# UNIT 5:

# PL/SQL OBJECTS

## 1. DEFINE: PL/SQL

- PL/SQL stands for Procedural Language extensions to the Structured Query Language.
- PL/SQL is a combination of SQL along with the procedural features of programming languages.
- It was develop by Oracle Corporation in the early 90's to enhance the capabilities of SQL.
- Oracle uses a PL/SQL engine to processes the PL/SQL statements.

## 2. WRITE DOWN ADVANTAGES OF PL/SQL.

1. **Procedural Capabilities**
    - PL/SQL provides procedural capabilities such as **condition checking,** branching and looping.
    - This enables programmer to control execution of a program based on some conditions and user inputs.

2. **Support to variables**
    - PL/SQL supports declaration and use of variables.
    - These variables can be used to store intermediate results of a query or some expression.

3. **Error Handling**

- When an error occurs, user-friendly message can be displayed.
- Execution of program can be controlled instead of abruptly terminating the program.

4. **User Defined Functions**
    - PL/SQL allows you to create your own user defined functions and procedures.

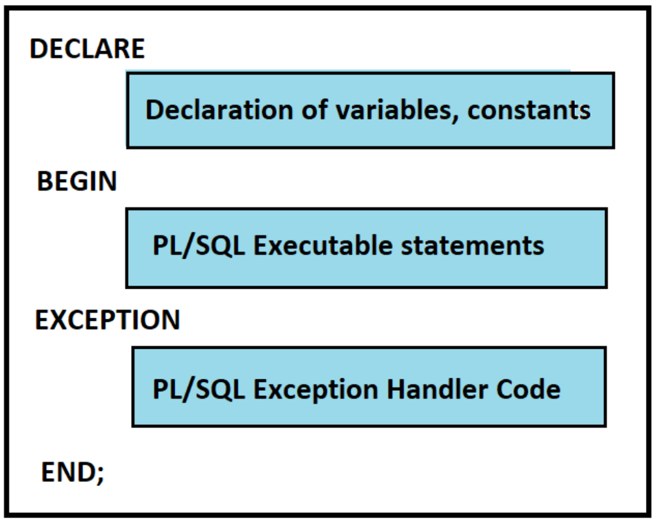
5. **Portability**
    - Programs written in PL/SQL are portable.
    - You can run PL/SQL applications on any operating system and platform where Oracle Database runs.

6. **Sharing of Code**
    - Allows user to store compiled code in database.
    - PL/SQL code can accessed and shared by different applications.

7. **Efficient Execution**
    - PL/SQL sends an entire block of SQL statements to the Oracle engine, where these statements are executed in one go.
    - Reduces network traffic and improves efficiency of execution.

## 3. EXPLAIN PL/SQL BLOCK IN DETAILS.

- PL/SQL Block consists of three sections:
  - The Declaration section
  - The Execution section
  - The Exception (or Error) Handling section


DECLARE
Declaration of variables, constants
BEGIN
PL/SQL Executable statements
EXCEPTION
PL/SQL Exception Handler Code
END;

**DECLARATION SECTION:**

- The Declaration section of a PL/SQL Block starts with **DECLARE** keyword.
- This section is optional.
- It is use to declare variables, constants, cursors etc.

**EXECUTION SECTION:**

- The Execution section of a PL/SQL Block starts **BEGIN** keyword and ends with **END** keyword.
- This is a mandatory section.
- The program logic written in this block.

- It contain various SQL commands & PL/SQL statement like loops, conditional statement and SQL statements form the part of execution section.

## EXCEPTION SECTION:

- The Exception section of a PL/SQL Block starts with **EXCEPTION** keyword**.**
- This section is optional.
- It's a subpart of **EXECUTION** section
- It contain error-handling code.

## 4. WHAT IS CONSTANTS?

- A constant holds a value that once declared, does not change in the program.
- A constant declaration specifies its name, data type, and value, and allocates storage for it.
- A constant is declare using the constant keyword.

