UNIT - V

PL/SQL: Introduction, Shortcoming in SQL, Structure of PL/SQL, PL/SQL Language Elements, Data Types, Operators Precedence, Control Structure, Steps to Create a PL/SQL, Program, Iterative Control, , Procedure, Function, , Database Triggers, Types of Triggers.

⇨ **1) PL/SQL (Procedural Language/Structured Query Language):**

1. PL/SQL stands for Procedural Language extension of SQL.
2. PL/SQL is a combination of SQL along with the procedural features of programming languages.
3. It was developed by Oracle Corporation in the early 90's to enhance the capabilities ofSQL.

PL/SQL Block Structure:

1. A PL/SQL block contains 1 or more PL/SQL statements.
2. Each PL/SQL program consists of SQL and PL/SQL statements which from a PL/SQLblock.

Syntax:

Declare

Variable declaration

Begin

Process statements

Or execution statement

[Exception

Exception statement]

End;

A PL/SQL block can be divided into four sections. They are

1. Declaration section
2. Begin section
3. Exception section
4. End section

1. Declaration Section:

- Code blocks start with a declaration section
- In this block memory variable and other oracle objects can be declared

- They can be used in SQL statements for data manipulation.

Example:

```
Declare
First_name varhcar2(10);
Num number(10);
```

2. Begin Section:

- It consists of a set of SQL and PL/SQL statements
- It describes process that has to be applied to table data.
- Actual data manipulation, retrieval, looping and branching constructs arespecified in this section.

3. Exception Section:

- This section deals with handling of errors
- That arise during execution of the data manipulation statements
- The errors can arise due to syntax and logic.

4. End Section:

a. This makes the end of a PL/SQL block.

Example:

```
Declare
a number(4);
b number(4);
c number(4);
begin
b:=20;
c:=a+b;
dbms_output.put_line(c);
end;
```

1. **dbms_ouput:** it is a package.
2. That includes a number of procedures and functions that accumulate informationin a buffer so that it can be retrieved later.
3. These functions can also be used to display message.
4. **put_line:** put a piece of information in the package buffer followed by an end-of-line marker.
5. dbms_ouput.put_line('Hello');

⇨ **PL/SQL Language Elements**

There are different elements

1. Character Set
2. Lexical Units

a. Delimiters

b. Identifiers

c. Literals

d. Comments

1. Character Set

1. A PL/SQL program consists of text having specific set of characters.
2. Character set may include the following characters:

a. Alphabets, both in upper case [A–Z] and lower case [a–z]

b. Numeric digits [0–9]

c. Special characters ( ) + − * /< >= ! ~ ^ ; : . _ @ %

d. Blank spaces, tabs, and carriage returns.

2. Lexical Units

1. A line of PL/SQL program contains groups of characters known as lexical units,which can be classified as follows:

A. Delimiters

B. Identifiers

C. Literals

D. Comments

A. Delimiters

a. A *delimiter* is a simple or compound symbol

b. That has a special meaning to PL/SQL.

c. Simple symbol consists of one character

d. Compound symbol consists of more than one character.

B. Identifiers

a. Identifiers are used in the PL/SQL programs

b. To name the PL/SQL program items

i. Like constants, variables, cursors, cursor variables, subprograms,etc.

c. Identifiers can consists of alphabets, numerals, dollar signs,underscores, and number signs only.

d. Any other characters like hyphens, slashes, blank spaces, etc.

C. Literals

a. A literal is an explicitly defined character, string, numeric, or Booleanvalue,

D. Comments

1. Comments are used in the PL/SQL program
2. It used to improve the readability and understandability of a program.
3. A comment can appear anywhere in the program code.
4. The compiler ignores comments.
5. Generally, comments are used to describe the purpose and use of eachcode segment.

Ex: /* Hello World! This is an example of multiline comments in PL/SQL */

Q) Introduction to PL/SQL data types

Each value in PL/SQL such as a constant, variable and parameter has a data type that determines the storage format, valid values, and allowed operations.

PL/SQL has two kinds of data types: scalar and composite. The scalar types are types that store single values such as number, Boolean, character, and datetime whereas the composite types are types that store multiple values, for example, record and collection.

This tutorial explains the scalar data types that store values with no internal components.

