



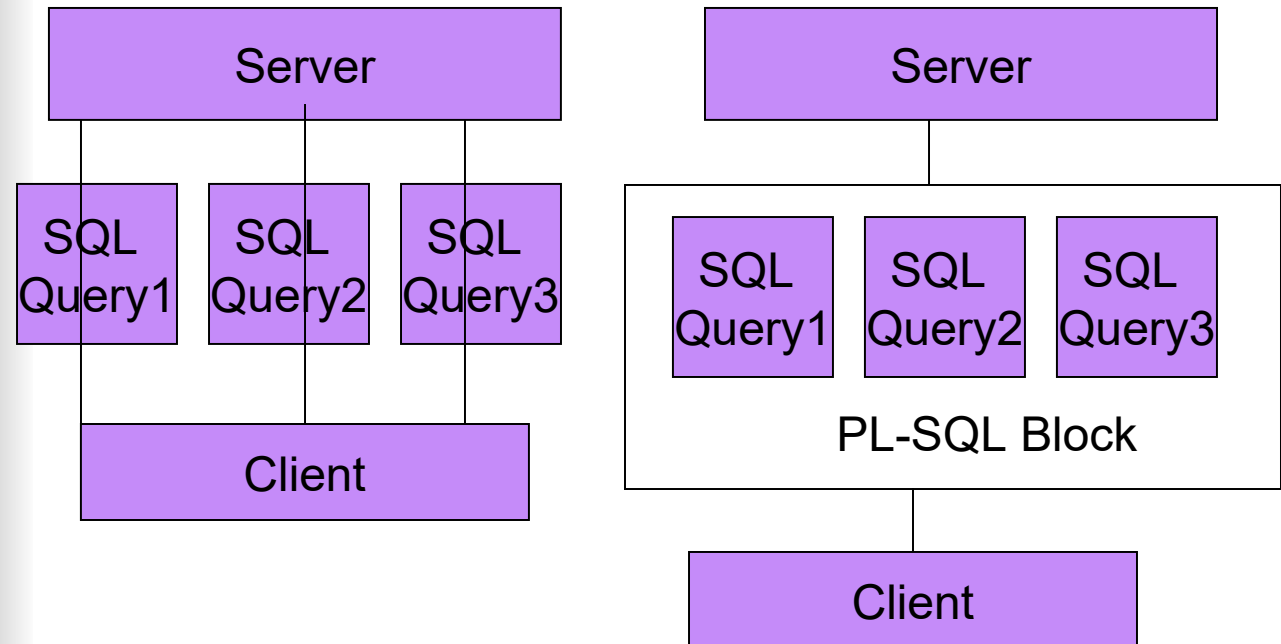
Objectives

- Why PL-SQL ?
- Language features
- Basic Structure of PL/SQL program
- Data Types
- Control Flow in PL-SQL
- Loops in PL-SQL

Why PL SQL ?

- PL/SQL stands for Procedural Language/SQL.
- PL/SQL extends SQL by adding constructs found in procedural languages like procedures, loops, variables, objects etc.
- Resulting a structural language that is more powerful than SQL

PL SQL, Is there any Advantage ?



- In case of SQL to send 3 queries we will need three network trips between client and server.
- In PL-SQL we bundle any number of queries in a block and in single network trip task is done.



Language features

- Supports constructs like any other 4th generation language:
 - Variables and Data types
 - Loops and Control statements
 - Procedures and Functions
 - Packages
 - Triggers
 - Objects
 - Records (Its like structure in C language)



PL SQL program structure

Declare

<All Variables, cursors, exception etc are declared here>

Begin

<All programming logic , queries , program statements are written here>

Exception

<All Error Handling code is written here>

End;

--It ends the program



PL SQL nested block

<<Outer Block>>

Declare

Begin

<<Inner Block>>

Declare

Begin

Exception

End;

Exception

End;

PL SQL Block



Remember :

Declare is optional and only required when variables need to be declared.

