**What Are Block Types In PL/SQL**

PL/SQL stands for procedural language-standard query language. It is a significant member of Oracle programming tool set which is extensively used to code server side programming. Similar to SQL language PL/SQL is also a **case-insensitive** programming language.

**Blocks**

Generally a program written in PL/SQL language is divided into blocks. **We can say blocks are basic programming units in PL/SQL programming language.**

PL/SQL Blocks contain set of instructions for oracle to execute, display information to the screen, write data to file, call other programs, manipulate data and many more.

**Does Blocks supports DDL statements?**

Yes, PL/SQL blocks support all DML statements and using Native Dynamic SQL (NDS) or they can run DDL statements using the build in DBMS\_SQL package.

**Types of PL/SQL Blocks**

There are two types of blocks in PL/SQL

1. Anonymous Block
2. Named Block

**Anonymous Block**

As the title suggests these anonymous blocks do not have any names as a result they cannot be stored in database and referenced later.

**Named Block**

On the other hand Named PL/SQL blocks are the one that have names and are used when creating subroutines such as procedures, functions and packages. These subroutines then can be stored in the database and referenced by their name later.

Both type of PL/SQL blocks are further divided into 3 different sections which are:

1. The Declaration Section
2. The Execution Section and
3. The Exception-handling Section

The Execution Section is the only mandatory section of block whereas Declaration and Exception Handling sections are optional.

**Basic prototype of Anonymous PL/SQL Block**

***DECLARE*** *Declaration Statements****BEGIN*** *Executable statements****Exception*** *Exception handling statements****END****;*

**Declaration Section**

This is the first section of PL/SQL block which contains definition of PL/SQL identifiers such as variables, Constants, cursors and so on. You can say this is the place where all local variables used in the program are defined and documented.

Example 1

***DECLARE*** *Var\_first\_name VARCHAR2(30);  
Var\_last\_name   VARCHAR2(30);  
Con\_flag        CONSTANT        NUMBER:=0;*

The above example shows declaration section of an anonymous block. It begins with keyword ***declare*** and contains two variables var\_first\_name and var\_last\_name and one constant con\_flag. Notice that semicolon terminates each declaration.

**Execution Section**

This section contains executable statements that allow you to manipulate the variables that have been declared in the declaration section. The content of this section must be complete to allow the block to compile. By complete I mean complete set of instruction for the PL/SQL engine must be between BEGIN and END keyword.

The execution Section of any PL/SQL block always begins with the Keyword BEGIN and ends with the Keyword END.

This is the only mandatory section in PL/SQL block. This section supports all DML commands and SQL\*PLUS built-in functions and using Native Dynamic SQL (NDS)  or using DMBS\_SQL built-in package it also supports DDL commands.

Example 2

***BEGIN******SELECT****first\_name, last\_name****INTO****var\_first\_name,  
var\_last\_name****FROM****employees WHERE employee\_id =100;****DBMS\_OUTPUT****.****PUT\_LINE*** *(‘Employee Name ’||var\_first\_name||’ ‘||var\_last\_name);****END****;*

This is very simple program where I fetched the value of first name and last name column from employees table where employee id is 100 and stored it into the variable var\_first\_name and var\_last\_name which we declared in our first example.

**Exception-Handling Section**

This is the last section of PL/SQL block which is optional like the declaration block. This section contains statements that are executed when a runtime error occurs within the block.

Runtime error occurs while the program is running and cannot be detected by the PL/SQL compiler. When a runtime error occurs, controlled is pass to the exception handling section of the block the error is evaluated and specific exception is raised.

Example 3

***EXCEPTION*** *WHEN NO\_DATA\_FOUND THEN      DBMS\_OUTPUT.PUT\_LINE (‘No Employee Found with ’||employee\_id);*

# Variables Declaration Initialization In PL/SQL

Variables are place holders in the computer’s main memory which hold some data. Every variable has a name which is user defined, also a data type that defines the nature of data a variable can hold and the total amount of space they can have in main memory along with some value. Like every other programming language in PL/SQL also we first need to declare a variable before using it.

## ****Variable Declaration****

All the variable declaration must be done in **Declare** **Section** of the PL/SQL block. As soon as you declare a variable, the compiler will allocate the memory according to the data type to that variable.  Though you can assign value to the variable either in declare section or in execution section of your PL/SQL block but the **declaration must be done in declare section**.

Here is a simple program to understand this.

***SET SERVEROUTPUT ON;  
DECLARE  
Test\_var1 NUMBER;  — Declaring variable Test\_var  
BEGIN  
Test\_var1:= 10;  
DBMS\_OUTPUT.PUT\_LINE (Test\_var1);  
END;***

This is a very simple program in which I first declared a variable by the name of test\_var1 which has data type number in declaration section and later in execution section I initialized it and assigned a numeric value 10 and then using DBMS\_OUTPUT statement I displayed the value of this variable.

## ****Assignment Operator (:=)****

If you have noticed in the above program that unlike conventional assignment operators in other programming language which is Equal to (=) operator, here we used colon ( : ) with equal to (=) operator for assigning the value to the variable.

Yes in PL/SQL the combo of colon and equal to operator (:=) works as assignment operator which is very different from the other programming languages.

**Note: There is no space between colon and equal to operator.**

## ****Variable Initialization****

Two main questions which I am going to address in this section are

1. Where can we initialize the variables?
2. Different ways of initializing variables in PL/SQL program.

### ****So let’s start with the first question which is where can we initialize the variables in PL/SQL program?****

Though it’s mandatory to **declare** all the variables of your programs in **declaration section** of your PL/SQL block but initializing them and assigning them some value in execution section is not mandatory. This means that you can initialize or say assign values to your variables anywhere in your program.

You can initialize a variable in declaration section while creating it or you can initialize the variable in execution section as we did in the example 1. This is the answer to the first question which I have given you in as simple a language as possible.

### ****Now the second question is what are the different ways of initializing a variable in PL/SQL program?****

Variable initialization means assigning some value to the variable which you previously declared. There are two ways of assigning value to the variable.

1. First is the direct way of giving value to the variable. We have seen this demonstrated in the previous example where we assign integer 10 to the variable test\_var1.
2. Second way is by fetching value from the column of a row of a table and assigning that value to the variable.

So let’s see some examples and try to understand the above concept.

#### **Example 2.**

Declaring variable in declaration section and assigning value by direct way.

**DECLARE****var\_test1 VARCHAR2(30) := ‘RebellionRider’; –Declare & initialize the variable at same time****BEGIN****DBMS\_OUTPUT.PUT\_LINE(var\_test1);****END;**

A very simple program where I declared a variable by the name of var\_test with data type VARCHAR2 and data width 30 also I initialized this variable and assigned a string ‘RebellionRider’ right after declaring it in the declaration section.

#### **Example 3**

In this example we will declare the variable in declaration section and initialize it by fetching value from the table employees of HR user.

**DECLARE****var\_salary NUMBER(5);****BEGIN****SELECT salary INTO var\_salary FROM employees WHERE employee\_id=100;****DBMS\_OUTPUT.put\_line(‘Salary is ‘||var\_salary);****END;**

Here in this simple PL/ SQL code I am fetching the salary of an employee whose employee id is 100 from the employees table of HR user into the variable var\_salary;

Whenever you want to store value from a column of a row into the variable you use INTO clause of SELECT statement as I did here. INTO clause signifies that you want to store the value of salary column into the variable var salary.

Similarly if you want you can store the value from two columns of a row into two variables, let’s see how.

**DECLAREvar\_salary NUMBER(5);var\_fname VARCHAR2 (15);BEGINSELECT salary, first\_name INTO var\_salary, var\_fnameFROM employees WHERE employee\_id=100;DBMS\_OUTPUT.put\_line (‘Salary is ‘||var\_salary);DBMS\_OUTPUT.put\_line (‘Name is ‘||var\_fname); END;**

To store two distinguished values we need two different variables thus I declared two variables var salary and var fname in declaration section. I stored value from salary column into var salary and value from first name into var fname column using Select statement in execution section.

Always remember here that mapping is one on one which means the value from first column in the Select list will be stored in the first variable in the INTO list as happened here. Value from salary column stored in var salary variable and value from 2nd column first name gets stored in 2nd variable which is var fname similarly you can have as many columns as you want.

# SELECT INTO Statement In PL/SQL

In the previous tutorial we learnt about variables as well as how to declare and initialize them. There we saw two different examples of direct initialization. Here in this tutorial we will see another way of initializing the variable using SELECT INTO statement.

The SELECT INTO statement retrieves data from one or more database tables, and assigns the selected values to variables or collections.

### Syntax

**SELECT column1, column2…. Column n INTO variable1, variable2… Variable n FROM table\_name WHERE <expression>;**

Now let’s see some examples of initializing a variable by fetching values from tables of your database. For the demonstration I will use the Employees table of HR sample Schema.

#### **Example 1**

**DECLARE****v\_salary NUMBER(8);**

As I mentioned in my previous tutorial, that every variable must be declared prior to its use and we can only declare a variable in declaration section of PL/SQL block. In the above demonstration I declared a variable by the name of v\_salary which has data type NUMBER and Data width 8. One thing you must take care while declaring variable here is that the data type and data width of your variable and the column whose value you want to fetch must match.

**BEGIN****SELECT salary INTO v\_salary FROM employees****WHERE employee\_id = 100;****DBMS\_OUTPUT.PUT\_LINE (v\_salary);****END;**

This is the execution section of our anonymous block. This section contains our select statement. This select statement is retrieving salary of the employee whose employee id is 100 from employees table and storing it into the variable v\_salary.

The variable v\_salary which we declare above in the declaration section is capable of holding single data at a time thus make sure your select statement must return only single data. This you can ensure by using WHERE clause of your SELECT statement as I did in this example.

Let’s put all the parts together and see the complete anonymous block.

**DECLARE****v\_salary NUMBER(8);****BEGIN****SELECT salary INTO v\_salary FROM employees****WHERE employee\_id = 100;****DBMS\_OUTPUT.PUT\_LINE (v\_salary);****END;**

#### **Example 2.**

Fetch data from multiple column and store it into multiple variables.

Suppose along with Salary you also want to display the first name of the employee using PL/SQL. In this case we will need two different variables as we want to fetch data from two different columns of the table first name and salary. Let’s see the example

**DECLARE****v\_salary      NUMBER(8);****v\_fname       VARCHAR2 (20);****BEGIN****SELECT first\_name, salary INTO v\_fname, v\_salary FROM employees****WHERE employee\_id =100;****DBMS\_OUTPUT.PUT\_LINE(v\_fname||’ has salary ‘||v\_salary);****END;**

In this query we have two variables v\_salary which will hold the value from salary column and v\_fname which will hold the value from first name column of employees table. The select statement is very similar to the previous one except the one extra column with first name and variable v\_fname.

This select statement will return the first name and salary of the employee whose employee id is 100 and then those values will be stored into our variable v\_ fname and v\_salary. Few things which you must take care here are:

1. As I explained a while ago that variable v\_fname and v\_salary can hold only one data at a time thus make sure your select statement will return data from one row at a time. You can ensure this by using WHERE clause.
2. The value from first name and salary columns will be stored into variable v\_fname and v\_salary respectively hence you should always make sure that the data type and data width of the variable matches that of the columns.

# Anchored Datatype In PL/SQL

Anchored data types are those data type which you assign to a variable based on a database object. They are called anchored data type because unlike the variable data type it is not dependent on that of any underlying object.

