PL/SQL Introduction

Introduction

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

- Following are notable facts about PL/SQL:

- PL/SQL is a completely portable, high-performance transaction-processing language.
- PL/SQL provides a built-in interpreted and OS independent programming environment.
- PL/SQL can also directly be called from the command-line SQL*Plus interface.
- PL/SQL's general syntax is based on that of ADA and Pascal programming language.

Features of PL/SQL

- PL/SQL is tightly integrated with SQL.
- It offers extensive error checking.
- It offers numerous data types.
- It offers a variety of programming structures.
- It supports structured programming through functions and procedures.
- It supports object oriented programming.
- It supports developing web applications and server pages.

Features of PL/SQL

- SQL is the standard database language and PL/SQL is strongly integrated with SQL.
  - PL/SQL supports both static and dynamic SQL.
    - Static SQL supports DML operations and transaction control from PL/SQL block.
    - Dynamic SQL is SQL allows embedding DDL statements in PL/SQL blocks.

Features of PL/SQL

- PL/SQL allows sending an entire block of statements to the database at one time.
  - This reduces network traffic and provides high performance for the applications.

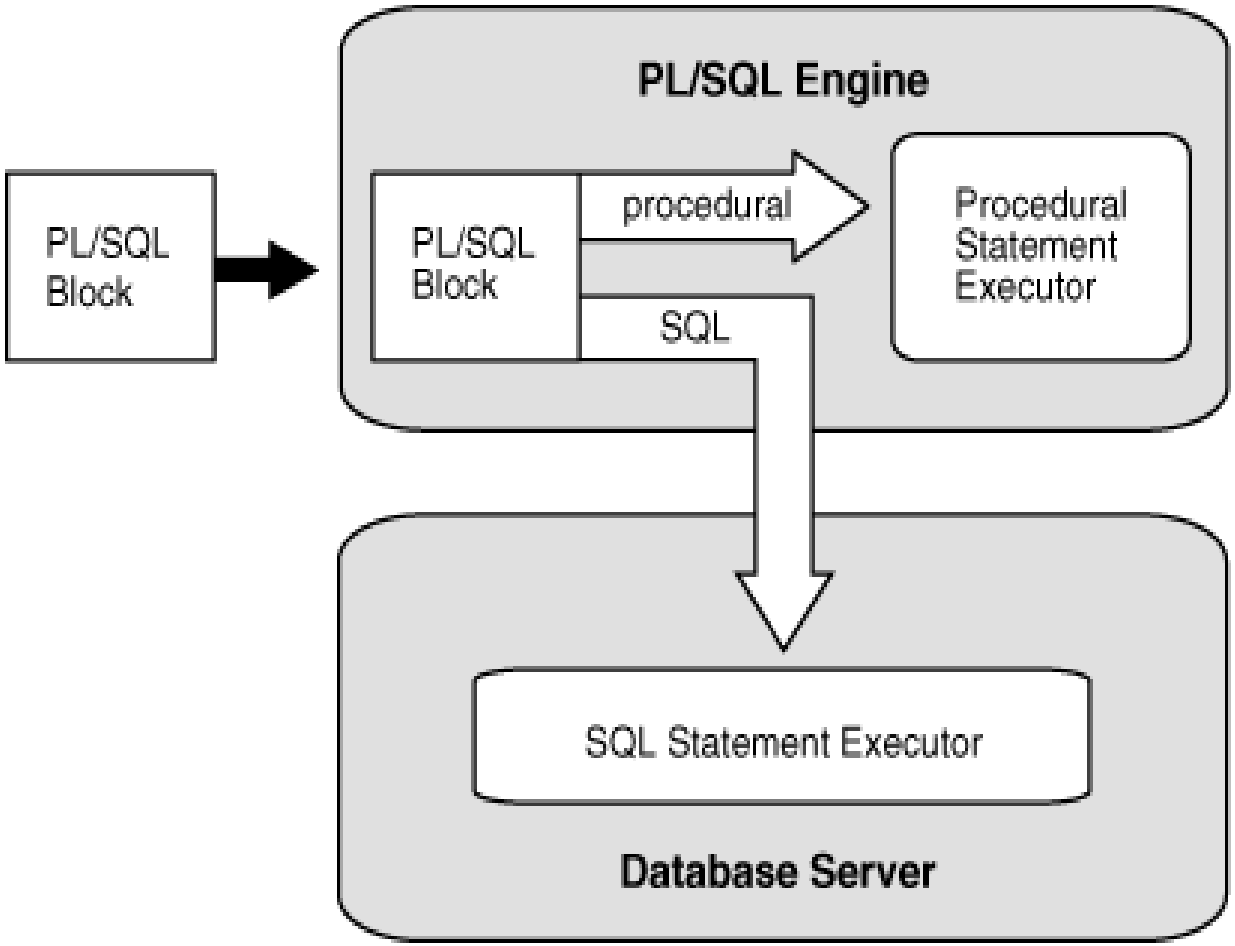
Features of PL/SQL

- PL/SQL give 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.