Exception is optional and required when Error/Exception handling is done.

Begin and End are mandatory as all logic and queries are written inside it.

Declare

Begin

Exception

End;



PL SQL program- Sample I



BEGIN

Insert into Dept values(70,'HR','Pune');

Insert into Dept values(80,'PSD','Mumbai');

Insert into Dept values(90,'ESG','Pune');



END;

--This program will insert three records at the same time in the table dept.





PL SQL program- Sample II

-- This program displays the sum of two numbers

DECLARE

v_num1 Number;

v_num2 Number;

v_sum Number;

BEGIN

V_num1 := &Number1;

V_num2 := &Number2;

V_sum := v_num1 + v_num2 ;

Dbms_Output.Put_Line ('The Sum of number is : ' || v_sum);

END;



Save , Edit and Execute program

- Type your program in SQL * plus
- To save : **Save <File Name>**
Program is saved in the bin directory to save in other folder give complete path.
Eg: **Save 'C:\ESG\FirstPrg.sql'**
- To make changes:
Edit <File Name>
To edit program saved in folder other then bin
Edit 'C:\ESG\FirstPrg.Sql'
- To Execute:
@ File Name
To execute program saved in folder other then bin.
@ 'C:\ESG\FirstPrg.Sql'



Important Keywords

- Following are the keywords in PL-SQL , should not be used as a variable name.
 - DECLARE
 - BEGIN
 - END
 - EXCEPTION
 - LOOP , END LOOP
 - IF , ELSE , ELSIF , END IF
 - CURSOR
 - PROCEDURE
 - FUNCTION

Cont..



Important Keywords

- Keywords
 - PACKAGE
 - TRIGGER
 - GRANT
 - REVOKE
 - FOR
 - WHILE
 - CASE
 - VARRAY
 - TYPE
 - OBJECT



Operators

■ Important operators in PL SQL

- Airthmetic : (+ , - , * , /)
- Logical: (AND , OR , NOT)
- Comparison: (<=, <, >, =)
- Comments (Two hyphens): --
- Assignment operator: In PL SQL assignment operator is

:=

So to assign values we need to write :=

Examples:

z := x + y

z := x

z := 100

name := 'MBT'



Operators

■ Important operators in PL SQL

- Line ends with operator: ;
- To join two strings: ||
- To accept value: &
- Power **
2**3 means 2 raise to power 3
- In loop we use ..
Example:
For X in 1..5 means
1 to 5
- Non numeric data
(string or date)
is written in single
quote: ''



Accept a value

■ Examples:

- `num1 := &Number1;`

At run time this will prompt as
Follows

Enter a value for Number1:

Whatever value user will enter
here will be assign to variable
`num1`



Accept a value



■ Examples:

- `name := '&Name';`

At run time this will prompt as
Follows



Enter a value for Name:

Whatever value user will enter
here will be assign to variable
name



' ' is used in case if entered data is not
numeric



Display value

- To display on same line:
`dbms_output.put()`
- To display on new line.
`dbms_output.put_line()`
- Here `dbms_output` is a Oracle package its like header file or library in C language.
- `.Put` and `.Put_Line` are functions like `printf` in 'C' language



Display value : Examples

- ```
Dbms_output.put ('Centre for');
Dbms_output.put ('Remote');
Dbms_output.put ('Sensing');
Dbms_output.put_line('');
```

It will display **Centre for Remote Sensing** on same line.

Note :

1. On SQL prompt after Login you need to set one command to see displayed values.  
**SET SERVEROUTPUT ON**
2. It is important that at least once you write `.put_line` after any number of `.put` functions else values are not displayed.



# Display value : Examples

- `Dbms_output.put_line ('Centre for');`  
`Dbms_output.put_line ('Remote ');`  
`Dbms_output.put_line ('Sensing');`  
It will display  
**Mahindra**  
**British**  
**Telecom**  
on different lines.