### ****Syntax****

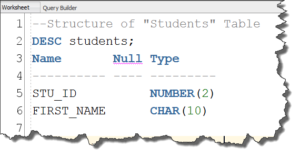
**variable\_name typed-attribute%type**

Where variable name is user defined name given to a variable and type attribute can be anything such as previously declared PL/SQL variable or column of a table. And at the end %type is the direct reference to the underlying database object.

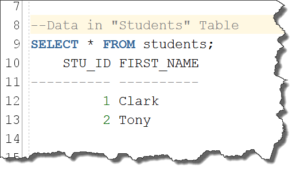
### ****Examples****

For the demonstration I have created a table by the name of Students which has two columns

Stu\_id with data type Number and data width 2 and First\_name with data type varchar2 and data width 8.



I have also inserted two rows into this table.



#### **Example 1: How to Declare a variable with Anchored Datatype**

Next I will write an anonymous block where I will declare a variable with anchored data type and then initialize that variable by fetching value from this table.

So let’s do it.

**SET SERVEROUTPUT ON;DECLAREv\_fname students.first\_name%TYPE;**

Here In the declaration section I have declared a variable by the name of v\_fname with identical data type as the column First Name of table Students. This means that the data type of variable v\_fname will be the varchar2 with data width 8. This is the data type and data width of the column first name of our table students.

So let’s add execution section to this anonymous PL/SQL block and initialize this variable v\_fname by fetching data from the table students.

***BEGIN  
SELECT first\_name INTO v\_fname FROM students WHERE stu\_id =1; DBMS\_OUTPUT.PUT\_LINE (v\_fname); END;***

Here in this execution block I have a Select… Into statement using which I am fetching first name of the student whose stu\_id is 1 and storing it into our variable v\_fname.

That’s how we declare a variable with anchored data type.

**What Are Constants In PL/SQL**

Like several other programming languages, the constant in PL/SQL is also a user defined identifier whose value remains unchanged throughout the program.  Like variables in PL/SQL constants also need to be declared prior to their use. Furthermore you can only declare them in the declaration section of your PL/SQL block.

Syntax

PL/SQL has its own way of declaring a constant. To learn how to declare a constant in PL/SQL let’s quickly take a look at the syntax. This is our syntax:

***constant\_name CONSTANT datatype (data-width) :=  value;***

First you need to give a valid name to your constant followed by keyword CONSTANT that indicates the declaration of a constant in your program. Then you have to specify the data type and data width for your constant followed by the assignment operator and the value which you want to assign to your constant.

**Note here:**  
**You must initialize a constant at its declaration. You have to initialize your constant at the time of its creation in declaration section of your PL/SQL block. You cannot initialize it anywhere else.**

First example will demonstrate how to declare and initialize a constant.

***SET SERVEROUTPUT ON;******DECLARE******v\_pi     CONSTANT NUMBER(7,6) := 3.141592;******BEGIN******DBMS\_OUTPUT.PUT\_LINE (v\_pi);******END;***

This is a simple example of Constant declaration and initialization. Here in declaration section I have declared a constant v\_pi and initialized it with the approximate value of pi. In the execution section we have our DBMS output statement which is displaying the value stored into our constant.

This is the proper way of declaring and initializing a constant in PL/SQL. We have two more attributes of PL/SQL constants to discuss which are “DEFAULT” and “NOT NULL”.

**DEFAULT**

You can use default keyword instead of assignment operator to initialize the constant in PL/SQL.  Let’s do an example and see how to initialize a constant using DEFAULT keyword.

***DECLARE******v\_pi CONSTANT NUMBER(7,6) DEFAULT 3.1415926;******BEGIN******DBMS\_OUTPUT.PUT\_LINE(v\_pi);******END;******/***

Same code just this time I used keyword DEFAULT instead of assignment operator for initializing the constant.

**NOT NULL**

Next attribute is NOT NULL. Using this attribute you can impose NOT NULL constraint while declaring constants as well as variables. This will prevent you from assigning NULL values to your constants or variables.

To impose not null constraint simply write NOT NULL keyword before the Keyword default or before assignment operator in case you have used it. Let me show you how

***DECLARE******v\_pi CONSTANT NUMBER(7,6) NOT NULL DEFAULT 3.1415926;******BEGIN******DBMS\_OUTPUT.PUT\_LINE (v\_pi);******END;/***

That’s all you have to do.

# Bind Variables In PL/SQL

## ****How To Create, Declare, Initialize and Display Bind Variables in PL/SQL****

There are two types of variables in Oracle database.

1. User variables. Discussed in [PL/SQL Tutorial 2](http://www.rebellionrider.com/variables-declaration-initialization-in-pl-sql-by-rebellionrider-manish-sharma/)
2. Bind variables a.k.a Host variables.

Unlike user variables which can only be declared inside the declaration section of PL/SQL block you can declare bind variable anywhere in the host environment and that is the reason why we also refer bind variables as host variable.

**Definition**  
Bind variables in Oracle database can be defined as the variables that we create in SQL\* PLUS and then reference in PL/SQL.  ~ Oracle Docs

## ****How To Declare a Bind Variable (Variable command)****

Let’s see how to create or say declare a bind variable. We can declare a bind variable using VARIABLE command. Variable command declares the bind variable which you can refer in PL/SQL. Also as I said earlier in this tutorial that in order to declare bind variables we do not need to write any PL/SQL block or section.

Let’s do an example and declare our first bind variable

**VARIABLE v\_bind1 VARCHAR2 (10);**

See how easy it is to declare a bind variable in oracle database! You simply have to write a command which starts with keyword VARIABLE followed by the name of your bind variable which is completely user defined along with the data type and data width. That’s how we declare a bind variable in Oracle database.

Did you notice that I didn’t write any PL/SQL block or section here to declare this bind variable which is very unlike the user variable.

## ****Other Uses of Variable Command.****

Declaring the bind variable is the first use of this variable command there are few other uses of it also. Let’s see what those are:

### ****List all the bind variables declared in the session.****

Yes using Variable command you can display the list of all the bind variables you have declared in the session. To display the list of all the bind variables you simply have to write the keyword variable and execute. Doing so will return list of all the bind variables.

Let’s see.

**VARIABLE;**

Execute the above command and that will show you the list of all the bind variables that you have declared in your session.

### ****See the definition of bind variable.****

Variable command can also show you the definition of any bind variable created in the session. By definition I mean the data type and data width of the variable. To see the definition of the bind variable you have to write the keyword VARIABLE followed by the name of the bind variable in question.

Let’s do an example and see the definition of this bind variable v\_Bind2.

**Variable v\_bind2;**

Execution of above command will show you the definition of bind variable RebellionRider.

**Restriction:**  
**﻿If you are creating a bind variable of NUMBER datatype then you can not specify the precision and scale.**

## ****Initialize the Bind Variable****

As we have now declared the bind variable next we have to initialize it. We have several different ways of initializing the bind variable. Let’s see what those are.

### ****You can initialize the bind variable using “Execute” command.****

Execute command is like a wrapper which works as an execution section of PL/SQL block. Let’s see how it works.  Let’s initialize our bind variable v\_bind1 with a string RebellionRider.

**Exec :v\_bind1   := ‘Rebellion Rider’;**

This statement starts with keyword Exec which is the starting 4 alphabets of Keyword Execute.  You can either write whole keyword Execute or just the starting 4 alphabets “Exec” both will work fine. This is followed by the name of our bind variable which is v\_bind1. After that we have assignment operator followed by the string Rebellion Rider, as it’s a string thus it’s enclosed in single quotes.

That’s the first way of initializing the bind variable. The second way is:

### ****Initialize the bind variable by explicitly writing execution section of PL/SQL block.****

If you do not like shortcuts and are willing to do some hard work of writing a few extra lines of code then this is for you.

**SET SERVEROUTPUT ON;****BEGIN****:v\_bind1 := ‘Manish Sharma’;****END;****/**

This is a simple execution block where I initialized the bind variable v\_bind1 with the string Manish Sharma.

That is how we initialize the bind variable in Oracle Database or in PL/SQL.

## ****Referencing the Bind Variable****

Manish why did you put the colon sign (:) before the name of bind variable (:v\_bind1) while initializing it? Glad you asked.

Unlike user variables which you can access simply by writing their name in your code, you use colon before the name of bind variable to access them or in other words you can reference bind variable in PL/SQL by using a colon (:) followed immediately by the name of the variable as I did in the previous section.

## ****Display The Bind variable.****

There are 3 ways of displaying the value held by bind variable in PL/SQL or say in Oracle Database.

1. Using DBMS OUTPUT package.
2. Using Print command
3. Setting Auto print parameter on

Let’s check out each of them one by one.

### ****Using DBMS\_OUTPUT Package****

This is the simplest way of displaying the value held by any variable in PL/SQL. To display the value held by your bind variable you simply have to pass the name of your bind variable as a parameter to the PUT\_LINE procedure of DBMS\_OUTPUT package. Let’s see an example

**BEGIN****:v\_bind1 := ‘RebellionRider’;****DBMS\_OUTPUT.PUT\_LINE(:v\_bind1);****END;****/**

Things you must take care of here are:

1. PUT\_LINE is an executable statement which will require the execution section of PL/SQL block for its execution. In simple words you can only execute this statement in execution section of PL/SQL block otherwise you will get an error.
2. To see the output returned from this statement you have set the serveroutput on. You can do that by simply writing and executing

**SET SERVEROUTPUT ON;**

I highly recommend you to read my PL/SQL [Tutorial 1](http://www.rebellionrider.com/block-types-in-pl-sql-by-rebellionrider-manish-sharma/) to understand Blocks & Section in PL/SQL you can also watch my [video tutorial](https://youtu.be/rbarR4_gaH8) on the same.

### ****Using PRINT command****

Like DBMS\_OUTPUT statement print command also displays the current value of the bind variable except that you can write this command in host environment rather than inside any PL/SQL block or section. Yes similar to variable command, print command does not require any PL/SQL block for execution.

Suppose you want to see the current value of bind variable v\_bind1 for that simply write the print command in your SQL\*PLUS

**Print :v\_bind1;****Or****Print v\_bind1;**

Writing keyword PRINT without any argument will display you the current values of all the bind variables with their names in the session.

### ****Setting Auto print parameter on****

The last way of displaying the current value of a bind variable is by setting a session based parameter AUTOPRINT on. Doing so will display you the value of all the bind variables without the use of any specific commands such as Print or DBMS\_OUTPUT which we just saw.

To set this parameter you simply have to write

**SET AUTOPRINT ON;**

And this command will set AutoPrint parameter on for the session which will automatically print the values of bind variable.

# What Are Conditional Control Statements In PL/SQL

## ****An Introduction To Conditional Control Statements In PL/SQL****

Conditional control statements are those which allow you to control the execution flow of the program depending on a condition. In other words the statements in the program are not necessarily executed in a sequence rather one or other group of statements are executed depending on the evaluation of a condition.

## ****Types Of Conditional Control Statement in PL/SQL****

In Oracle PL/SQL we have two types of conditional control statements which are

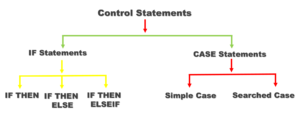
1. IF statements and
2. CASE statements

Both these statements can be further divided into different forms. For example IF statements has 3 different forms

1. IF THEN
2. [IF THEN ELSE](http://www.rebellionrider.com/if-then-else-conditional-control-statement-in-pl-sql/)
3. [IF THEN ELSEIF](http://www.rebellionrider.com/if-then-elsif-conditional-control-statement-in-pl-sql/)

And CASE statement has 2 different forms such as

1. SIMPLE CASE and
2. SEARCHED CASE



 Simple IF-THEN Conditional Control Statements In PL/SQL

IF-THEN is the most basic kind of conditional statements in PL/SQL that enables you to specify only a single group of action to be taken. You can also say that this specific group of action is taken only when a condition is evaluated to be true.