Features of PL/SQL

- Applications written in PL/SQL are fully portable.
- PL/SQL provides access to predefined SQL packages.
- PL/SQL provides support for Object-Oriented Programming.
- PL/SQL provides support for Developing Web Applications and Server Pages.

PL/SQL execution



PL/SQL BLOCK STRUCTURE

DECLARE (optional)

- variable declarations

BEGIN (required)

- SQL statements
- PL/SQL statements or sub-blocks

EXCEPTION (optional)

- actions to perform when errors occur

END; (required)

PL/SQL Block Types

Anonymous

```
DECLARE
BEGIN
    -statements
EXCEPTION
END;
```

Procedure

```
PROCEDURE <name>
IS
BEGIN
    -statements
EXCEPTION
END;
```

Function

```
FUNCTION <name>
RETURN <datatype>
IS
BEGIN
    -statements
EXCEPTION
END;
```

PL/SQL Variable Types

- Variable-name datatype(size);

```
DECLARE
   a number := 10;
   b number := 20;
   c number;
```

Declaring a Constant

```
PI CONSTANT NUMBER := 3.141592654;
DECLARE
   -- constant declaration
```

Assignment

- 1. :=
  - A:=10;
  - Sum:=A+B+C;

  2. Get value from data base object.

  SELECT INTO

  Select salary into SAL from employee where empid=12;

PL/SQL is strongly typed

- All variables must be declared before their use.
- The assignment statement

  : =

  is not the same as the equality operator

  =

- All statements end with a ;

The PL/SQL Literals

Literal Type	**Example:**
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';

PL/SQL Operators

Operator	Description	Example
+	Adds two operands	A + B will give 15
-	Subtracts second operand from the first	A - B will give 5
*	Multiply both operands	A * B will give 50
/	Divide numerator by de-numerator	A / B will give 2
**	Exponentiation operator, raises one operand to the power of other	A ** B will give 100000

Relational

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

Comparison Operators

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 and FALSE if it does not.	If 'Zara Ali' like 'Z% A_i' returns a Boolean true, whereas, 'Nuha Ali' like 'Z% A_i' returns a Boolean false.
BETWEEN	The BETWEEN operator tests whether a value lies in a specified range. x BETWEEN a AND b means that x >= a and x <= b.	If x = 10 then, x between 5 and 20 returns true, x between 5 and 10 returns true, but x 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') returns boolean false but x in ('m', 'n', 'o') returns Boolean 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 always yield NULL.	If x = 'm', then 'x is null' returns Boolean false.

Logical Operators

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

PL/SQl Comments

- A:=5; -- assign value 5 to variable A.
- A:=b+c; /* the value of variable A and B are added and assign to variable A */

Important PL/SQL delimiters

+, **-**, *****, **/** arithmetic operators
; statement terminator
:= assignment operator
=> association operator
|| strings concatenation operator
. component indicator
% attribute operator
‘ character string delimiter
-- single line comment
/*, ***/** multi line comment delimiters
.. range operator
=, **>**, **>=**, **<**, **<=** relational operators
!=, **~=**, **^=**, **<>** not equal relational operators
is null, like, between PL/SQL relational operators

Example Prog

```
DECLARE
message  varchar2(30):= ‘Hello World’;
BEGIN
  dbms_output.put_line(message);
END;
```

Out Put

- SQL> Set Serveroutput ON;

- dbms_output.put_line(A);
- dbms_output.put_line('Value of A is:' || A);

Read a value during runtime

- Num:= #

  – This will produce a message on screen

    - Enter the value of NUM:

      User can enter any value at run time to NUM.

```
Declare
        a number(2);
         b number (2);
         c number(2);
Begin
        a:=&a;
        b:=&b;
         c:=a+b;
    dbms_output.put_line('sum=' || c);
End;
```

Control statements

- Conditional / selection
- Iterative
- Sequence

Conditional / selection

- IF condition then
  Sequence of statements;
  End if;