Note :

1. On SQL prompt after Login you need to set one command to see displayed values.

**SET SERVEROUTPUT ON**



# DML operations in PL-SQL

- All DML operations (Insert/Update/Delete /Select) are to be written in Begin part of the block.
- No change in the Syntax of Insert , Update and Delete , it is same as SQL.
- Select syntax is different then SQL , it contains INTO clause.
- If Select query can return more then one rows then you should always use cursors .



# Select Syntax for a Single Row Query.

- Select column1, column2  
    **INTO Variable1, Variable2**  
From Table Name  
Where condition .....
- The only change is as many columns you want to get from the query you need to declare that many variables and use INTO clause.
- All other parts of query are unchanged
- If Where condition here is such that query will return multiple records then CURSOR should be used. Without that it will give error.



# Data Types in PL SQL

## ■ Scalar Types

- **Char**

CHAR datatype to store fixed-length character data. Maximum size = 2000 bytes

- **Varchar2**

VARCHAR2 datatype to store variable-length character . Maximum size = 4000 bytes

- **Number**

Number types let you store numeric data (integers, real numbers, and floating-point numbers), represent quantities, and do calculations.



# Data Types in PL SQL

## ■ Scalar Types

- **Binary\_Integer**

The BINARY\_INTEGER datatype to store signed integers ( $-2^{31}$  to  $2^{31}$ )

- **Date**

DATE datatype to store fixed-length datetimes

- **Long**

The LONG datatype to store variable-length character strings. The LONG datatype is like the VARCHAR2 datatype, except that the maximum size of a LONG value is 32760 bytes.



# Data Types in PL SQL

## ■ Scalar Types

- **NChar**

To store multi byte fixed length character data. Its same as Char only difference is it is used to store characters of different language like Japanese , chinese etc.

Number of characters it can store depend on language.

- **NVarchar**

To store multi byte variable length character data. Its same as Varchar2 only difference is it is used to store characters of different language like Japanese , chinese etc.

Number of characters it can store depend on language.



# Data Types in PL SQL

## ■ Composite Types

- **Record**

Its like structure in C Language. To be discussed in Second day session.

- **Table**

Its like Array in C Language. To be discussed in detail in Second day session.

This Array type is un-constrained array

- **VArray**

Its like Array in C Language. To be discussed in detail in Fourth day session.

This Array type is constrained array



# Data Types in PL SQL

## ■ Reference Types

- **Ref Cursor**

Its used for dynamic cursor. To be discussed in Second day session.



# Data Types in PL SQL

## ■ **LOB**

- **BLOB**

Binary Large Object A column or variable of type BLOB can store up to 4GB of binary data in each record.

- **CLOB**

Character Large Object A column or variable of type CLOB can store up to 4GB of character data in each record.

- **BFILE**

It can store a file of size 4GB externally outside database for each record and can refer to that from inside the database.



# Data Types in PL SQL

## ■ LOB

- Column or variable of this type can be accessed only using a Oracle package DBMS\_LOB.
- This should be used only if required to store a large amount of data in each record of a table
- You should avoid making un-necessary use of LOB's.
- To be discussed in last session of PL-SQL



# Variable Declaration in PL SQL

- Variables are always declared in DECLARE section of the program.
- Variable Name <Data Type>
- Various way to declare them  
v\_empno Number;  
V\_ename varchar2;  
v\_job Char(10);



# Variable Declaration in PL SQL

- Dynamic and preferred way to declare a variable

Variable Name    TableName.ColName%Type

```
v_empno Emp.Empno%Type;
```

```
V_ename Emp.Ename%Type;
```

```
v_deptno Dept.Deptno%Type;
```

- Advantages of declaring in above way.
  - Variable will always have same datatype as column
  - Any change in column will change the type of variable also, so we need not have to change and recompile the program to run.