**Syntax:**

```
<Constant name> CONSTATNT <datatype>:=value;
```

**E.g.:**

```
PI CONSTATNT number: = 3.14;
```

## 5. DEFINE: LITERALS

- A literal is the same as a constant.
- PL/SQL, literals are case-sensitive.
- PL/SQL supports the following kinds of literals –

| Literals | Description | Example |
|---|---|---|
| Numeric Literals | • Number literals can be up to 38 digits.<br>• Number literals can be either positive or negative numbers | 25, +25, -25, 25e-04<br>25.607 |
| Character Literals | • Character literals are always surrounded by single quotes ('). | 'a', 'B', 'R' ,'*' |
| String Literals | • String literals are group of characters always surrounded by single quotes ('). | 'RDBMS', 'SQL server', 'Oracle Server' |
| BOOLEAN Literals | • Used for storing Boolean value. | TRUE, FALSE, and NULL. |
| Date and Time Literals | • Date and time are enclosed in single quotes (') | '2015-04-30'<br>'2015-04-30 08:13:24' |

## 6. WHAT IS VARIABLE?

- Variable is a container that store value that may be change during the execution of program.
- Variables are declare in Declaration section of the PL/SQL block.
- It can assign valid data type and initialized if necessary.
- **Syntax:**

```
Variable_Name   datatype   [NOT NULL]:=   initial Value;
```

### Assign Value to Variable

- Three ways to assign value in variable.

1. **By assignment operator**
   - Value of constants or return value of expression stored like this way.
   - For e.g.

     **Srno: = 101;**

2. **By reading value from keyboard**
   - Value of variable read from keyboard during the execution of program.
   - To get the value **'&'** sign will be used.
   - For e.g.

     **Srno: = &Srno;**

3. **Selecting or Fetching table data values into Variables**
   - This method allows you to fetch value from the field of table & store inside variable.
   - For E.G.

Select rlno into R from student where name='ekta';

## 7. EXPLAIN DATA TYPE.

- PL/SQL is super set of the SQL.
- Therefore, it supports all the data types provided by SQL.
- In PL/SQL Oracle provides subtypes of the data types.

| Category | Data Type | Sub types/values |
|---|---|---|
| Numerical | NUMBER | BINARY_INTEGER, DEC, DECIMAL, DOUBLE PRECISION, FLOAT, INTEGER, INT, NATURAL, POSITIVE, REAL, SMALLINT |
| Character | CHAR, LONG, VARCHAR2 | CHARACTER, VARCHAR, STRING, NCHAR, NVARCHAR2 |
| Date | DATE | - |
| Binary | RAW, LONG RAW | - |
| Boolean | BOOLEAN | Can have value like TRUE, FALSE and NULL |
| RowID | ROWID | Stores values of address of each record |

## 8. DEFINE DBMS_OUTPUT.PUT_LINE.

- A dbms_output is a package, which provides functions to accumulate information in a buffer.
- A put_line is a function, which display messages on the screen.
- Use || (concatenate operator) to connect string.
- **E.g.**

```
A number: = 20;
DBMS_OUTPUT.PUT_LINE ('the value of a='||a);
```

## 9. HOW TO DEFINE COMMENTS IN PL/SQL.

- A comment can have two forms:
  - **Single line comment:**
    - The comment line begins with a double hyphen (--).
    - The entire line will be treated as a comment.
  - **Multiline comment:**
    - The comment line begins with a slash followed by an asterisk (/*) and ends with a slash (*/).
    - All lines within /*….*/ will be treated as comments.

## 10. EXPLAIN CONDITIONAL STATEMENT OF PL/SQL WITH EXAMPLE.

- PL/SQL provides basic three types of control structure:
    - If....Then....End If
    - If....Then....Else....End If
    - If.... Then...Elseif....End If

### IF.. THEN..... END IF

- The simple IF block that execute the block of statement if the condition is true.
- IF condition becomes false then the statement placed after end if will be execute.
- **Syntax:**

```
If  <Condition>  Then
    <Statements>
End If;
```

- **Example:**

```
Declare
   A Number:=0;
Begin
If A>0 Then
    Dbms_Output.Put_Line(‘ A Is Positive’);
Else
   Dbms_Output.Put_Line(‘ A Is Nagetive’);
End If;
```

```
End;
/
```

## IF.. THEN.. ELSE ..END IF

- The simple IF block that execute the block of statement if the condition is true.
- If the condition becomes false then else part will be executed.