Correct me, if I am wrong, I think IF conditional statement is used in almost every programming language. If you disagree then tweet me @RebellionRider and tell me which programming languages do not support IF statements.

### ****IF-THEN Structure****

Before jumping into the tutorial let’s quickly understand the structure/syntax of the IF-THEN statement.

**IF condition THEN****Statement1;****…****Statement N;****END IF;**

In the starting we have the keyword IF which marks the beginning of IF-THEN block followed by a valid condition or a valid expression which will get evaluated followed by another keyword THEN. Similarly the reserved phrase END IF marks the ending of IF-THEN block.  In between we have a sequence of executable statements which will get executed only if the condition evaluated to be true otherwise the whole IF-THEN block will be skipped.

### ****Working****

When an IF-THEN statement is executed a condition is evaluated to either True or False. If the condition evaluates to true, control is passed to the first executable statement of the IF-THEN construct. If the condition evaluates to false, control is passed to the first executable statement after the END-IF statement.

#### **Examples:**

**SET SERVEROUTPUT ON;****DECLARE****v\_num NUMBER := 9;****BEGIN****IF v\_num < 10 THEN****DBMS\_OUTPUT.PUT\_LINE(‘Inside The IF’);****END IF;****DBMS\_OUTPUT.PUT\_LINE(‘outside The IF’);****END;****/**

This is a very simple PL/SQL anonymous block. In the declaration section I have declared a variable v\_num with data type NUMBER and initialized it with integer 9.

Let’s come to our execution section. Here as you can see we have 2 DBMS\_OUTPUT statements one is inside the IF-THEN block and another is outside it.

The first DBMS\_OUTPUT statement will execute only if the condition of our IF-THEN block is evaluated as true otherwise it will be skipped but the 2nd DBMS\_OUTPUT statement which is outside the IF-THEN block will execute every time you execute this PL/SQL block.

This means that if the condition is true then both the string INSDIE THE IF and OUTSIDE THE IF will be printed otherwise only OUTSIDE THE IF will be printed.

**DECLARE****v\_website VARCHAR2(30) := ‘RebellionRider.com’;****v\_author  VARCHAR2(30) := ‘Manish’;****BEGIN****IF v\_website =’RebellionRider.com’ AND  v\_author= ‘Manish’ THEN****DBMS\_OUTPUT.PUT\_LINE(‘Everything is Awesome :)’);****END IF;****DBMS\_OUTPUT.PUT\_LINE(‘Give this Video a Thumbs Up’);****END;****/**

This one is slightly different than the previous example. Here we used logical AND operator in the condition. If this condition is evaluated to be true then both strings from both DBMS\_OUTPUT statements will be printed otherwise only the string from 2nd DBMS\_OUTPUT statement will be displayed back to you.

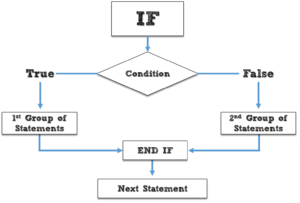
This example is for showing how you can check multiple conditions in a single go using logical operator. You can even use logical OR instead of logical AND operator.

# IF-THEN-ELSE Conditional Control Statement In PL/SQL

The previous tutorial was all about IF-THEN statement in Oracle PL/SQL. There we learnt that a simple IF-THEN statement enables us to specify the sequence of statements to be executed only if the condition is evaluated to be true. In case this condition is evaluated to be false then no special action is to be taken except to proceed with execution of the program.

To overcome this drawback in oracle PL/SQL we have IF-THEN-ELSE statement widely pronounced as IF-ELSE statement.

With IF-THEN-ELSE in PL/SQL we have two groups of executable statements, one which gets executed if the condition is evaluated to be true and another group gets executed if the condition is evaluated to be false. Once the IF-THEN-ELSE construct gets completed the next statement right after IF-THEN-ELSE block is executed.



## ****Syntax****

**IF condition THEN****Statement 1;****ELSE****Statement 2;****END IF;****Statement 3**

And here is the syntax as you can see IF-THEN-ELSE construct starts with the keyword IF and ends with the reserved phrase END IF. Followed by IF keyword we have to specify a valid condition which will get evaluated. If this condition is evaluated to be TRUE then control is passed to the statement 1, and if this condition is evaluated to be false then control will jump over to statement 2. Once the construct is completed then statement 3 is executed.

## ****Example****

Here in this example we will take a numeric input from the user and will check whether a user has entered an even number or an odd number using IF THEN ELSE statement. This is going to be a very easy example.

**SET SERVEROUTPUT ON;****DECLARE****v\_num    NUMBER := &enter\_a\_number;****BEGIN****IF MOD (v\_num, 2) = 0 THEN****DBMS\_OUTPUT.PUT\_LINE (v\_num || ‘ Is Even’);****ELSE****DBMS\_OUTPUT.PUT\_LINE (v\_num ||’ is odd’);****END IF;****DBMS\_OUTPUT.PUT\_LINE (‘IF THEN ELSE Construct complete ‘);****END;**

In the declaration section of this block I have declared a variable v\_num with data type NUMBER and using substitution operator I am taking values from the user.

Next in the execution section of the block we have our IF THEN ELSE construct where we are checking whether the number entered by the user is even or odd. For this test I have used MOD function of Oracle library as our IF condition which will divide the first parameter by the second parameter and return the remainder or say Modulus. If this modulus is zero then number will be even otherwise number will be odd. Thus if the modulus comes as zero then control will jump over first DBMS\_OUTPUT statement and will print the given string but if modulus is not zero then control will jump over the 2nd DBMS\_OUTPUT statement and will display its string. Once this is complete then control will come out from the IF THEN ELSE construct and will print the third DBMS\_OUTPUT statement.

# IF-THEN-ELSIF Conditional Control Statement In PL/SQL

In the previous tutorial we saw that IF ELSE condition gives us the provision of executing statements not only when the condition is evaluated to be true but also when the condition is evaluated to be false. But using IF ELSE statement we can only check one condition at a time, there is no provision for checking multiple conditions. This becomes its major drawback.

To overcome this we have IF THEN ELSIF condition in Oracle PL/SQL. Using this statement you can check multiple conditions unlike the IF conditions discussed in the previous tutorials.

## ****Syntax****

**IF CONDITION 1 THENSTATEMENT 1;ELSIF CONDITION 2 THENSTATEMENT 2;ELSIF CONDITION 3 THENSTATEMENT 3;…ELSESTATEMENT N; END IF;**

Similar to the other IF conditions, keyword IF marks the beginning and reserved phrase END IF marks the ending of the block. Make sure to put a white space in between END and IF of ending phrase. In this ELSIF construct we have multiple conditions. CONDITION 1 through CONDITION N are a sequence of conditions that have to be evaluated for TRUE or FALSE. These conditions are mutually exclusive. This means that if condition 1 is evaluated to be TRUE then statement 1 is executed and control will jump over the first executable statement outside the ELSIF construct and rest of the conditions will be ignored. If condition 1 is evaluated to be false then the compiler will jump inside and check the rest ELSIF conditions to look for the one which is true. If it finds any then it will execute the corresponding statements otherwise it will run the else statement.

In simple words the **IF** **THEN** **ELSIF** statement is responsible for running the first **statement** for which the **condition** is true. Once this is done the rest of the conditions are not evaluated. In case none of the **conditions** are true, then the **else\_statements** run provided that they exist; otherwise no action is taken by the **IF** **THEN** **ELSIF** statement.

## ****Example****

**DECLARE****v\_Place VARCHAR2(30) := ‘&Enter Place’;****BEGIN****IF v\_Place = ‘Metropolis’ THEN****DBMS\_OUTPUT.PUT\_LINE(‘This City Is Protected By Superman’);****ELSIF v\_Place = ‘Gotham’ THEN****DBMS\_OUTPUT.PUT\_LINE(‘This City is Protected By Batman’);****ELSIF v\_Place = ‘Amazon’ THEN****DBMS\_OUTPUT.PUT\_LINE(‘This City is protected by Wonder Woman’);****ELSE****DBMS\_OUTPUT.PUT\_LINE(‘Please Call Avengers’);****END IF;****DBMS\_OUTPUT.PUT\_LINE(‘Thanks For Contacting us’);END;**

This is a very simple Example where I have declared a variable v\_place which has data type varchar2 and we are taking input in this variable from the user. As you can see here I have used substitution operator (&). Don’t forget to enclose this input string along with ampersand operator (or substitution operator &) inside single quotes as variable is of varchar2 data type.

In the execution section we have ELSIF construct. Where we have one IF condition one else condition and 2 ELSIF conditions. Every condition is accompanied with a DBMS\_OUTPUT statement which will get executed if the respective condition is evaluated to be true. Otherwise the DBMS\_OUTPUT statement in ELSE section will run. Once this block is executed then control will come out and execute the first executable statement outside the ELSIF construct which is our last DBMS\_OUTPUT statement.

# What Are Loops & Simple Loops In PL/SQL

## ****Introduction To Loops & Simple Loops In PL/SQL****

With the previous tutorial we finished our Conditional Control Statement Series and learnt the concept of different types of IF conditions in Oracle PL/SQL. That was a very important topic from the examination’s perspective. Another topic which is again very important for your exam and could help you in getting good percentage is Iterative Statements.

Iterative statements famously known as Loops in Programming language. It executes block of statements or a part of a program several times.

### ****Types of Loops in Oracle PL/SQL****

There are 4 types of Loops in Oracle PL/SQL

1. Simple Loop
2. While Loop
3. Numeric For Loop and
4. Cursor For loop

In this series we will focus on the first 3 types of loops. The last type which is “Cursor For Loop” will be discussed with Cursor in the future Tutorial. Having said that let’s start today’s tutorial with Simple Loop.

### ****Simple Loop****

Simple loop is the most basic loop in Oracle PL/SQL

### Syntax

**LOOPStatement 1;Statement 2;…Statement 3;END LOOP;**

Here keyword LOOP marks the beginning and phrase END LOOP marks the ending of the loop. In between we have a sequence of executable statements.

As you can see in this syntax that unlike conventional loops here we do not have update statements or for that matter exist conditions which will terminate the loop. May be that is why we call this a simple loop.

### ****Example 1****

**DECLARE****v\_counter   NUMBER :=0;****v\_result  NUMBER;****BEGIN****LOOP****v\_counter := v\_counter+1;****v\_result := 19\*v\_counter;****DBMS\_OUTPUT.PUT\_LINE(’19’||’ x ‘||v\_counter||’ = ‘|| v\_result);****END LOOP;****END;**

Here in this example as you can see we do not have any exit statement to terminate the loop. This means that if we execute this program then the execution will keep on printing till we halt it manually.

In this case Oracle PL/SQL gives us two clauses to terminate the loop

1. Exit
2. Exit When

Exit clause will terminate the loop when Exit condition is evaluated to be true. The exit condition is evaluated with the help of Simple IF THEN condition which we discussed in PL/SQL Tutorial 8. So let’s see how you can use this exit statement in this example.