# Variable Declaration in PL SQL

- %RowType

Variable Name    TableName%RowType

v\_emp    Emp%RowType;

- Advantages of declaring in above way.
  - Variable will become like a structure variable in C (i.e. v\_emp will have same structure like Emp Table) and you can refer to individual element as follows:  
v\_emp.empno  
v\_emp.ename  
v\_emp.sal



# Variable Declaration in PL SQL

- **Type** : You can also make your own type in program and use in the declare section to declare variable.

Type t\_name is Varchar2(50);

-- now you can make variable of this type

v\_name            t\_name;

v\_name2          t\_name;

v\_name and v\_name2 both will become varchar2(50)



# Conditional Statements

- IF ... Then ... ELSE

```
If <condition1> Then
 <Code>
ELSIF <Condition2> Then
 <Code>
ELSE
 <Code>
END IF;
```

- Note here that for one IF we only need one END IF;
- No END IF is required for ELSIF i.e for one set of IF condition only one END IF; is required



# Conditional Statements

- IF ... Then ... ELSE

```
 If v_deptno = 10 Then
 DBMS_OUTPUT.PUT_LINE
('Accounting');
 ELSIF v_deptno = 20 Then
 DBMS_OUTPUT.PUT_LINE ('ESG');

 ELSE
 DBMS_OUTPUT.PUT_LINE ('Invalid');
 END IF;
```



# Conditional Statements

- CASE : This is available from ORACLE 8i onwards only , not in ORACLE 8 and version prior to that.

CASE

WHEN <Variable> = <Value1> Then  
<Code>

WHEN <Variable> = <Value2> Then  
<Code>

ELSE  
<Code>

END CASE;



# Conditional Statements

- CASE :

CASE

When v\_deptno =10 Then  
DBMS\_OUTPUT.PUT\_LINE  
( 'Accounting' );

When v\_deptno =20 Then  
DBMS\_OUTPUT.PUT\_LINE ( 'ESG' );

ELSE

DBMS\_OUTPUT.PUT\_LINE ( 'Invalid' );

END CASE;



# TYPES OF LOOPS

- Simple Loop

Loop

Exit When <Condition>

<Code>

End Loop;

- Exit when is required to give the condition to end the loop
- It is pre tested as condition is checked first and then code is executed



# TYPES OF LOOPS

- Simple Loop

Loop

Exit When i = 10

```
dbms_output.put_line (i);
```

```
End Loop;
```

--Pre Tested



# TYPES OF LOOPS

- Simple Loop

Loop

    <Code>

    Exit When <Condition>

End Loop;

- Exit when is required to give the condition to end the loop
- It is post tested as condition is checked after the code is executed



# TYPES OF LOOPS

- Simple Loop

Loop

```
dbms_output.put_line (i);
```

```
Exit When i = 10
```

```
End Loop;
```

--Post Tested



# TYPES OF LOOPS

- While Loop

While <Condition>

Loop

<Code>

End Loop;

- While is required for condition to end the Loop
- This is also pre tested.



# TYPES OF LOOPS

## ■ While Loop

**While**  $i < 10$

Loop

dbms\_output.put\_line (i);

End Loop;



# TYPES OF LOOPS

- FOR Loop

FOR <Variable> IN <Min> .. <Max>

Loop

<Code>

End Loop;

- This Loop is used when we know the number of time the loop is to be executed.
- This is also pre tested.



# TYPES OF LOOPS

- FOR Loop

```
FOR i IN 1 .. 100
```

```
Loop
```

```
 <Code>
```

```
End Loop;
```

- This Loop will execute the given code 100 times for  $i = 1$  to 100



# TYPES OF LOOPS

- FOR Loop Reverse

```
FOR i IN Reverse 1 .. 100
```

```
Loop
```

```
<Code>
```

```
End Loop;
```

- This Loop will execute the given code 100 times for  $i = 100$  to 1
- This is reverse i.e from last value to first value