- **Syntax:**

```
IF  <CONDITION>  THEN
            <Statement 1> ;
Else
           <Statement 2>;
END IF;
```

- **Example:**

```
Declare
    A Number:=0;
Begin
If A>0 Then
            Dbms_Output.Put_Line(‘ A Is Positive’);
Else
            Dbms_Output.Put_Line(‘A Is Negative’);
End If;
End;
/
```

## IF.. THEN.. ELSEIF.... END IF

- This structure is used if you need to select an action from several conditions.
- **Syntax:**

```
If  <Condition>  Then
    <Statement1>;
Elseif <Condition2> Then
    <Statement 2>;
Elseif...


Else
    Default Statement;
End If;
```

- **Example:**

```
DECLARE
  A Number(2):=&A;
    B Number(2):=&B;
Begin
  If A>B Then
       Dbms_Output.Put_Line(‘ A Is Big’);
Elsif B>A Then
       Dbms_Output.Put_Line(‘ B Is Big’);
Else
```

```
        Dbms_Output.Put_Line('Both Are Equal');
End If;
End;
/
```

## 11. EXPLAIN GOTO STATEMENT WITH SUITABLE EXAMPLE.

- The GOTO statement allows you to transfer control to a labeled block or statement.
- **Syntax:**

```
GOTO label;
..
..
<< label >>
statement;
```

- **Example:**

```
BEGIN
GOTO second_message; <<first_message>>
        Dbms_Output.Put_Line( 'Hello' );
        GOTO the_end;
<<second_message>>
         Dbms_Output.Put_Line( 'PL/SQL GOTO Demo' );
         GOTO first_message;
<<the_end>>
         Dbms_Output.Put_Line (‘Good bye...’);
END;
```

## 12. EXPLAIN LOOPING STRUCTURE WITH EXAMPLE.

- A loop marks a sequence of statements that has to be repeated.
- PL/SQL supports the following structures for iterative control:
  - LOOP
  - WHILE loop
  - FOR loop

### LOOP

- The basic form of interactive control is the LOOP.
- Statements written between **LOOP** and **END LOOP.**
- The condition is written after **EXIT WHEN** statement.
- You can check condition at the beginning of the loop or end of the loop.

- **Syntax:**

```
Loop
             <Statements>;
             [Exit When <Condition>];
End Loop;
```

- **Example:**

```
Declare
  A Number:=1;
Begin
   Loop
            Dbms_Output.Put_Line(A);
            A:=A+1;
            Exit When A>5;
  End Loop;
End;
/
```

## WHILE LOOP

- A WHILE LOOP statement in PL/SQL programming language repeatedly executes a target statement as long as a given condition is true.
- Statement must be placed between LOOP and END LOOP
- **Syntax:**

```
While <Condition>
Loop
            <Statements>;
End Loop;
```

- **Example:**

```
Declare
  A Number:=1;
Begin
     While A<=5
     Loop
           Dbms_Output.Put_Line(A);
           A:=A+1;
     End Loop;
End;
/
```

## FOR LOOP

- We can use FOR...LOOP when we want the iterations to occur a fixed number of times.
- In for loop variables need not to be declared.
- In for loop increment/decrement need not to write.
- FOR LOOP is incremented by 1 after each iteration.
- Use REVERSE word to decrement value by 1.
- **Syntax:**

```
For Variable In [Reverse] Start..End
Loop
        <Statement>
End Loop;
```

- **Example:**

```
Declare
                A Number:=1;
Begin
       For A In 1..5
       Loop
                        Dbms_Output.Put_Line(A);
       End Loop;
End;
/
```

## 13. EXPLAIN PROCEDURE WITH EXAMPLE.

- A procedure is a group of PL/SQL statements that you can call by name.
- A procedure is a module performing one or more actions , it does not need to return any values
- The user must call a procedure either from a program or manually.
- Generally, you use a procedure to perform an action.
- Procedure contain 2 parts:
    - Procedure header:
        - The header contains the name of the procedure and the parameters or variables passed to the procedure.
    - Procedure body
        - The body contains a declaration section, execution section and exception section similar to a general PL/SQL block.