### ****Example 2 Terminate Loop with EXIT****

**DECLARE****v\_counter   NUMBER :=0;****v\_result  NUMBER;****BEGIN****LOOP****v\_counter := v\_counter+1;****v\_result := 19\*v\_counter;****DBMS\_OUTPUT.PUT\_LINE(’19’||’ x ‘||v\_counter||’ = ‘|| v\_result);****IF v\_counter >=10 THEN****EXIT;****END IF;****END LOOP;****END;**

You simply have to add this IF THEN block either right above the phrase END LOOP or immediately below the keyword loop. What this IF THEN block will do? This block will keep an eye on your counter and will tell the control to exit the loop when counter either becomes greater than or equal to 10. Which means loop will execute for 10 times.

### ****Example 3 Terminate the Loop with EXIT WHEN Clause****

Second way of terminating the loop is by using EXIT WHEN clause. Using this clause you can replace this whole IF THEN block with a simple single statement.

**DECLARE****v\_counter   NUMBER :=0;****v\_result  NUMBER;****BEGIN****LOOP****v\_counter := v\_counter+1;****v\_result := 19\*v\_counter;****DBMS\_OUTPUT.PUT\_LINE(’19’||’ x ‘||v\_counter||’ = ‘|| v\_result);****EXIT WHEN i\_counter>=10;****END LOOP;****END;**

# What Is WHILE Loop In PL/SQL

## ****Iterative Statements/Loops In PL/SQL****

Concept of While Loop is not new to the programming world and almost all the programming languages support this concept. While loop, is not the exception to other loops. It also executes block of statements several times but this loop is best usable when number of iterations to be performed are unknown.

### Syntax

**WHILE condition LOOP****Statement 1;****Statemen 2;****…****Statement 3;****END LOOP;**

The keyword WHILE marks the beginning of the loop followed by word CONDITION which will serve your test condition. This will get evaluated either to be true or to be false and at the end of our first line we have another keyword which is LOOP. The statements 1 through N are sequence of executable statements which define the body of the loop. And at the end we have a reserved phrase END LOOP which indicates the ending of the while loop.

In order to execute the body of the loop the test condition needs to be true. If this condition is evaluated to be true then the control will jump inside the loop and execute whatever statements it has. This iteration will continue until the test condition becomes false. As soon as the test condition is evaluated to be false the control will come out of the loop and execute the statement which immediately follows the loop.

### ****Examples****

Example 1

**DECLARE****v\_counter  NUMBER :=1;****v\_result NUMBER ;****BEGIN****WHILE  v\_counter <= 10****LOOP****v\_result := 9  \*v\_counter;****DBMS\_OUTPUT.PUT\_LINE(‘9’||’ x ‘||v\_counter||’ = ‘||v\_result);****v\_counter  := v\_counter+1;****END LOOP;****DBMS\_OUTPUT.PUT\_LINE(‘out’);****END;****/**

In this example we have declared two variables – v\_counter which will serve as a counter and variable v\_result which will hold the result of multiplication.

Down in the execution section we have our while loop. The first statement inside the loop body is an arithmetic expression which will perform the multiplication of our table and will store the result in v\_result variable.

2nd Statement is an output statement which will print the result of the multiplication in a formatted manner.

And the third statement is an update statement which will update the counter with each iteration.

This while loop will keep on iterating until the counter is less than 10 or it becomes equal to 10. Once the value of the counter becomes 10 the while loop will terminate and execute the first statement immediately outside the loop body.

Example 2 Boolean Expression

**DECLARE****v\_test    BOOLEAN := TRUE;****v\_counter NUMBER  := 0;****BEGIN****WHILE v\_test LOOP****v\_counter := v\_counter+1;****DBMS\_OUTPUT.PUT\_LINE( v\_counter );****IF v\_counter = 10 THEN****v\_test    := FALSE;****END IF;****END LOOP;**  
**﻿DBMS\_OUTPUT.PUT\_LINE (‘This Statement is outside the loop and will always execute’);****END;**

With the Boolean expression in loop we have to write the code which will change its value to false and terminate the loop. Failing to do so can make your loop an infinity loop. In the above program the simple IF THEN block inside the loop body will change the value of the Boolean expression v\_test and set it on false when counter becomes equal to 10 this till terminate the loop and bring the control over the first statement immediately outside the loop body

# Numeric FOR Loop In PL/SQL

The simplicity and easy to use behavior of FOR loop has won the hearts of millions and has become the most widely used loop in programming. In PL/SQL we have two types of FOR loops:

1. Numeric FOR loop and
2. Cursor FOR loop.

FOR loop allows you to execute the block of statements repeatedly for a fixed number of time whereas WHILE loop is better suited when the number of iterations are unknown.

This tutorial will concentrate on Numeric “FOR LOOP”. We’ll leave the [Cursor FOR loop](http://www.rebellionrider.com/cursor-for-loop-with-simple-explicit-cursor-in-oracle-database/) for the future when we will learn the concepts of Cursor.

## ****Syntax****

FOR loop\_counter IN [REVERSE] lower limit.. upper\_limit LOOP

Statement 1;

Statement 2;

…

Statement 3;

END LOOP;

For the in-depth explanation of the above syntax please watch my [video](https://youtu.be/DfAmnj2j7WI). There I have explained the same in detail. Now let’s see some examples.

## ****Examples of Numeric FOR Loop In Oracle PL/SQL.****

### ****Example 1: FOR loop****

SET SERVEROUTPUT ON;

BEGIN

FOR v\_counter IN 1..10 LOOP

DBMS\_OUTPUT.PUT\_LINE(v\_counter);

END LOOP;

END;

Here we only have the execution section and inside that we have our FOR loop which will print the value of v\_counter variable from 1 to 10.

Have you noticed that we didn’t declare the v\_counter variable anywhere in the program? Even we don’t have the declaration section here in this code. This is because variable v\_counter is an implicit index integer variable which gets declared automatically with the definition of FOR loop. Moreover the variable v\_counter will increment by 1 with each iteration automatically by FOR loop construct thus you do not need to write update statement (v\_counter := v\_counter +1) explicitly. As a matter of fact if you will try to write the update statement in the “FOR loop” then you will get an error.

### ****Example 2: FOR Loop with IN REVERSE keyword.****

Now suppose you want to print the counting, same as we did in the previous example but this time in reverse order. To do so you don’t have to change the loop definition or even you don’t have to add any extra line of codes, PL/SQL block will be same as of the previous example. You just have to add one keyword REVERSE immediately after IN keyword in FOR LOOP definition. Rest of the code will remain the same as that of the previous example.

BEGIN

FOR v\_counter IN REVERSE 1..10 LOOP

DBMS\_OUTPUT.PUT\_LINE(v\_counter);

END LOOP;

END;

/

This code will give you counting from 1 to 10 in reverse order on execution.

### ****Example 3: Multiplication Table using Numeric FOR loop****

DECLARE

v\_result NUMBER;

BEGIN

FOR v\_counter IN 1..10 LOOP

v\_result:= 19\*v\_counter;

DBMS\_OUTPUT.PUT\_LINE(v\_result);

END LOOP;

END;

/

In this example we need one extra variable to store the result of the multiplication thus we declared a variable v\_result with NUMBER data type. In the execution section we have our “FOR loop” and this time inside the loop we have only two statements. First is an arithmetic expression which will perform the multiplication of our table and will store the result in v\_result variable. Second is the output statement which will display you the result in a formatted manner.

The Insider’s Guide to Database Triggers

## ****Definition of Database Triggers****

Triggers are named PL/SQL blocks which are stored in the database.  We can also say that they are specialized stored programs which execute implicitly when a triggering event occurs. This means we cannot call and execute them directly instead they only get triggered by events in the database.

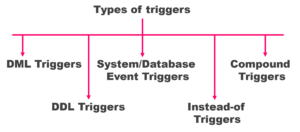
### ****Events Which Fires the Database Triggers****

These events can be anything such as

1. **A DML Statement**– An Update, Insert or Delete statement executing on any table of your database. You can program your trigger to execute either BEFORE or AFTER executing your DML statement. For example, you can create a trigger which will get fired Before the Update. Similarly, you can create a trigger which will get triggered after the execution of your INSERT DML statement.
2. **A DDL Statement** – Next type of triggering statement can be a DDL Statement such as CREATE or ALTER. These triggers can also be executed either BEFORE or AFTER the execution of your DDL statement. These triggers are generally used by DBAs for auditing purposes. And they really come in handy when you want to keep an eye on the various changes on your schema. For instance, who created the object or which user. Just like some cool spy tricks.
3. **A system event**. – Yes, you can create a trigger on a system event. And by a system event, I mean shut down or startup of your database.
4. **A User Events** – Another type of triggering event can be User Events such as log off or log on onto your database. You can create a trigger which will either execute before or after the event. Furthermore, it will record the information such as time of event occur, the username who created it.

## ****Types of Database Triggers****

There are 5 types of triggers in the Oracle database. 3 of them are based on the triggering event which are discussed in the previous section.



#### **Data Manipulation Language Triggers or DML triggers**

As the name suggests these are the triggers which depend on DML statements such as Update, Insert or Delete. They get fired either before or after them. Using DML trigger you can control the behavior of your DML statements. You can audit, check, replace or save values before they are changed. Automatic Increment of your Numeric primary key is one of the most frequent tasks of these types of triggers.

#### **Data Definition Language Triggers or DDL triggers.**

Again as the name suggests these are the type of triggers which are created over DDL statements such as CREATE or ALTER. They get fired either before or after the execution of your DDL statements. Using this type of trigger you can monitor the behavior and force rules on your DDL statements.

#### **System or Database Event triggers.**

Third type of triggers is system or database triggers. These are the type of triggers which come into action when some system event occurs such as database log on or log off. You can use these triggers for auditing purposes. For example, keeping an eye on information of system access like say who connects with your database and when. Most of the time System or Database Event triggers work as Swiss Knife for DBAs and help them in increasing the security of the data.

#### **Instead-of Trigger**

This is a type of trigger which enables you to stop and redirect the performance of a DML statement. Often this type of trigger helps you in managing the way you write to non-updatable views. You can also see the application of business rules by INSTEAD OF triggers where they insert, update or delete rows directly in tables that are defining updatable views.  Alternatively, sometimes the INSTEAD OF triggers are also seen inserting, updating or deleting rows in designated tables that are otherwise unrelated to the view.

#### **Compound triggers**

These are multi-tasking triggers that act as both statement as well as row-level triggers when the data is inserted, updated or deleted from a table. You can capture information at four timing points using this trigger:

* before the firing statement;
* prior to the change of each row from the firing statement;
* post each row changes from the firing statement;
* after the firing statement.

All these types of triggers can be used to audit, check, save and replace the values. Even before they are changed right when there is a need to take action at the statement as well as at row event levels.

**The Syntax Of Database Trigger**

|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8  9  10  11  12  13 | CREATE [OR REPLACE] TRIGGER Ttrigger\_name  {BEFORE|AFTER} Triggering\_event ON table\_name  [FOR EACH ROW]  [FOLLOWS another\_trigger\_name]  [ENABLE/DISABLE]  [WHEN condition]  DECLARE    declaration statements  BEGIN    executable statements  EXCEPTION    exception-handling statements  END; |

For the detailed explanation of the syntax, I would suggest you watch the [video tutorial](https://youtu.be/R3fvX_xf5P4). There I have explained each and every clause of the syntax in detail.

## ****Uses of Database triggers.****

1. Using database triggers we can enforce business rules that can’t be defined by using integrity constants.
2. Using triggers we can gain strong control over the security.
3. We can also collect statistical information on the table access.
4. We can automatically generate values for derived columns such as auto increment numeric primary key.
5. Using database triggers we can prevent the invalid transactions.