PL/SQL divides the scalar data types into four families:

- Number
- Boolean
- Character
- Datetime

A scalar data type may have subtypes. A subtype is a data type that is a subset of another data type, which is its base type. A subtype further defines a base type by restricting the value or size of the base data type.

Numeric data types

The numeric data types represent real numbers, integers, and floating-point numbers. They are stored as NUMBER, IEEE floating-point storage types (BINARY_FLOAT and BINARY_DOUBLE), and PLS_INTEGER.

Boolean data type

The BOOLEAN datatype has three data values: TRUE, FALSE, and NULL. Boolean values are typically used in control flow structure such as IF-THEN, CASE, and loop statements like LOOP, FOR LOOP, and WHILE LOOP.

SQL does not have the BOOLEAN data type, therefore, you cannot:

- Assign a BOOLEAN value to a table column.
- Select the value from a table column into a BOOLEAN variable.
- Use a BOOLEAN value in a SQL function.
- Use a BOOLEAN expression in a SQL statement.
- Use a BOOLEAN value in the DBMS_OUTPUT.PUTLINE and DBMS_OUTPUT.PUT subprograms.

Character data types

The character data types represent alphanumeric text. PL/SQL uses the SQL character data types such as CHAR, VARCHAR2, LONG, RAW, LONG RAW, ROWID, and UROWID.

- CHAR(n) is a fixed-length character type whose length is from 1 to 32,767 bytes.
- VARCHAR2(n) is varying length character data from 1 to 32,767 bytes.

Datetime data types

The date time data types represent dates, timestamp with or without time zone and intervals. PL/SQL datetime data types are DATE, TIMESTAMP, TIMESTAMP WITH TIME ZONE, TIMESTAMP WITH LOCAL TIME ZONE, INTERVAL YEAR TO MONTH, and INTERVAL DAY TO SECOND.

Data type	Description
Char	Character value of fixed length
Varchar2	Variable length character value
Number	Numeric values
Date	Date values
% type	Inherits the data type from a variable that you declaredpreviously in database table.

% row type	It is used to declare variable to keep a single record, since a record is nothing but collection of column. This is also known ascomposite data type.
Boolean	Boolean data type can be used to store the values true, false ornull.

PL SQL Operator Operator Precedence

Operator	Description
**	Exponentiation
+, -	Identity, negation (unary operation)
*, /	Multiplication, division
+, -, ||	Addition, subtraction, concatenation
=, <, >, <=, >=, <>, !=, ~= IS NULL, LIKE, BETWEEN, IN	Comparison
NOT	Logical negation
AND	Conjunction
OR	Inclusion

Q) Explain Control Structures in PL/SQL?

The follow of control statements can be classified into the following categories.

1. Conditional control
2. Iterative control
3. Sequential control

Conditional control

1. Conditional control, which run different statements for different data values.
2. It check the condition for single time only even it is true or false

3. The conditional control statements are If and case.

a. IF

i. If then statement

ii. If then else statement

iii. If then else if statement

iv. Nested if statement

b. Case

- **If statement**

If condition is true it can execute statement 1 to statement n otherwise itcannot execute statement 1 to statement n.

Syntax:

```
if(condition) then
Statement 1;
…… Statement n;
End if;
```

Example:

```
DECLARE
a number:=&a;
BEGIN
if(a<10)then
  dbms_output.put_line('welcome to pl/sql');
end if;
END;
```

- **If then else statement**

If condition is true it can execute statement 1. If the condition is false itexecute else statement or execute statement 2.

Syntax:

```
if(condition) then
Statement 1;
else
end if;
statement 2;
```

Example:

```
declare
a integer;
b  integer;        begin
a:=&a; /* it take run time values*/
b:=&b; /* it take run time values*/
if(a>b) then
   dbms_output.put_line('A is big');
else
dbms_output.put_line('b is big');
end if;
end;
```

- **If then else if statement**

If condition is true it can execute statement 1. If the condition is false its again chek for another condition if it is true it can execute statement 2. Other execute else statement or execute statement 3.

