP
  + WHILE-LOOP
  + FOR-LOOP

**LOOP Statement**

* LOOP repeats a sequence of statements
* Statements to be repeated are placed between keyword LOOP and END LOOP
* With each iteration of the loop, the sequence of statements is executed, then control resumes at the top of the loop

LOOP

statements;

END LOOP;

**EXIT Statement**

* Used to complete a loop if further processing in a loop is undesirable or impossible
* There are two forms of the EXIT statement
  + EXIT
  + EXIT-WHEN
* Forces a loop to complete unconditionally
* Must be placed inside a loop

LOOP

statements;

IF <condition> THEN

EXIT; -- exit loop immediately

END IF;

END LOOP;

-- control resumes here

Example of Loop –

**/\*To show 1 to 10 on screen\*/**

Declare

x number;

Begin

x := 1;

**Loop**

**dbms\_output.put\_line(x);**

**x := x + 1;**

**exit when x > 10;**

**End Loop;**

dbms\_output.put\_line('end');

End;

**Database Oriented Example of Basic Loop:**

create table five

(no number);

**/\*Inserting multiples of five in table five\*/**

Declare

x number;

Begin

x := 5;

Loop

**Insert into five**

**values(x);**

x := x + 5;

exit when x > 50;

End Loop;

End;

/

WHILE-LOOP Statement

* Associates a condition with a sequence of statements enclosed within LOOP-END LOOP
* Condition evaluated before each iteration
* If condition evaluates to TRUE, sequence of statements is executed and control resumes at the top of the loop
* If condition evaluates to FALSE or NULL, loop is bypassed and control passes to next statement
* Number of iterations depends on the condition and is unknown until the loop completes

WHILE <condition>

LOOP

statements;

END LOOP;

**Example 1 of while loop to show 1 to 15**

declare

x number;

Begin

x := 1;

**while x <=15**

**Loop**

**dbms\_output.put\_line(x);**

**x := x + 1;**

**End Loop;**

end;

Example 2 Forces a loop to complete unconditionally

declare

z number;

/\*Using break after z reaches to 8\*/

Begin

z := 1;

while z <=15

Loop

dbms\_output.put\_line(z);

z := z + 1;

exit when z = 8;

End Loop;

end;

**Database Oriented Example of While Loop:**

drop table while\_loop;

create table while\_loop

(sr\_no int,

sal int);

Insert Into while\_loop

Values(1, 300);

Insert Into while\_loop

Values(2, 400);

Insert Into while\_loop

Values(3, 500);

commit;

declare

current\_sal int;

sum\_sal int := 0;

counter int :=1;

/\* To display the total of first two salaries\*/

begin

while counter <= 2

loop

select sal into current\_sal

from while\_loop

where sr\_no = counter;

counter := counter + 1;

sum\_sal := sum\_sal + current\_sal;

end loop;

dbms\_output.put\_line(sum\_sal);

end;

/

**While Loop v/s Basic Loop**

### While Loop

declare

x number;

Begin

x := 1;

while x > 15

Loop

dbms\_output.put\_line(x);

x := x + 1;

End Loop;

dbms\_output.put\_line('End of program');

end;

/

The loop will never get executed since the condition is wrong from the start of the iteration.

### Basic Loop

Declare

x number;

Begin

x := 1;

Loop

dbms\_output.put\_line(x);

**exit when x = 1;**

x := x + 1;

End Loop;

dbms\_output.put\_line('End of program');

End;

/

The loop gets executed at least once.

**FOR LOOP**

Advantages --

1) No need of declaring loop variable

2) No need of giving condition

3) No need of updation statement (increment or decrement )

4)Code becomes small and compact

Disadvantage --

Updation can be done by only one.

Syntax –

FOR <counter> IN [REVERSE]

lower\_bound .. higher\_bound LOOP

statements;

END LOOP

**Example 1 of for loop**

**/\*To show 1 to 10 on screen\*/**

begin

for x in 1..10

Loop

dbms\_output.put\_line(x);

End Loop;

end;

**Example 2**

**/\*Reverse for loop 10,9,8 … 1\*/**

Begin

for i in REVERSE 1 ..10

Loop

dbms\_output.put\_line(i);

End Loop;

end;

**Example 3 –**

**Calculating compound interest for a principal of Rs.100 @10% for each year.**

**Values will be shown of the CI after each year.**

create table CI\_100

(year number(2),

total number(4));

----------------------------------

Declare

p number := 100;

tot number;

/\*Calculation of compound interest.

Rs.100 is principal.

Rate of interest is 10%.

Period is 5 years.