## ****Restriction on The Database Triggers****

1. The maximum size of the database trigger body must not exceed 32,760 bytes. This is because triggers’ bodies are stored in LONG datatypes columns.
2. A trigger may not issue transaction control statements or TCL statements such as COMMIT, ROLLBACK or SAVEPOINT. All operations performed when the trigger fires, become part of a transaction. Therefore whenever this transaction is rolled back or committed it leads to the respective rolling back or committing of the operations performed.
3. Any function or procedure called by a database trigger may not issue a transactional control statement. That is unless it contains an autonomous transaction.
4. Declaring LONG or LONG RAW variable is not permissible in the body of the trigger.

# What Are DML Triggers In PL/SQL

## ****Data Manipulation Language (DML) Triggers.****

As the name suggests these are the triggers which execute on DML events or say depend on DML statements such as Update, Insert or Delete. Using DML trigger you can control the behavior of your DML statements.

Since the theory has already been discussed in the previous tutorial hence I won’t bore you further. You can refer to the previous tutorial “Introduction to Triggers” anytime.

### ****Examples****

In order to demonstrate the creation process of DML trigger we need to first create a table.

|  |  |
| --- | --- |
| 1  2  3 | CREATE TABLE superheroes (    sh\_name VARCHAR2 (15)  ); |

I have created this table with the name SUPERHEROES which has only one column sh\_name with data type varchar2 and data width 15.  Now I will write a DML trigger which will work on this table.

So the table is created. Now let’s do some examples which will help you in understanding the concepts more clearly.

**Before that a simple tip:  
Always remember to set your server output ON otherwise the output message returned from your trigger will not be displayed back to you.**

|  |  |
| --- | --- |
| 1 | SET SERVEROUTPUT ON; |

### ****Example 1. Before Insert Trigger****

In the first example we will see how to create a trigger over Insert DML. This trigger will print a user defined message every time a user inserts a new row in the superheroes table.

|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8  9  10  11 | CREATE OR REPLACE TRIGGER bi\_Superheroes  BEFORE INSERT ON superheroes  FOR EACH ROW  ENABLE  DECLARE    v\_user  VARCHAR2 (15);  BEGIN   SELECT user INTO v\_user FROM dual;   DBMS\_OUTPUT.PUT\_LINE('You Just Inserted a Row Mr.'|| v\_user);  END;  / |

On successfully compiling, this trigger will show you a string along with the user name who performed the “Insert” DML on superheroes table. Thus you check this trigger by Inserting a row in Superheroes table.

|  |  |
| --- | --- |
| 1 | INSERT INTO superheroes VALUES ('Ironman'); |

### ****Example 2: Before Update Trigger.****

Update Trigger is the one which will execute either before or after Update DML. The creation process of an Update trigger is the same as that of Insert Trigger. You just have to replace Keyword INSERT with UPDATE in the 2nd Line of the above example.

|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8  9  10  11 | CREATE OR REPLACE TRIGGER bu\_Superheroes  BEFORE UPDATE ON superheroes  FOR EACH ROW  ENABLE  DECLARE    v\_user  VARCHAR2 (15);  BEGIN   SELECT user INTO v\_user FROM dual;   DBMS\_OUTPUT.PUT\_LINE('You Just Updated a Row Mr.'|| v\_user);  END;  / |

On successfully compiling, this trigger will print a user defined string with the username of the user who updated the row. You can check this trigger by writing an update DML on the superheroes table.

|  |  |
| --- | --- |
| 1 | UPDATE superheroes SET SH\_NAME = ‘Superman’ WHERE SH\_NAME='Ironman'; |

### ****Example 3: Before Delete Trigger****

Similar to Insert and Update DML you can write a trigger over Delete DML. This trigger will execute either before or after a user deletes a row from the underlying table.

|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8  9  10  11 | CREATE OR REPLACE TRIGGER bu\_Superheroes  BEFORE DELETE ON superheroes  FOR EACH ROW  ENABLE  DECLARE    v\_user  VARCHAR2 (15);  BEGIN   SELECT user INTO v\_user FROM dual;   DBMS\_OUTPUT.PUT\_LINE('You Just Deleted a Row Mr.'|| v\_user);  END;  / |

You can check the working of this trigger by executing a DELETE DML on underlying table which is superheroes.

|  |  |
| --- | --- |
| 1 | DELETE FROM superheroes WHERE sh\_name = 'Superman'; |

Above three examples showed you 3 different triggers for 3 different DML events on one table. Don’t you think that if we can cover all these 3 events in just 1 trigger then it will be a great relief? If you think so then my dear friend I have some good news for you. Let me show you how we can achieve this feat.

#### **INSERT, UPDATE, DELETE All in One DML Trigger Using IF-THEN-ELSIF**

|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8  9  10  11  12  13  14  15  16  17  18  19 | CREATE OR REPLACE TRIGGER tr\_superheroes  BEFORE INSERT OR DELETE OR UPDATE ON superheroes  FOR EACH ROW  ENABLE  DECLARE    v\_user  VARCHAR2(15);  BEGIN      SELECT      user INTO v\_user FROM dual;    IF INSERTING THEN      DBMS\_OUTPUT.PUT\_LINE('one line inserted by '||v\_user);    ELSIF DELETING THEN      DBMS\_OUTPUT.PUT\_LINE('one line Deleted by '||v\_user);    ELSIF UPDATING THEN      DBMS\_OUTPUT.PUT\_LINE('one line Updated by '||v\_user);    END IF;  END;  / |

Using this one trigger you can achieve the same results as that of the above three triggers. I have explained every single line of this trigger along with the other three triggers in detail in my Video tutorial. I highly suggest you to watch that tutorial.

# Table Auditing Using DML Triggers In Oracle PL/SQL

Whereas the [previous tutorial](http://www.rebellionrider.com/what-are-dml-triggers-in-pl-sql/) was more focused on how to create and use Before Insert, Update and Delete DML triggers with simple and easy to understand examples, today’s tutorial on the other hands will be slightly more complex as we will learn how to perform table auditing in Oracle Database using DML triggers.

## ****Table Auditing****

Table auditing means keeping a track of all the dml activities performed on a specific table of the database for example which user Inserted, updated or deleted a row from the table and when. It is like spying on the users who are messing your table’s data.

### ****Example****

For the demonstration we will use the table ‘Superheroes’ which we created in the previous tutorial.

Suppose you want to keep an eye on the users who are inserting, updating or deleting data from the ‘Superheroes’ table. Let’s see how we can achieve this. To do so we will need another table in which we can journal the auditing data entries.

|  |  |
| --- | --- |
| 1  2  3  4  5  6  7 | CREATE TABLE sh\_audit(    new\_name varchar2(30),    old\_name varchar2(30),    user\_name varchar2(30),    entry\_date varchar2(30),    operation  varchar2(30)  ); |

This table sh\_audit has 5 columns which will store the auditing information such as the new data inserted or updated, old data which is updated or deleted from the table, name of the user who did it along with the date and time, also the type of DML operation performed.

Next we will write a trigger on the source table superheroes and will store the data into the auditing table sh\_audit.

|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8  9  10  11  12  13  14  15  16  17  18  19  20  21 | CREATE OR REPLACE trigger superheroes\_audit  BEFORE INSERT OR DELETE OR UPDATE ON superheroes  FOR EACH ROW  ENABLE  DECLARE    v\_user varchar2 (30);    v\_date  varchar2(30);  BEGIN    SELECT user, TO\_CHAR(sysdate, 'DD/MON/YYYY HH24:MI:SS') INTO v\_user, v\_date  FROM dual;    IF INSERTING THEN      INSERT INTO sh\_audit (new\_name,old\_name, user\_name, entry\_date, operation)      VALUES(:NEW.SH\_NAME, Null , v\_user, v\_date, 'Insert');    ELSIF DELETING THEN      INSERT INTO sh\_audit (new\_name,old\_name, user\_name, entry\_date, operation)      VALUES(NULL,:OLD.SH\_NAME, v\_user, v\_date, 'Delete');    ELSIF UPDATING THEN      INSERT INTO sh\_audit (new\_name,old\_name, user\_name, entry\_date, operation)      VALUES(:NEW.SH\_NAME, :OLD.SH\_NAME, v\_user, v\_date,'Update');    END IF;  END;  / |

I would highly suggest you to watch the YouTube Video Tutorial on the same topic since there I have explained the working of this particular trigger line by line in detail.

On successful compilation this trigger will insert a row containing auditing data such as the data inserted, updated and deleted from the source table superheroes along with the username who tampered the data as well as the date and time when it was done and also the name of DML statement executed by user to tamper the data of your table.

## ****Pseudo Records (New/Old)****

If you will carefully see the Insert statements used in the IF-THEN-ELSIF statements in the above code, we used some Pseudo Records such as ‘:New’ or ‘:Old’ followed by the name of the column of our source table sh\_name.

These Psuedo Records helps us in fetching data from the sh\_name column of the underlying source table ‘Superheroes’ and storing it into the audit table sh\_audit.

Pseudo Record ‘: NEW’, allows you to access a row currently being processed. In other words, when a row is being inserted or updated into the superheroes table. Whereas Pseudo Record ‘: OLD’ allows you to access a row which is already being either Updated or Deleted from the superheroes table.

In order to fetch the data from the source table, you have to first write the proper Pseudo Record (New/Old) followed by dot (.) and the name of the column of the source table whose value you want to fetch. For example in our case we want to fetch the data from sh\_name column which belongs to our source table superheroes. Thus we will write “: New. sh\_name” for fetching the current value and to fetch the previously stored value we will write “: OLD. sh\_name”. Once the values are fetched the INSERT dml will store these values into the respective columns of the audit table.

## ****Restriction on Pseudo Record****

* For an INSERT trigger, OLD contain no values, and NEW contain the new values.
* For an UPDATE trigger, OLD contain the old values, and NEW contain the new values.
* For a DELETE trigger, OLD contain the old values, and NEW contain no values.



Once you execute and compile this trigger then you can take it on a test run by writing DML statements on the underlying source table ‘Superheroes’. For example you can try Inserting a row in superheroes table and then check the audit table whether there is some data or not.

|  |  |
| --- | --- |
| 1 | INSERT INTO superheroes VALUES ('Superman'); |

Similarly you can write Update and Delete DML statements on Superheroes table.

|  |  |
| --- | --- |
| 1  2  3 | UPDATE SUPERHEROES SET SH\_NAME = 'Ironman' WHERE SH\_NAME='Superman';  Or  DELETE FROM superheroes WHERE SH\_NAME = 'Ironman'; |

As soon as you execute any of these DML statements on the underlying table superheroes, the trigger will execute in the background and insert the audit data into the audit table sh\_audit.

# How To Create Synchronized Table Backup Using DML Trigger In Oracle PL/SQL

Recently we learnt [how to audit a table using DML Triggers in Oracle database](http://www.rebellionrider.com/table-auditing-using-dml-triggers-in-oracle-pl-sql/) now we will see how we can make a synchronized backup copy of a table using the same. By synchronized backup copy I mean the backup table gets automatically populated or updated with the main table simultaneously.

For the demonstration we will require two identical tables; one which will serve as your main table that will accept the data from your database user and the second which will be your backup table. I will use the Superheroes table which we have been using since the beginning of this DML trigger series as our main table.

|  |  |
| --- | --- |
| 1  2  3 | CREATE TABLE superheroes(      Sh\_name VARCHAR2(30)  ); |

Next we will have to create an identical table to this one which will work as our backup table.

Let’s create this backup table.

|  |  |
| --- | --- |
| 1 | CREATE TABLE superheroes\_backup AS SELECT \* FROM superheroes WHERE 1=2; |

The above command will create the identical table just like the main table superheroes only without data.