Syntax:

```
if(condition) then
Statement 1;
elsif(condition) then
statement 2;
else
statement 3;
end if;
```

Example:

```
declare
a integer; b integer;
c  integer;
begin
a:=&a;
b:=&b;
c:=&c;
```

```
if(a>b and a>c)
then dbms_output.put_line('a is big');
elsif(b>c) then
dbms_output.put_line('b is big');
else
dbms_output.put_line('c is big');
end if;
end;
```

- **Nested if statement**

Inner if is called nested if. if condition is false it execute stamen 3. Other wise it is true its check for another condition if it is true it execute statement 1 other wise execute statement 2.

Syntax:

```
if(condition) then
if(condition) then
statement 1;
else
statement 2;
end if;else
statement 3;
end if;
```

- **Case:**

Syntax

```
case variable_name
When value1 then stmt; When value2  then stmt;
……Else stmt;
End case;
```

1. The case statement runs the first statements for which value equals variablename remaining conditions are not evaluated.
2. If no value equals to variable name, the case statements runs else statements.

Ex:

```
Declare
Grade char:='&grade';
Begin
case grade
when 'A' then
dbms_output.put_line('Excellent');
when 'B' then
dbms_output.put_line('Very good');
when 'C' then
dbms_output.put_line('Good');
when 'D' then
 dbms_output.put_line('Fair');
when 'F' then
dbms_output.put_line('poor');
else
dbms_output.put_line('no such grade');
end case;
end;
```

Q) Iterative control (or) Loop controls

The PL/SQL loops are used to repeat the execution of one or more statements for specified number of times. These are also known as iterative control statements.

Syntax for a basic loop:

```
LOOP
 Sequence of statements;
END LOOP;
```

Types of PL/SQL Loops

There are 3 types of PL/SQL Loops.

1. Basic Loop / Exit Loop
2. While Loop
3. For Loop

PL/SQL Exit Loop (Basic Loop)

PL/SQL exit loop is used when a set of statements is to be executed at least once before the termination of the loop. There must be an EXIT condition specified in the loop, otherwise the loop will get into an infinite number of iterations. After the occurrence of EXIT condition, the process exits the loop.

Syntax of basic loop:

```
LOOP
  Sequence of statements;
END LOOP;
```

Syntax of exit loop:

```
LOOP
statements;
EXIT;
{or EXIT WHEN condition;}  END LOOP;
```

Follow these steps while using PL/SQL Exit Loop.

- Initialize a variable before the loop body
- Increment the variable in the loop.
- You should use EXIT WHEN statement to exit from the Loop. Otherwise the EXIT statement without WHEN condition, the statements in the Loop is executed only once.

Example :

```
DECLARE
i NUMBER := 1;
BEGIN
LOOP
EXIT WHEN i>10;
DBMS_OUTPUT.PUT_LINE(i);
i := i+1;
END LOOP;
END;
```

PL/SQL While Loop

PL/SQL while loop is used when a set of statements has to be executed as long as a condition is true, the While loop is used. The condition is decided at the beginning of each iteration and continues until the condition becomes false.

Syntax of while loop:

```
WHILE <condition>
 LOOP statements;
END LOOP;
```

Follow these steps while using PL/SQL WHILE Loop.

- Initialize a variable before the loop body.
- Increment the variable in the loop.
- You can use EXIT WHEN statements and EXIT statements in While loop but it is not done often.

Example of PL/SQL While Loop

```
DECLARE
i INTEGER := 1;
BEGIN
```

```
WHILE i <= 10 LOOP
DBMS_OUTPUT.PUT_LINE(i);
i := i+1;
END LOOP;
END;
```

PL/SQL FOR Loop

PL/SQL for loop is used when when you want to execute a set of statements for a predetermined number of times. The loop is iterated between the start and end integer values. The counter is always incremented by 1 and once the counter reaches the value of end integer, the loop ends.

Syntax of for loop:

```
FOR counter IN initial_value .. final_value LOOP
  LOOP statements;
END LOOP;
```

- initial_value : Start integer value
- final_value : End integer value

Follow these steps while using PL/SQL WHILE Loop.

- You don't need to declare the counter variable explicitly because it is declared implicitly in the declaration section.
- The counter variable is incremented by 1 and does not need to be incremented explicitly.
- You can use EXIT WHEN statements and EXIT statements in FOR Loops but it is not done often.