**Syntax:**

```
CREATE [OR REPLACE] PROCEDURE <PROCEDURE_NAME>
    [Parameter1 {IN, OUT, IN OUT}, [Parameter2 {IN, OUT, IN    OUT},]]
    {IS/AS}
            [CONSTANT / VARIABLE Declaration;]
    BEGIN
            Executable statements/ <procedure body>;
    [EXCEPTION
            Exception handling statements;]
    END < PROCEDURE_NAME>;
```

In syntax:

| | |
|---|---|
| [OR REPLACE] | Allows the modification of an existing procedure |
| <PROCEDURE_NAME> | Specifies the name of the procedure. |
| Parameter | Parameter list contains name, mode and types of the parameters |
| AS/IS | For creating a  standalone procedure |
| <procedure body> | Contains the executable part. |
| EXCEPTION | Contains exception handling |

| | |
|---|---|
| | statements |
| END <PROCEDURE_NAME>; | End of the procedure. |

**Example:**

```
CREATE OR REPLACE PROCEDURE pro_sum (x IN number, y IN number, z OUT
number)
IS
BEGIN
    z: = x + y;
    dbms_output.put_line ('sum of x + y ='||z);
end pro_sum;
/
```

## 14. WHAT IS PARAMETERS? EXPLAIN VARIOUS TYPES OF PARAMETERS.

- The parameter is variable of any valid PL/SQL datatype through which the PL/SQL subprogram exchange the values with the main code.
- There are 3 types of parameters:
  - IN Parameter
  - OUT Parameter
  - IN OUT Parameter

### IN parameter:

- The parameters are of IN type.
- It is use for giving input to the subprograms.
- This value read only type of value. It cannot be change.

### OUT parameter:

- This parameter used for getting output from the subprograms.
- It is a read-write variable inside the subprograms.
- Their values can be changed inside the subprograms.

### IN OUT parameter:

- This parameter used for both giving input and for getting output from the subprograms.
- It is a read-write variable inside the subprograms.
- Their values can be changed inside the subprograms

## 15. COMPARE IN, OUT & IN OUT PARAMETER.

| IN | OUT | IN OUT |
|---|---|---|
| Default mode | Must be specified | Must be specified |
| Value is passed into subprogram | Returned to calling environment | Passed into subprogram; returned to calling environment |
| Formal parameter acts as a constant | Uninitialized variable | Initialized variable |
| Actual parameter can be a literal, expression, constant, or initialized variable | Must be a variable | Must be a variable |
| Can be assigned a default value | Cannot be assigned a default value | Cannot be assigned a default value |

## 16. HOW TO EXECUTE & DELETE PROCEDURE?

- A procedure can be execute by following ways:

```
SQL >     @ <Procedure-file name>.SQL;
          Procedure is successfully created…
                         OR
          Procedure is created with compilation error…
SQL >     SHOW ERRORS;
                         OR
SQL >     SELECT * FROM USER_ERRORS;
SQL >     EXECUTE <Procedure-name> (Parameter Value);
SQL >     SELECT * FROM <Table-name>;
```

### Drop procedure: 3

- Use the DROP PROCEDURE statement to remove a standalone stored procedure from the database.
- Syntax:

```
DROP PROCEDURE <procedure name>;
```

## 17. WHAT IS FUNCTION? EXPLAIN FUNCTION WITH EXAMPLE.

- Function – User Define Functions is similar to stored procedure.
- The PL/SQL function is a named PL/SQL block, which performs one or more specific tasks and must returns a value.
- It is similar to procedure only the difference between procedure & **function is function returns a value.**
- Functions can accept one, many, or no parameters, but a function must have a return clause in the executable section of the function.