***Suggested Reading:***[***Create Table As Command.***](http://www.rebellionrider.com/copy-table-with-or-without-data-using-create-table-as-statement/)

Next we have to write the trigger which will insert, update or delete the rows from the backup table when someone does the same with our main table.

|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8  9  10  11  12  13  14  15 | CREATE or REPLACE trigger Sh\_Backup  BEFORE INSERT OR DELETE OR UPDATE ON superheroes  FOR EACH ROW  ENABLE  BEGIN    IF INSERTING THEN      INSERT INTO superheroes\_backup (SH\_NAME) VALUES (:NEW.SH\_NAME);    ELSIF DELETING THEN      DELETE FROM superheroes\_backup WHERE SH\_NAME =:old.sh\_name;    ELSIF UPDATING THEN      UPDATE superheroes\_backup      SET SH\_NAME =:new.sh\_name WHERE SH\_NAME =:old.sh\_name;    END IF;  END;  / |

After successful execution of the trigger, changes from the main table will get reflected on the backup table too. For detail explanation please watch the [video](https://youtu.be/jSv1LIlNKU0) on the same topic.

Before ending up this blog a quick disclaimer, though you can write this trigger for any table but I would not advise you to use such trigger on those tables that involve heavy data input, deletion and updation. This is chiefly due to the performance reasons.

# Schema & Database Auditing Using DDL Trigger In PL/SQL

DDL triggers are the triggers which are created over DDL statements such as CREATE, DROP or ALTER. Using this type of trigger you can monitor the behavior and force rules on your DDL statements.

For detailed theory on the subject I would suggest you to read Introduction of Triggers in Oracle database. In this tutorial we will concentrate on the particular part where we will learn how to create a DDL trigger for auditing the schema/user and the whole database.

In order to proceed ahead and start writing the trigger first we need a table in which we can journal the auditing information created by the trigger.

|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8 | CREATE TABLE schema\_audit    (      ddl\_date       DATE,      ddl\_user       VARCHAR2(15),      object\_created VARCHAR2(15),      object\_name    VARCHAR2(15),      ddl\_operation  VARCHAR2(15)    ); |

In case of schema/user auditing using DDL trigger creates this table in the same schema which you are auditing and in case of Database auditing using DDL trigger create this table in sys or system schema (sys or system both schemas can be used to perform database auditing).

## ****DDL Trigger for Schema Auditing****

First you need to log on to the database using the schema which you want to audit. For example suppose you want to create the DDL trigger to audit the HR schema then log on to your database using the HR schema.

Then Write, Execute and Compile the below trigger.

|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8  9  10  11 | CREATE OR REPLACE TRIGGER hr\_audit\_tr  AFTER DDL ON SCHEMA  BEGIN      INSERT INTO schema\_audit VALUES (  sysdate,  sys\_context('USERENV','CURRENT\_USER'),  ora\_dict\_obj\_type,  ora\_dict\_obj\_name,  ora\_sysevent);  END;  / |

If you will notice carefully the second line of the code (“AFTER DDL ON SCHEMA”) indicates that this trigger will work on the schema in which it is created. On successful compilation this trigger will insert the respective information such as the date when the DDL is executed, username who executed the DDL, type of database object created, name of the object given by the user at the time of its creation and the type of DDL into the table which we created earlier.

## ****DDL Trigger for Database Auditing.****

Similar to the schema auditing with some minor changes in the above trigger you can audit your database too. But for that first you need to logon to the database using either SYS user or SYSTEM user.

**Suggested reading:**[***How to connect to the database using SYS user.***](http://www.rebellionrider.com/how-to-make-new-database-connection-in-sql-developer-rebellionrider/)

After doing that you have to create the above shown table under the same user so that your trigger can dump the auditing data without any read and write errors.

|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8  9  10  11 | CREATE OR REPLACE TRIGGER db\_audit\_tr  AFTER DDL ON DATABASE  BEGIN      INSERT INTO schema\_audit VALUES (  sysdate,  sys\_context('USERENV','CURRENT\_USER'),  ora\_dict\_obj\_type,  ora\_dict\_obj\_name,  ora\_sysevent);  END;  / |

If you notice the second line of this code carefully then you will find that we have replaced the **keyword** **Schema** with the **keyword** **Database** which indicates that this trigger will work for the whole database and will perform the underlying work.

To create a trigger on database we require **ADMINISTER** **DATABASE** **TRIGGER** system privilege. All the administrative users such as sys or system already has these privileges by default that is the reason we created this database auditing DDL trigger using these users. Though you can create the same trigger with any user by granting the same privileges to them but that is not advisable because of your database security reasons.

# Schema Level Database LOGON Trigger In PL/SQL

Database event triggers also known as system event triggers come into action when some system event occurs such as database log on, log off, start up or shut down. These types of triggers are majorly used for monitoring activity of the system events and have been proved quite a powerful tool for a DBA.

## ****Types of Database Event Triggers.****

1. Schema Level Event Triggers
2. Database Level Event Triggers

Schema level event triggers can work on some specific schemas while the database event triggers have database wide scope. In other words database event triggers can be created to monitor the system event activities of either a specific user/schema or the whole database.

### ****Object/System Privileges****

Schema level event triggers can be created by any user of your database who has CREATE TRIGGER system privilege while the database event trigger can only be created by privileged user such as SYS or SYSTEM who has ‘Administrative Database Trigger’ System Privileges.

**Suggested Reading:**[**Introduction To Privileges In Oracle Database [Interview Edition]**](http://www.rebellionrider.com/user-privileges-in-oracle-database/)

## ****Syntax****

|  |  |
| --- | --- |
| 1  2  3  4  5  6 | CREATE OR REPLACE TRIGGER trigger\_name  BEFORE | AFTER database\_event ON database/schema  BEGIN      PL/SQL Code  END;  / |

Please watch the video tutorial for detailed explanation of the syntax.

### ****Example. Schema Level Event Trigger.****

Suppose user HR is a control freak and wants to monitor its every log on and log off activity. In this case what HR can do is, create event triggers on Log on and log off database event in its own schema.

#### **Step 1: Connect to the database**

Connect to the database using the user/schema in which you want to create the trigger. For the demonstration I will connect using my HR user.

|  |  |
| --- | --- |
| 1 | C:/> Conn hr/hr |

Or if you are using SQL Developer then read [here](http://www.rebellionrider.com/how-to-make-new-database-connection-in-sql-developer-rebellionrider/) on how to connect to the database using the same.

#### **Step 2: Create a Table**

Next you will need a table to store the logon and logoff data.

|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8 | CREATE TABLE hr\_evnt\_audit    (      event\_type VARCHAR2(30),      logon\_date DATE,      logon\_time VARCHAR2(15),      logof\_date DATE,      logof\_time VARCHAR2(15)    ); |

#### **Step3: Write the trigger Logon Schema Event Trigger.**

Now you are connected to the database using the desired user and also have the table ready to store the data. The only thing which is left is the trigger.

This trigger will fire every time HR user logs on to the database and respective values will be stored into the table which we just created in the step 2.

|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8  9  10  11  12  13 | CREATE OR REPLACE TRIGGER hr\_lgon\_audit  AFTER LOGON ON SCHEMA  BEGIN    INSERT INTO hr\_evnt\_audit VALUES(      ora\_sysevent,      sysdate,      TO\_CHAR(sysdate, 'hh24:mi:ss'),      NULL,      NULL    );    COMMIT;  END;  / |

That is how we create a LogOn event trigger in Oracle Database. You can watch the video tutorial on my YouTube channel on the same topic where I created and tested the trigger live. And Stay tuned as in the next tutorial we will see how to create a logoff event trigger in Oracle Database.

# What Is LogOff Database Event Trigger In Oracle PL/SQL

Welcome to the next tutorial of PL/SQL Triggers in Oracle Database. In this tutorial we will learn how to write:

1. Schema Level Logoff System Event Trigger and
2. Database Level Logoff System Event Trigger

For this tutorial, knowledge of System/Database Event trigger is required which we have already discussed in the previous tutorial.

Unlike logon database event trigger which fires **after** a valid logon attempt by a user on the database, logoff triggers execute **before** the user logs off from the database. Logoff trigger can be proved as a versatile tool for a DBA as well as for a database user.

## ****Schema Level Logoff System Event Trigger****

As I explained in the previous tutorial that a schema level trigger is one which worked only for a specific schema in which it is created or designed to work for. Any user of the database who has “**Create Trigger**” system privilege can design and create this trigger.

Example

Let’s write a trigger to audit the logoff

### ****Step 1: Logon to the database****

Logon to the database using any user such as HR, SH, OE or any other you want.

|  |  |
| --- | --- |
| 1 | C:\> SQLPLUS hr/hr |

**Read here:**[**How to connect to the database With Sample User (e.g. HR) using SQL Developer**](http://www.rebellionrider.com/how-to-make-new-database-connection-in-sql-developer-rebellionrider/)

### ****Step 2: Create a table.****

Create a table to dump the data generated by your schema level logoff trigger.

|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8 | CREATE TABLE hr\_evnt\_audit   (    event\_type VARCHAR2(30),    logon\_date DATE,    logon\_time VARCHAR2(15),    logof\_date DATE,    logof\_time VARCHAR2(15)   ); |

### ****Step 3: Write the trigger.****

Below written trigger will execute every time user HR logs off from the database.

|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8  9  10  11  12  13 | CREATE OR REPLACE TRIGGER log\_off\_audit  BEFORE LOGOFF ON SCHEMA  BEGIN    INSERT INTO hr\_evnt\_audit VALUES(      ora\_sysevent,      NULL,      NULL,      SYSDATE,      TO\_CHAR(sysdate, 'hh24:mi:ss')    );    COMMIT;  END;  / |

On successful compilation this trigger will fire every time the user, using which it was created, logs off from the database and that user in our case is HR.

As I said above that this trigger is bound to work only for the user in which it is created. What if we want to keep track of all the logoff activities of all the users of the database? In such a scenario we can write a database level system event trigger.

## ****Database Level System/Database Event Trigger.****

An often asked question in the interview is what are the differences between Schema Level and Database Level system Event triggers? There can be variety of answers for this question but major differences are as follows.

As the name suggests Database event trigger fires for the entire database or you can say that it fires for all the schemas created under the database in which these triggers are created which is unlike Schema Level System Event trigger which runs for a specific schema only.

Also database level system event trigger can only be created by high privileged users such as sys or system or any user who has **ADMINISTER DATABASE TRIGGER** system privilege where Schema level system event trigger can be created by any user of the database on its own schema which has Create Trigger System Privilege.

### ****Example: How To Create Database Level Logoff event Trigger In Oracle PL/SQL****

Creation process of Database Level Logoff Event Trigger is pretty similar to the trigger which we just saw except for a few minute changes.