- IF condition then
  Sequence of statements;
  Else
  Sequence of statements;
  End if;
- IF condition1 then
  Sequence of statements;
  Else if condition2 then
  Sequence of statements;
  Else
  Sequence of statements;
  End if;

```
Declare
Num1 number(2);
Num2 number(2);
Begin
Num1:=&num1;
Num2:=&num2;
If num1>num2 then
dbms_output.put_line('greater number is:=' || num1);
Else
dbms_output.put_line('greater number is:=' || num2);
End if;
End;
```

- Iterative
  - Loop
  - While – loop
  - For-loop

```
Loop
Sequence of statements;
Exit when condition;
End loop;
```

```
Set serveroutput on;
Declare
i number(2);
Begin
i:=1;
Loop
Dbms_output.put_line(i);
i:=i+1;
Exit when i>10;
End loop;
End;
```

```
Set serveroutput on;
Declare
A number(2);
Begin
A:=1;
While a<=10
Loop
Dbms_output.put_line(a*a);
A:=a+1;
End loop;
End;
```

For loop

- For counter in [reverse] lower bound..higher bound
- Loop
- Sequence of statements;
- End loop;

```
Declare
Total number(4);
Begin
For i in 1..10
Loop
Total:=2*i;
Dbms_output.put_line('2*'||i||'='||total);
End loop;
End;
```

goto

- Declare
- Num1 number(2);
- Num2 number(2);
- Begin
- Num1:=&num1;
- Num2:=&num2;
- If num1> num2 then
- Goto p1;
- Else
- Goto p2;
- End if;
- <<p1>>
- Dbms_output.put_line('num1 is bigger');
- Goto p3;
- <<p2>>
- Dbms_output.put_line('num2 is bigger');
- <<p3>>
- Dbms_output.put_line('End of Program ');

- End;

Subprograms

- Functions
- Procedures
- Stored procedures
  - Modularity
  - Reusability

Declare
Global variables declaration;
Procedure procedure name
(Arguments IN/OUT/IN OUT data types)
IS/AS
Variable and constant declaration;
Begin
PL/SQL statements;
Exception
Statements;
End procedure name;
Begin
Executable statements;
Procedure calling;
End;

Function

- Create or replace function f101(num1 number, num2 number) return number IS
- c number(10);
- Begin
- c:=num1+num2;
- return c;
- End;

Execute Func

- set serveroutput on;
- begin
- dbms_output.put_line(f101(10,20));
- end;

Stored Procedure

- Create or replace procedure addition6 IS
- num1 number(10) :=3;
- num2 number(10) :=7;
- c number(10);

- Begin
- c:=num1+num2;
- dbms_output.put_line(c);
- End;

Execute Stored Proc

- set serveroutput on;
- exec addition6 ;

emp table

Emp_name	Emp_id	TA	DA	Total	Branch_City
abc	10	1200	1345	2545	Delhi
xyz	12	1100	1200	2300	Mumbai

```
Declare
    a number(5);
    b number(5);
   t number(5);
Begin

  select ta,da,into a,b from emp where empid=12;
            t:=a+b;
update emp set total =t where empid=12;
end;
```

Emp_name	Branch	City	Contact_number
A kumar	Delhi main	Delhi	9872177002

%TYPE

- Provide the data type of a variable or column.

- Exp:
  - sal employee.salary%TYPE;

%ROWTYPE

- Declare
- dept_rec dept%ROWTYPE; -- declaring record variable.

  detp_rec.deptno;

  dept_rec.deptname; -- accessing coloums

  - %ROWTYPE has all properties of %TYPE and one additional that we required only one variable to access any number of columns.

emp table

Emp_name	Emp_id	TA	DA	Total	Branch_City
abc	10	1200	1345	2545	Delhi
xyz	12	1100	1200	2300	Mumbai

```
Declare
a emp.ta%TYPE;
b emp.da%TYPE;
t emp.total%TYPE;
Begin
Select ta,da into a,b from emp where emp_id=12;
t:=a+d;
Update emp set total =t where empid=12;
End;
```

```
Declare
Record1  emp%ROWTYPE;
Begin
Select * into record1 from emp where empid=12;
Record1.total:=record1.ta+record1.da;
Update emp set total=record1.total where empid=12;
End;
```

Triggers

- Stored procedures that automatically executed when some event occurs to data base.
- Events are
  - Insert
  - Delete
  - Update

Trigger vs procedures

- Triggers do not accept parameters.
- Triggers are executed automatically without user calling.

Parts of trigger

- Triggering event or statement
- Trigger restriction
  - Is boolean value true or false.
- Trigger action

Types of triggers

- Row trigger
  - Fired for each row effected by trigger statement.
- Statement trigger
  - Fired once for triggering statements regardless of number of rows effected.