**SYNTAX:**

```
CREATE [OR REPLACE] FUNCTION <FUNCTION_NAME>
    [Parameter1 {IN, OUT, IN OUT}, [Parameter2 {IN, OUT, IN    OUT},]] RETURN
<data type>
    {IS/AS}
            [CONSTANT / VARIABLE Declaration;]
    BEGIN
            Executable statements/ <procedure body>;
    [EXCEPTION
            Exception handling statements;]
    END < FUNCTION_NAME>;
```

In syntax:

| | |
|---|---|
| [OR REPLACE] | Allows the modification of an existing function. |
| <FUNCTIONNAME> | Specifies the name of the function. |
| RETURN <data type> | Specify the data type for return variable |
| Parameter | Parameter list contains name, mode and types of the parameters |
| AS/IS | For creating a function. |
| <function body> | Contains the executable part. |

| | |
|---|---|
| EXCEPTION | Contains exception handling statements |
| END <FUNCTION_NAME>; | End of the function |

**Example:**

```
CREATE OR REPLACE FUNCTION fun_cube (x IN number, cb out number)
RETURN number
IS
BEGIN
    cb: =x * x * x;
    Return cb;

end fun_cube;
/
```

## 18. WRITE DOWN THE DIFFERENCE BETWEEN PROCEDURE & FUNCTION.

| Procedure | Function |
|---|---|
| A procedure used to perform certain task in order. | A function used to calculate result using given inputs. |
| A procedure cannot be called by a function | A function can called by a procedure. |
| It supports try-catch blocks. | It does not support try-catch blocks. |
| DML statements can executed within a procedure. | DML statements cannot executed within a function. |
| Procedures always executes as PL SQL statement | Functions executes as part of expression |
| It does not contain return clause in header section | It must contain return clause in header |
| A procedure need not deal with expressions. | A function must deal with expressions. |
| Procedures will not return the value | Functions must return the value. |
| Procedures need to be compiled once, and if necessary, we can call them repeatedly without compiling them every time. | Functions are compiled whenever they are called. |

## 19. EXPLAIN TRIGGER WITH TYPES & EXAMPLE.

- Trigger is a series of PL/SQL statements attached to a database table that execute whenever a triggering event (select, update, insert, delete) occurs.
- The triggers are standalone procedures that are fired implicitly (internally) by the oracle.
- Triggers need not explicitly called, but they are activated (triggered) when a triggering event occurs.

### Syntax:

```
CREATE OR REPLACE TRIGGER <TRIGGER_NAME>
{ BEFORE/AFTER }
{INSERT/UPDATE/DELETE} [OF COLUMN] ON <TABLE_NAME>
[FOR EACH ROW]
[WHEN CONDITION]
[PL/SQL BLOCK]
END <TRIGGER_NAME>;
```

In syntax:

| | |
|---|---|
| **CREATE OR REPLACE TRIGGER** | Creates or replaces an existing trigger with the trigger_name. |
| **BEFORE/AFTER** | Specifies when the trigger will be executed |
| **INSERT/UPDATE/DELETE** | Specifies the DML operation. |
| **ON <TABLE_NAME>** | Specifies the name of the table associated with the trigger |
| **FOR EACH ROW** | This specifies a row-level trigger |
| **PL/SQL BLOCK** | PL/SQL block that provides the operation to be performed as the trigger is fired |

### Example:

```
Set Serveroutput On
Create Or Replace Trigger Tri_Msg
After Update Or Delete On Product
```

```
For Each Row
Begin
If Updating Then
    Dbms_Output.Put_Line('Product Table Updated...');
Elsif Deleting Then
    Dbms_Output.Put_Line('Record Removed From Product');
End If;

End Tri_Msg;
/
```

## 20. EXPLAIN VARIOUS TYPES OF TRIGGERS.

- There are 3 types of triggers:
  1. Row Trigger
  2. Statement Trigger
  3. Before Trigger and After Trigger

### Row parameter:

- A Row Trigger fired each time a row in the table affected by the triggering statement.
- For example:
  - if deletion is defined as a triggering event for a particular table, and a single DELETE statement deletes five rows from that table, the trigger fires five times, once for each row.
- To create row trigger we need to use **FOR EACH ROW** during trigger creation.