PL/SQL For Loop Example

```
DECLARE
VAR1 NUMBER;
BEGIN
VAR1:=10;
FOR VAR2 IN 1..10
LOOP
DBMS_OUTPUT.PUT_LINE (VAR1*VAR2);
END LOOP;  END;
```

Q) PL/SQL Procedure

The PL/SQL stored procedure or simply a procedure is a PL/SQL block which performs one or more specific tasks. It is just like procedures in other programming languages.

The procedure contains a header and a body.

- **Header:** The header contains the name of the procedure and the parameters or variables passed to the procedure.
- **Body:** The body contains a declaration section, execution section and exception section similar to a general PL/SQL block.

Creating a Procedure

A procedure is created with the **CREATE OR REPLACE PROCEDURE** statement. The simplified syntax for the CREATE OR REPLACE PROCEDURE statement is as follows −

```
CREATE [OR REPLACE] PROCEDURE procedure_name
[(parameter_name [IN | OUT | IN OUT] type [, ...])]
{IS | AS}
BEGIN
  < procedure_body >
END procedure_name;
```

Where,

- *procedure-name* specifies the name of the procedure.
- [OR REPLACE] option allows the modification of an existing procedure.
- The optional parameter list contains name, mode and types of the parameters. **IN** represents the value that will be passed from outside and OUT represents the parameter that will be used to return a value outside of the procedure.
- *procedure-body* contains the executable part.
- The AS keyword is used instead of the IS keyword for creating a standalone procedure.

Example :

The following example creates a simple procedure that displays the string 'Hello World!' on the screen when executed.

```
CREATE OR REPLACE PROCEDURE greetings
AS
BEGIN
  dbms_output.put_line('Hello World!');
END;
/
```

When the above code is executed using the SQL prompt, it will produce the following result −

Procedure created.

Executing a Standalone Procedure

A standalone procedure can be called in two ways −

- Using the **EXECUTE** keyword
- Calling the name of the procedure from a PL/SQL block

The above procedure named '**greetings**' can be called with the EXECUTE keyword as −

EXECUTE greetings;

The above call will display −

Hello World

PL/SQL procedure successfully completed.

The procedure can also be called from another PL/SQL block −

```
BEGIN
  greetings;
END;
/
```

The above call will display − **Hello World .** PL/SQL procedure successfully completed.

Deleting a Standalone Procedure

A standalone procedure is deleted with the **DROP PROCEDURE** statement. Syntax for deleting a procedure is −

DROP PROCEDURE procedure-name;

You can drop the greetings procedure by using the following statement −

DROP PROCEDURE greetings;

Parameter Modes in PL/SQL Subprograms

1. **IN parameters:** The IN parameter can be referenced by the procedure or function. The value of the parameter cannot be overwritten by the procedure or the function.
2. **OUT parameters:** The OUT parameter cannot be referenced by the procedure or function, but the value of the parameter can be overwritten by the procedure or function.
3. **INOUT parameters:** The INOUT parameter can be referenced by the procedure or function and the value of the parameter can be overwritten by the procedure or function.

Q) PL/SQL Function

The PL/SQL Function is very similar to PL/SQL Procedure. The main difference between procedure and a function is, a function must always return a value, and on the other hand a procedure may or may not return a value. Except this, all the other things of PL/SQL procedure are true for PL/SQL function too.

Syntax to create a function:

CREATE [OR REPLACE] **FUNCTION** function_name [parameters]
[(parameter_name [IN | **OUT** | IN **OUT**] type [, ...])]
RETURN return_datatype
{**IS** | **AS**}
BEGIN
  < function_body >
END [function_name];

Here:

- **Function_name:** specifies the name of the function.
- **[OR REPLACE]** option allows modifying an existing function.
- The **optional parameter list** contains name, mode and types of the parameters.
- **IN** represents that value will be passed from outside and OUT represents that this parameter will be used to return a value outside of the procedure.

The function must contain a return statement.

- RETURN clause specifies that data type you are going to return from the function.
- Function_body contains the executable part.
- The AS keyword is used instead of the IS keyword for creating a standalone function.

EXAMPLE :

```
create or replace function adder(n1 in number, n2 in number)
return number
is
n3 number(8);
begin
n3 :=n1+n2;
return n3;
end;
/
```

Calling PL/SQL Function: While creating a function, you have to give a definition of what the function has to do. To use a function, you will have to call that function to perform the defined task. Once the function is called, the program control is transferred to the called function.

After the successful completion of the defined task, the call function returns program control back to the main program.