Another classification of triggers

- BEFORE trigger
  - Trigger executes its trigger action before the triggering statement
- AFTER trigger
  - Trigger executes its trigger action after the triggering statement

```
Create or  replace trigger t11
BEFORE/AFTER
DELETE/INSERT/UPDATE of column name
On table
REFFERENCING OLD AS old, NEW AS new
For each row
When condition
Declare
Variable declarations
Begin
Statements
Exception
Error handling statements
End;
```

write a trigger that will copy a record in second table before deleting the record in the 1st table

```
create or replace trigger trigger5243
BEFORE DELETE on t5243
referencing OLD as OLD
for each row
BEGIN
Insert into t52432 values(:OLD.id,:OLD.name);
end;
```

- Create PL/SQL trigger which will tell about the operation performed on database.

```
Create or replace trigger t11
Before INSERT or UPDATE or DELETE
ON Student
Begin
IF INSERTING then
Dbms_output.put_line(“operation performed inserting”);
ELSEIF UPDATING  then
Dbms_output.put_line(“operation performed Updating”);
ELSE
Dbms_output.put_line(“operation performed Deletion”);
End if;
End;
```

Cursors

- A cursor is a work area where the result of a SQL query is stored at server side.
- The contents of a cursor are then displayed at client machine via a network.
- Known as active data set
  - Declare a cursor
  - Open a cursor
  - Read from a cursor
  - Close cursor

Types of cursors

- Implicit cursor
- Work area that is declared, opened and closed internally by oracle engine.
- Explicit cursor (user defined)
  - Define in DECLARE section of PL/SQL block

Cursor attributes

- %ISOPEN
- %FOUND
- %NOTFOUND
- %ROWCOUNT

Implicit cursors

- SQL%ISOPEN
- SQL%FOUND
- SQL%NOTFOUND
- SQL%ROWCOUNT

- Write a PL/SQL block to display a message that whether a record is updated or not.

- Begin
- Update student set city = ‘delhi’ where rollono=&rollno;
- If SQL%FOUND then
- Dbms_output.put_line(‘record updated’);
- End if;
- If SQL%NOTFOUND then
- Dbms_output.put_line(‘record not updated’);
- End if;
- End;

- set serveroutput on;
- Begin
- Update student2 set class = 'me' where std_no=&std_no;
- If SQL%FOUND then
- Dbms_output.put_line('record updated');
- End if;
- If SQL%NOTFOUND then
- Dbms_output.put_line('record not updated');
- End if;
- End;

- Write a PL/SQL block to count the number of rows affected by an update statement.

- Declare
- Num number(2);
- Begin
- Update student set grade =‘b’ where grade =‘c’;
- Num:=SQL%ROWCOUNT;
- Dbms_output.put_line(‘total rows affected =’ || num);
- End;

Explicit cursors

- %ISOPEN
- %FOUND
- %NOTFOUND
- %ROWCOUNT

Steps of execution

- Declare the cursor
- Open the cursor
- Using loop, fetch the data from cursor one row at a time and store in memory variable
- Exit from the loop
- Close the cursor

- Cursor cursorname IS select statements.

- Cursor C123 IS select rollno,name from student where branch=‘CSE’;

- Open cursorname;
- Open C123;

- FETCH cursorname INTO variable;
- FETCH C123 INTO my_rollno,my_name;

- Loop
- FETCH C123 into MY_record;
- Exit when C123%notfound;
- Other statements;
- End loop;

- CLOSE cursorname;
- CLOSE C123;

- Write a PL/SQL cursor to display the name of the students belonging to CSE branch...

- Declare
- Cursor C123 is select name from student where branch ='CSE';
- my_name student.name%type;
- Begin
- Open C123;
- Loop
- Fetch C123 into my_name;
- Exit when C123%NotFound;
- dbms_output.put_line(my_name);
- End loop;
- Close C123;
- End;

Exception

- set serveroutput on;
- declare
- n emp.name%type;
- begin
- select name into n from emp where id=502;
- dbms_output.put_line('empname :='||n);
- exception
- when **too_many_rows** then
- dbms_output.put_line('more than one row returned');
- end;

User Defined Exceptions

```
set serveroutput on;
declare
a number:=3;
b number:=0;
c number;
e exception;
begin
if b=0 then
raise e;
end if;
c:=a/b;
dbms_output.put_line('result='||c);
exception when e then
dbms_output.put_line('error!- your divisor is zero');
end;
```