### Statement trigger:

- A *statement-level trigger* fires only once for each statement.
- For example:
  - If deletion is defined as a triggering event for a particular table, and a single DELETE statement deletes five rows from that table, the trigger fires once.

### Before and after trigger

- BEFORE triggers run the trigger action before the triggering statement is run.
- AFTER triggers run, the trigger action after the triggering statement is run.

## 21. DIFFERENCE BETWEEN TRIGGER & PROCEDURE.

| TRIGGER | PROCEDURE |
|---|---|
| Trigger don’t accept parameters | Procedure can accept parameters |
| The oracle engine executes a trigger implicitly (automatically fired) | Procedure it must explicitly called by the users. |
| Trigger fired automatically so we need not to use any command. | Execute command used to execute a procedure. |
| The syntax that defines a trigger inside a database is:<br>**CREATE TRIGGER TRIGGER_NAME** | The syntax that defines a procedure inside a database is:<br>**CREATE PROCEDURE PROCEDURE_NAME** |
| Triggers cannot be called inside a procedure. | However, you can call a procedure inside a trigger. |

## 22. HOW TO EXECUTE TRIGGER & REMOVE TRIGGER?

### EXECUTE TRIGGER:

```
SQL>  START TR_EMPDELETE.SQL;  OR  @TR_EMPDELETE.SQL;
                Trigger is created...........
SQL>  DELETE FROM EMP WHERE EMPNO = 100;
SQL>  SELECT * FROM NEWEMP;
```

**If trigger is created with compilation error then execute the following command:**

```
SQL>  SHOW ERRORS;
```

**To view the list of triggers created by user follow following steps:**

```
SQL>  DESCRIBE USER_TRIGGERS;
```

### REMOVE TRIGGER:

- DROP TIGGER command used to remove trigger.
- **Syntax:**

  DROP TRIGGER <TRIGGER_NAME>;
- **Example**:

  DROP TRIGGER TRI_MSG;

## 23. HOW TO APPLY TRIGGER IN DATABASE?

- Three basic steps to apply triggers in database.
  - Triggering Event or Statement
  - Trigger Restriction
  - Trigger Action

### Triggering Event or Statement

- A SQL statement causes a trigger to be fired.
- It can be INSERT, UPDATE or DELETE statement for a specific table.

### Trigger Restriction

- A trigger restriction specifies a Boolean Expression that must be TRUE for the trigger to fire.
- It is an option available for the triggers that are fired for each row.
- A trigger restriction is specified using a WHEN clause.

### Trigger Action:

- A trigger action is the PL/SQL code to be executed when triggering statement is encountered and the value of trigger restriction is TRUE.
- The PL/SQL block contains SQL and PL/SQL statements.

## 24. EXPLAIN :OLD AND :NEW QUALIFIERS WITH EXAMPLE.

- The OLD and NEW qualifiers are used to reference the values of a column before and after the data change, respectively.
- The OLD and NEW qualifiers can be used only with row triggers.
- They cannot be used with statement triggers.
- The OLD and NEW qualifiers must be prefixed with a colon (:) in every SQL and PL/SQL statement except when they are referenced in a WHEN restricting clause.

EXAMPLE:

```
SET SERVEROUTPUT ON
Create Or Replace Trigger Tr_All
Before Insert Or Update Or Delete On Emp
For Each Row
Begin
If Inserting Then
                Insert Into Newemp Values (:NEW.EMPNO,
:NEW.ENAME,:NEW.SAL);
Elsif Updating Then
                Insert Into Newemp Values (:OLD.EMPNO, :OLD.ENAME,
:OLD.SAL);
ELSE
                Insert Into Newemp  Values (:OLD.EMPNO, :OLD.ENAME,
:OLD.SAL);
END IF;
END;
/
```