### ****Step 1: Logon to the database****

As only the user with **ADMINISTER DATABASE TRIGGER** system privilege can create a database level event trigger thus we need to make sure that this time we should log on to the database using one of these users.

|  |  |
| --- | --- |
| 1 | C:\> SQLPLUS / as SYSDBA |

**Suggested Reading: Make Database Connection With SYS User In SQL Developer.**

### ****Step 2: Create a Table****

Again in order to store the audit data we need to create a table where this trigger will journal the entries of all the users. The structure of the table will be pretty similar to the above one except one extra column for storing the username with the help of which we can clearly identify the details and avoid the confusion.

|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8  9 | CREATE TABLE db\_evnt\_audit    (  User\_name VARCHAR2(15),      event\_type VARCHAR2(30),      logon\_date DATE,      logon\_time VARCHAR2(15),      logof\_date DATE,      logof\_time VARCHAR2(15)    ); |

### ****Step 3: Write the Database Level logoff system event trigger.****

Following Trigger will keep an eye on the logoff activity of the user of the database.

|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8  9  10  11  12  13  14 | CREATE OR REPLACE TRIGGER db\_lgof\_audit  BEFORE LOGOFF ON DATABASE  BEGIN    INSERT INTO db\_evnt\_audit    VALUES(      user,      ora\_sysevent,      NULL,      NULL,      SYSDATE,      TO\_CHAR(sysdate, 'hh24:mi:ss')      );  END;  / |

Code is very similar to the previous one except that this time this trigger will execute for all the users of the database. This we are making sure by using the keyword DATABASE instead of SCHEMA in the second line of the trigger unlike the previous trigger.

There can be various ways of taking advantage of these triggers; it depends on your creativity. All the above examples are meant to teach you the proper way of creating a System/Database Event Trigger.

# Startup & Shutdown Database Event Triggers In Oracle PL/SQL

In the series of Database Event Triggers so far we have learned how to create and work with database event Logon and Logoff triggers. Now the only options which are left are Database event Startup and Shutdown Triggers. Today in this blog we will learn how to create these triggers with some easy to understand examples.

## ****Startup Trigger.****

Startup triggers execute during the startup process of the database. In order to create a database event trigger for shutdown and startup events we either need to logon to the database as a user with DBA privileges such as sys or we must possess the ADMINISTER DATABASE TRIGGER system privilege.

**Suggested Reading: System Privileges**

### ****Example****

#### **Step1: Logon to the database**

In order to create a trigger on Startup Database Event first we will have to logon to our database using the user SYS with DBA privileges.

#### **Step 2: Create a Table**

To store the data generated by the execution of trigger we will require a table.

|  |  |
| --- | --- |
| 1  2  3  4  5  6 | CREATE TABLE startup\_audit  (    Event\_type  VARCHAR2(15),    event\_date  DATE,    event\_time  VARCHAR2(15)  ); |

#### **Step 3: Create the database Event Startup Trigger**

In this step we will create a trigger which will execute every time the database in which it is created starts up.

|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8  9  10  11 | CREATE OR REPLACE TRIGGER startup\_audit  AFTER STARTUP ON DATABASE  BEGIN    INSERT INTO startup\_audit VALUES  (      ora\_sysevent,      SYSDATE,      TO\_CHAR(sysdate, 'hh24:mm:ss')    );  END;  / |

On successful execution this trigger will insert a row of data each time database starts up.

## ****Shutdown Triggers****

SHUTDOWN triggers execute before database shutdown processing is performed. Similar to the startup trigger, only a user with DBA role or ADMINISTER DATABASE TRIGGER system privilege can create a shutdown trigger.

### ****Example.****

First 2 steps of creating a database event shutdown triggers are same as that of the startup trigger which we saw above.

|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8  9  10 | CREATE OR REPLACE TRIGGER tr\_shutdown\_audit  BEFORE SHUTDOWN ON DATABASE  BEGIN    INSERT INTO startup\_audit VALUES(      ora\_sysevent,      SYSDATE,      TO\_CHAR(sysdate, 'hh24:mm:ss')    );  END;  / |

Table used in this trigger is the same one which we created during the coding of the Startup trigger above.

SHUTDOWN triggers execute only when the database is shut down using NORMAL or IMMEDIATE mode. They do not execute when the database is shut down using ABORT mode or when the database crashes.

You can also use shutdown database event triggers for gathering your database system statistics. Here is an example

|  |  |
| --- | --- |
| 1  2  3  4  5  6 | CREATE OR REPLACE TRIGGER before\_shutdown  BEFORE SHUTDOWN ON DATABASE  BEGIN   gather\_system\_stats;  END;  / |

# How To Create Instead-Of Insert Trigger In Oracle PL/SQL

So far we have learnt DML, DDL and System Event Triggers in Oracle Database. Today in this tutorial we will explore the concepts of Instead-of trigger in Oracle Database. This blog has been written keeping in mind the certification exam as well as the job interview. Hope you will enjoy reading it. If so then click here to tweet and share.

## ****Instead Of Trigger****

Instead-of triggers in oracle database provide a way of modifying views that cannot be modified directly through the DML statements. By using Instead-of triggers, you can perform Insert, Update, Delete and Merge operations on a view in oracle database.

## ****Restriction on Instead-of View.****

Instead-of triggers can control Insert, Delete, Update and Merge operations of the **View**, **not the table**. Yes you heard it right, you can write an instead-of trigger on Views only and not on tables in Oracle database. That is the restriction that you have to comply with. Along with this you even have to comply with every general restriction that is imposed on all types of triggers. We have discussed those in the Introduction of triggers section.

## ****Uses of Instead-of trigger.****

Since an Instead-of trigger can only be used with views therefore we can use them to make a non-updatable view updatable as well as to override the default behavior of views that are updatable.

## ****What are Modifiable and Non Modifiable Views?****

A view is **naturally** **modifiable** if you do not require INSTEAD OF triggers to insert, delete or update data as well as if it complies to the restrictions discussed herewith. If the view query comprises of any of the mentioned constructs, then it is not naturally modifiable and therefore you cannot perform inserts, updates, or deletes on the view:

* Set operators
* Aggregate functions
* GROUP BY, CONNECT BY, or START WITH clauses
* The DISTINCT operator
* Joins (however, some join views are updatable)

In case a view consists of pseudo columns or expressions, then it is only possible to update it with an UPDATE statement and that also when it does not refer to any such pseudo columns or for that matter, expressions.

## ****Syntax of Instead-Of Trigger****

|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8 | CREATE [OR REPLACE] TRIGGER trigger\_name  INSTEAD OF operation  ON view\_name  FOR EACH ROW  BEGIN      ---Your SQL Code—  END;  / |

For line-by-line detailed explanation of the above syntax please refer my video tutorial. There I have explained all the clause and keyword of the syntax in detail.

### ****So you asked…****

When does an Instead-of trigger fires – Before or After the triggering event?

If you noticed carefully then you’ll see that we do not have either BEFORE or AFTER clause in the syntax. This is because unlike other triggers, instead-of trigger executes neither BEFORE nor AFTER but instead of a triggering event. That is why we do not need to specify either BEFORE or AFTER clause.

## ****Examples****

Instead-of Insert Trigger

Instead-of trigger can be best demonstrated using a View joining two or more tables. Thus in this example I will create two simple tables and will then create a view over them. After that I will create an Instead of trigger for Insert operation on this view.

### ****Step1: Create Tables****

Table 1- trainer

|  |  |
| --- | --- |
| 1  2  3  4 | CREATE TABLE trainer    (      full\_name VARCHAR2(20)    ); |

Table 2- Subject

|  |  |
| --- | --- |
| 1  2  3  4 | CREATE TABLE subject    (      subject\_name VARCHAR2(15)    ); |

Insert dummy data into the above tables

|  |  |
| --- | --- |
| 1  2 | INSERT INTO trainer VALUES ('Manish Sharma');  INSERT INTO subject VALUES ('Oracle'); |

### ****Step 2: Create View****

In this step I will create a view which will show you the combined result of the data from the two tables above.

**Suggested Reading:**[**How To Create SQL View in Oracle Database**](http://www.rebellionrider.com/what-is-sql-view-in-oracle-database/)

|  |  |
| --- | --- |
| 1  2 | CREATE VIEW vw\_rebellionrider AS  SELECT full\_name, subject\_name FROM trainer, subject; |

This is a non-updatable view which you can confirm by executing any DML statement over it. Error as a result of DML operation on this view will be your confirmation.

### ****Step 3: Create Trigger****

Next I will create an Instead-of Insert trigger over the view vw\_rebellionrider that we created in step 2.

|  |  |
| --- | --- |
| 1  2  3  4  5  6  7 | CREATE OR REPLACE TRIGGER tr\_Io\_Insert  INSTEAD OF INSERT ON vw\_rebellionrider  FOR EACH ROW  BEGIN    INSERT INTO trainer (full\_name) VALUES (:new.full\_name);    INSERT INTO subject (subject\_name) VALUES (:new.subject\_name);  END |

On successful execution, this trigger will insert a new row of data into both the underlying tables of the view vw\_rebellionrider. You can confirm that by executing an insert DML over the view.

# How To Create Instead-Of Update Trigger In Oracle PL/SQL

Similar to Instead-of Insert trigger, which we discussed in the previous tutorial, Instead-of Update trigger overrides default behavior of Update DML operation executed on the view. Execution of Update DML on a complex view is restricted because of the involvement of multiple tables over which your view is created. To override this restriction we can take the help of Instead-Of Update trigger.

## ****Instead-Of Update Trigger****

Instead-of update trigger will override the default behavior of your update operation when you execute the update statement and will let you update the data of the underlying tables over which your view is created.

### ****Example:****

Tables (Trainer and Subject) and View (VW\_RebellionRider) used in this example are the same as the ones we created in the previous tutorial.

|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8  9  10 | CREATE OR REPLACE TRIGGER io\_update  INSTEAD OF UPDATE ON vw\_rebellionrider  FOR EACH ROW  BEGIN    UPDATE trainer SET FULL\_NAME = :new.full\_name    WHERE FULL\_NAME = :old.full\_name;    UPDATE subject SET subject\_NAME = :new.subject\_name    WHERE subject\_NAME = :old.subject\_name;  END;  / |

On successful execution this trigger will let you execute an Update DML on the view.

# How To Create Instead-Of DELETE Trigger In Oracle PL/SQL

Welcome to the last tutorial on the “INSTEAD-OF” triggers in Oracle database. Till so far in this series we have seen how to create INSTEAD-OF INSERT trigger and INSTEAD-OF UPDATE trigger. The only trigger that is left now is the INSTEAD-OF DELETE trigger which we will cover in today’s tutorial.

Similar to other instead-of triggers which we have seen in previous tutorial, using INSTEAD-OF DELETE we can override the standard action of Delete DML on a view.

### ****Instead-of Delete trigger Example.****

In this example I will again use the View VW\_RebellionRider which we created earlier and have consistently used in this Instead-of trigger series so far.

Needless to say that executing DELETE DML on this view will return an error because of its non-updatable nature. Thus the only way to perform DELETE DML on this view is by using an Instead of trigger. Let’s quickly create one.

|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8 | CREATE OR REPLACE TRIGGER io\_delete  INSTEAD OF DELETE ON vw\_RebellionRider  FOR EACH ROW  BEGIN    DELETE FROM trainer WHERE FULL\_NAME = :old.FULL\_NAME;    DELETE FROM subject WHERE SUBJECT\_NAME= :old.SUBJECT\_NAME;  END;  / |

On successful execution this trigger will allow you to execute DELETE DML on the view.

# What Are PL/SQL Cursors In Oracle Database

What is a database cursor? I guess, is the question which majority of us have faced at least once in our life either during our college studies, or job interview or while doing oracle certification. Database cursor is an important topic from oracle certification as well as from job interview perspective. Thus I am writing this blog by taking both the perspectives in mind so that you will get good marks in your exams as well as ace your Job interview.

## ****What is Database Cursor?****

Cursor is a pointer to a memory area called context area. This context area is a memory region inside the Process Global Area or PGA assigned to hold the information about the processing of a SELECT statement or DML Statement such as INSERT, DELETE, UPDATE or MERGE.