\*/

Begin

for y in 1..5

Loop

/\* Tot variable is getting 10% more than p \*/

tot := p + p \* 0.10;

Insert into CI\_100

values(y,tot);

/\*Since the next interest is based on the current interest

so the tot will be considered as p for the next year\*/

p := tot;

End Loop;

end;

**Nested Loops**

create table discount\_details

(quarter number(2),

month number(2),

discount varchar2(5));

## Expected Output –

# QUARTER MONTH DISCOUNT

1 1 12%

1 2 11%

1 3 10%

2 4 9%

2 5 8%

2 6 7%

3 7 6%

3 8 5%

3 9 4%

4 10 3%

4 11 2%

4 12 1%

12 rows selected.

**declare**

**q number;**

**m number;**

**d number;**

**dis varchar2(10);**

**c number;**

**begin**

**q := 1;**

**m := 0;**

**d := 12;**

**loop**

**exit when q > 4;**

**c := 0;**

**loop**

**exit when c >= 3;**

**m := m + 1;**

**dis := d || '%';**

**insert into discount\_details**

**values(q,m,dis);**

**d := d - 1;**

**c := c + 1;**

**end loop;**

**q := q + 1;**

**end loop;**

**end;**

**GOTO Statement**

* Branches to a label unconditionally
* When executed, it changes the flow of control in a PL/SQL block
* Two parts needed to code a GOTO statement
  + Define a label name
  + Use the GOTO statement to pass control to the label
* Label name
  + optionally used to name a PL/SQL block or statements within the block
  + Defined using angular brackets (<< >>)

<<if\_fare\_label>>

IF condition THEN

statements;

END IF;

statements;

GOTO if\_fare\_label;

* Transfer of control using GOTO statement is allowed in the following places
  + from a block to an executable statement
  + branch from an exception handler into an enclosing block
* Transfer of control using GOTO statement is NOT allowed in the following places
  + from within one IF statement or loop clause to another
  + from an enclosing block into a sub-block
  + from an exception handler into the current block
  + out of a subprogram
  + to keywords

**Examples of GOTO –**

1)

create table prec

(name varchar2(20),

dept varchar2(20));

create table mahrec

(name varchar2(20),

dept varchar2(20),

city varchar2(30));

**/\*Pune records should go in both the tables prec and mahrec.**

**Any other city record should go in mahrec table only.\*/**

declare

n prec.name%type;

d prec.dept%type;

c mahrec.city%type;

begin

n := '&n';

d := '&d';

c := '&c';

if c = 'Pune' then

goto s1;

else

goto s2;

end if;

<<s1>>

Insert into prec

values(n,d);

<<s2>>

Insert into mahrec

values(n,d,c);

end;

/

#### Legal and Illegal Usage of GOTO

Ex 1

declare

x number;

begin

x := 90;

<<s1>>

dbms\_output.put\_line('S1');

if x = 90 then

dbms\_output.put\_line('Condition is true');

**goto s1;**

else

dbms\_output.put\_line('Condition is false');

end if;

end;

/

Ex 2

declare

x number := 90;

begin

if x = 90 then

goto abc;

else

<<abc>>

dbms\_output.put\_line('Failed');

end if;

end;

/

Ex 3

declare

<<abc>>

x number := 9;

begin

goto abc;

end;

/

Ex 4 Cannot jump from the parent block into a child block.

Begin

dbms\_output.put\_line('a');

goto x1;

begin

<<x1>>

dbms\_output.put\_line('x1');

end;

dbms\_output.put\_line('b');

end;

Ex 5 Can jump from the child block into the parent block provided the label is after the child block.

Begin

dbms\_output.put\_line('a');

begin

goto x1;

end;

dbms\_output.put\_line('b');

<<x1>>

dbms\_output.put\_line('x1');

end;

/

Ex 6 Cannot jump from the child block into the parent block if the label is above the child block.

Begin

dbms\_output.put\_line('a');

<<x1>>

dbms\_output.put\_line('x1');

begin

goto x1;

end;

dbms\_output.put\_line('b');

end;

/

Ex 7 Can jump outside the loop.

declare

x number := 1;

begin

loop

dbms\_output.put\_line(x);

x := x + 1;

exit when x >= 10;

if x = 5 then

goto label1;

end if;

end loop;

dbms\_output.put\_line('1');

dbms\_output.put\_line('2');

dbms\_output.put\_line('3');

<<label1>>

dbms\_output.put\_line('Came out of loop!');

end;

/

Ex 7 If the label is defined inside the loop then from outside the loop cannot jump inside the loop.

declare

x number := 1;

begin

goto label1;

loop

dbms\_output.put\_line(x);

x := x + 1;

exit when x >= 10;

<<label1>>

dbms\_output.put\_line('Came inside the loop!');

end loop;

end;

Ex 8 Cannot jump from parent loop into the child loop;

begin

for a in 1 .. 5

loop

dbms\_output.put\_line(a);

if a = 3 then

goto label1;

end if;

for b in 1 .. 4

Loop

dbms\_output.put\_line(b);

<<label1>>

dbms\_output.put\_line('XYZ');

End Loop;

end loop;

end;

/