To call a function you have to pass the required parameters along with function name and if function returns a value then you can store returned value

Example

Create Function:

```
CREATE OR REPLACE FUNCTION totalCustomers
RETURN number IS
  total number(2) := 0;
BEGIN
  SELECT count(*) into total
  FROM customers;
```

```
   RETURN total;
END;
/
```

After the execution of above code, you will get the following result.

Function created.

Calling function code:

```
DECLARE
   c number(2);
BEGIN
   c := totalCustomers();
   dbms_output.put_line('Total no. of Customers: ' || c);
END;
/
```

PL/SQL Drop Function

If you want to remove your created function from the database, you should use the following syntax.

```
DROP FUNCTION function_name;
```

Q) TRIGGERS AND ITS TYPES

Triggers are stored programs, which are automatically executed or fired when some events occur. Triggers are, in fact, written to be executed in response to any of the following events −

- A **database manipulation (DML)** statement (DELETE, INSERT, or UPDATE)
- A **database definition (DDL)** statement (CREATE, ALTER, or DROP).
- A **database operation** (SERVER ERROR, LOGON, LOGOFF, STARTUP, or SHUTDOWN).

Triggers can be defined on the table, view, schema, or database with which the event is associated.

Benefits of Triggers

Triggers can be written for the following purposes −

- Generating some derived column values automatically
- Enforcing referential integrity
- Event logging and storing information on table access

- Auditing
- Synchronous replication of tables
- Imposing security authorizations
- Preventing invalid transactions

Creating a trigger:

Syntax for creating trigger:

CREATE [OR REPLACE] **TRIGGER** trigger_name
{BEFORE | **AFTER** | **INSTEAD OF** }
{**INSERT** [OR] | **UPDATE** [OR] | **DELETE**}
[**OF** col_name]
ON table_name
[REFERENCING OLD **AS** o NEW **AS** n]
[**FOR** EACH ROW]
WHEN (condition)
DECLARE
Declaration-statements
BEGIN
Executable-statements
EXCEPTION
Exception-handling-statements
END;

Here,

- CREATE [OR REPLACE] TRIGGER trigger_name: It creates or replaces an existing trigger with the trigger_name.
- {BEFORE | AFTER | INSTEAD OF} : This specifies when the trigger would be executed. The INSTEAD OF clause is used for creating trigger on a view.
- {INSERT [OR] | UPDATE [OR] | DELETE}: This specifies the DML operation.
- [OF col_name]: This specifies the column name that would be updated.
- [ON table_name]: This specifies the name of the table associated with the trigger.

- [REFERENCING OLD AS o NEW AS n]: This allows you to refer new and old values for various DML statements, like INSERT, UPDATE, and DELETE.
- [FOR EACH ROW]: This specifies a row level trigger, i.e., the trigger would be executed for each row being affected. Otherwise the trigger will execute just once when the SQL statement is executed, which is called a table level trigger.
- WHEN (condition): This provides a condition for rows for which the trigger would fire. This clause is valid only for row level triggers.

Type of Triggers

1. **BEFORE Trigger**: BEFORE trigger execute before the triggering DML statement (INSERT, UPDATE, DELETE) execute. Triggering SQL statement is may or may not execute, depending on the BEFORE trigger conditions block.
2. **AFTER Trigger**: AFTER trigger execute after the triggering DML statement (INSERT, UPDATE, DELETE) executed. Triggering SQL statement is execute as soon as followed by the code of trigger before performing Database operation.
3. **ROW Trigger**: ROW trigger fire for each and every record which are performing INSERT, UPDATE, DELETE from the database table. If row deleting is define as trigger event, when trigger file, deletes the five rows each times from the table.
4. **Statement Trigger**: Statement trigger fire only once for each statement. If row deleting is define as trigger event, when trigger file, deletes the five rows at once from the table.
5. **Combination Trigger**: Combination trigger are combination of two trigger type,
    1. **Before Statement Trigger**: Trigger fire only once for each statement before the triggering DML statement.
    2. **Before Row Trigger** : Trigger fire for each and every record before the triggering DML statement.
    3. **After Statement Trigger**: Trigger fire only once for each statement after the triggering DML statement executing.
    4. **After Row Trigger**: Trigger fire for each and every record after the triggering DML statement executing.