**A quick tip:**  
Refrain from saying that cursor is a pointer to the data stored in the database. Saying this in your interview will definitely put you in an indefinite queue of candidates who never receive a call back.  Because cursor is the pointer to the memory area called context area not to the data of the database.

## ****What is the Context Area?****

Let’s dig a little deeper and see what the context area is?

The context area is a special memory region inside the Process Global Area or PGA which helps oracle server in processing an SQL statement by holding the important information about that statement.

This information includes:

* Rows returned by a query.
* Number of rows processed by a query.
* A pointer to the parsed query in the shared pool.

Using cursor you can control the context area as it is a pointer to the same.

**A Quick Tip**:  
Parsing an SQL statement is the term used for the process that includes the transferring of information to the server, whereby the SQL statement is evaluated as being valid.

## ****Advantages of Cursors****

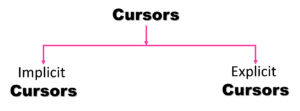
There are two main advantages of a cursor:

1. Cursor is names thus you can reference it in your program whenever you want.
2. Cursor allows you to fetch and process rows returned by a SELECT statement by a row at a time.

## ****Types of cursors in oracle database:****

There are two types of cursors in oracle database:

1. Implicit cursor
2. Explicit cursor



### ****Implicit Cursors in Oracle Database****

As the name suggests these are the cursors which are automatically created by the oracle server every time an SQL DML statement is executed. User cannot control the behavior of these cursors. Oracle server creates an implicit cursor in the background for any PL/SQL block which executes an SQL statement as long as an explicit cursor does not exist for that SQL statement.

Oracle server associates a cursor with every DML statement. Each of the Update & Delete statements has cursors which are responsible to identify those set of rows that are affected by the operation. Also the implicit cursor fulfills the need of a place for an Insert statement to receive the data that is to be inserted into the database.

**Info Byte:**  
**The Most recently opened cursor is called SQL Cursor.**

### ****Explicit Cursor in oracle database****

In contrast to implicit cursors, we have explicit cursors. Explicit cursors are user defined cursors which means user has to create these cursors for any statement which returns more than one row of data. Unlike implicit cursor user has full control of explicit cursor. An explicit cursor can be generated only by naming the cursor in the declaration section of the PL/SQL block.

### ****Advantages of Explicit Cursor:****

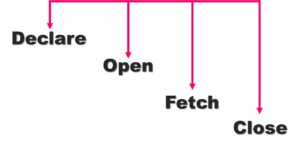
1. Since Explicit cursors are user defined hence they give you more programmatic control on your program.
2. Explicit cursors are more efficient as compared to implicit cursors as in latters case it is hard to track data errors.

### ****Steps for creating an Explicit Cursor****

To create an explicit cursor you need to follow 4 steps. These 4 steps are:

* Declare
* Open
* Fetch
* Close

In case of implicit cursors oracle server performs all these steps automatically for you.



**Info Byte:  
Unless the complete cycle of declaring, opening, fetching and closing has been performed, you can’t use a cursor.**

## ****Declare: How To Declare a Database Cursor?****

Declaring a cursor means initializing a cursor into memory. You define explicit cursor in declaration section of your PL/SQL block and associate it with the SELECT statement.

### ****Syntax****

|  |  |
| --- | --- |
| 1 | CURSOR cursor\_name IS select\_statement; |

## ****Open: How to Open a Database Cursor?****

Whenever oracle server comes across an ‘Open Cursor’ Statement the following four steps take place automatically.

1. All the variables including bind variables in the WHERE clause of a SELECT statement are examined.
2. Based on the values of the variables, the active set is determined, and the PL/SQL engine executes the query for that cursor. Variables are examined at cursor open time.
3. The PL/SQL engine identifies the active set of data.
4. The active set pointer sets to the first row.

**Active set: Rows from all the involved tables that meet the WHERE clause criteria.**

### ****Syntax****

|  |  |
| --- | --- |
| 1 | OPEN cursor\_name; |

## ****Fetch: How to retrieve data from cursor?****

The process of retrieving the data from the cursor is called fetching. Once the cursor is declared and opened then you can retrieve the data from it. Let’s see how.

### ****Syntax****

|  |  |
| --- | --- |
| 1  2  3 | FETCH cursor\_name INTO PL/SQL variable;  Or  FETCH cursor\_name INTO PL/SQL record; |

### ****What happens when you execute fetch command of a cursor?****

1. The use of a FETCH command is to retrieve one row at a time from the active set. This is usually done inside a loop. The values of each row in the active set can then be stored in the corresponding variables or PL/SQL record one at a time, performing operations on each one successively.
2. After completion of each FETCH, the active set pointer is moved forward to the next row. Therefore, each FETCH returns successive rows of the active set, until the entire set is returned. The last FETCH does not assign values to the output variables as they still contain their previous values.

## ****Close: How To Close a Database Cursor?****

Once you are done working with your cursor it’s advisable to close it. As soon as the server comes across the closing statement of a cursor it will relinquish all the resources associated with it.

### ****Syntax****

|  |  |
| --- | --- |
| 1 | CLOSE cursor\_name; |

**Info Byte:**  
**You can no longer fetch from a cursor once it’s closed. Similarly it is impossible to close a cursor once it is already closed. Either of these actions will result in an Oracle error.**

## ****Basic Programming Structure of the Cursor****

Here is the basic programming structure of the cursor in oracle database.

|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8 | DECLARE      CURSOR cursor\_name IS select\_statement;  BEGIN       OPEN cursor\_name;       FETCH cursor\_name INTO PL/SQL variable [PL/SQL record];  CLOSE cursor\_name;  END;  / |

## ****Cursor Attributes****

Oracle provides four attributes which work in correlation with cursors. These attributes are:

* %FOUND
* %NOTFOUND
* %ISOPEN
* %ROWCOUNT

First three attributes ‘Found’, ‘NotFound’ and ‘IsOpen’ are Boolean attributes whereas the last one ‘RowCount’ is a numeric attribute.

Let’s quickly take a look at all these attributes.

#### **%Found**

Cursor attribute ‘Found’ is a Boolean attribute which returns TRUE if the previous FETCH command returned a row otherwise it returns FALSE.

#### **%NotFound**

‘Not Found’ cursor attribute is also a Boolean attribute which returns TRUE only when previous FETCH command of the cursor did not return a row otherwise this attribute will return FALSE.

#### **%IsOpen**

Again ‘Is Open’ Cursor attribute is a Boolean attribute which you can use to check whether your cursor is open or not. It returns TRUE if the cursor is open otherwise it returns FALSE.

#### **%RowCount**

Row count cursor attribute is a numeric attribute which means it returns a numeric value as a result and that value will be the number of records fetched from a cursor at that point in time.

# How To Create An Explicit Cursor In Oracle Database

Cursor is a pointer to a memory area called context area. This we have already learnt with all the other details in the [previous tutorial](http://www.rebellionrider.com/what-are-pl-sql-cursors-in-oracle-database/). Today in this blog we will learn how to create an explicit database cursor.

As we have already learnt that whenever we execute a DML statement, the oracle server creates an implicit cursor in the background. As these cursors are created by oracle server itself thus user does not have much programmatic control on them. In case if you want to control your own DMLs then you need to write an explicit cursor.

So let’s quickly see how you can create your own database cursor in oracle database.

|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8  9  10  11  12  13  14  15  16  17 | SET SERVEROUTPUT ON;  DECLARE    v\_name VARCHAR2(30);    --Declare Cursor    CURSOR cur\_RebellionRider IS    SELECT first\_name FROM EMPLOYEES    WHERE employee\_id < 105;  BEGIN    OPEN cur\_RebellionRider;    LOOP      FETCH cur\_RebellionRider INTO v\_name;      DBMS\_OUTPUT.PUT\_LINE (v\_name);      EXIT WHEN cur\_RebellionRider%NOTFOUND;    END LOOP;--Simple Loop End    CLOSE cur\_RebellionRider;  END;  / |

We used EMPLOYEE table of [HR sample Schema](http://www.rebellionrider.com/how-to-unlock-user-schema-in-oracle-database-using-sql-developer/) for creating the above explicit cursor.  You can watch my Video Tutorial on The Same Topic for line by line explanation of the above code.

Declaration of your cursor can only be done in the “Declaration” section of the PL/SQL block and the rest of the steps of explicit cursor creation cycle must be done in the execution section of a [PL/SQL block](http://www.rebellionrider.com/block-types-in-pl-sql-by-rebellionrider-manish-sharma/).

# How To Create Cursor Parameter In Oracle Database

In [previous PL/SQL tutorial](http://www.rebellionrider.com/how-to-create-an-explicit-cursor-in-oracle-database/) we saw how to create a simple explicit cursor. No doubt that explicit cursor has certain advantages over implicit cursor and can increase the efficiency of DML statements by giving more programmatic controls in user’s hands. Now let’s take a step ahead and learn how to create a parameterized explicit cursor a.k.a. cursor parameter.

Knowledge of cursor creation cycle and simple cursor creation is a must to understand the concept of cursor parameter thus I will highly suggest you to read the previous tutorial on Introduction to cursor and Simple Cursor Creation.

## ****What is Parameterized cursor?****

Unlike simple explicit cursor, parameterized cursors accept values as parameter. You specify the list of parameters separated by comma (,) while declaring the cursor and supply the corresponding argument for each parameter in the list while opening the cursor.

***Definition:  
Cursor parameter can be appropriately defined as an explicit cursor that accepts arguments from the user in the form of parameter.***

### ****Syntax of Parameterized Cursor in Oracle Database****

|  |  |
| --- | --- |
| 1 | CURSOR cur \_ name (parameter list) IS SELECT statement; |

Syntax of declaring a cursor parameter is pretty similar to that of the simple cursor except the addition of parameters enclosed in the parenthesis.

|  |  |
| --- | --- |
| 1 | OPEN cur \_ name (argument list) |

You have to provide corresponding arguments for each parameter that are specified during the declaration. Rest of the steps are the same as that of the simple cursor.

There are few things which you have to take care of while specifying the parameters in your explicit cursor.

* In case of multiple parameters, always separate parameters and the corresponding arguments in the list from each other using comma (,).
* You can specify as many parameters as you need just make sure to include an argument in parameter list for each parameter when you open the cursor.
* While specifying a parameter during the declaration of the explicit cursor only specify the data type not the data width.

## ****Some Wonderful Advantages of Parameterized Cursors****

### ****Makes the cursor more reusable****

You can use a parameter and then pass different values to the WHERE clause each time a cursor is opened instead of hardcoding a value into the WHERE clause of a query to select particular information.

### ****Avoids scoping problems****

When you pass parameters instead of hardcoding the values, the result set for that cursor is not tied to a specific variable in a program or block. Therefore in case your program has nested blocks, you can define the cursor at a higher-level (enclosing) block and use it in any of the sub-blocks with variables defined in those local blocks.

### ****When do we need a parameterized cursor?****

You must be wondering when we need a cursor with parameters in our PL/SQL.

The simplest answer is whenever you need to use your cursor in more than one place with different values for the same WHERE clause of your SELECT statement.

If you can add something to this and have another idea for using a parameterized cursor. Then I am always open to listening to your thoughts do make sure to share it with me on my Facebook or twitter.

### ****Example of Parameterized cursor.****

|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8  9  10  11  12  13  14  15  16  17 | SET SERVEROUTPUT ON;  DECLARE    v\_name VARCHAR2 (30);    --Declare Cursor    CURSOR p\_cur\_RebellionRider (var\_e\_id VARCHAR2) IS    SELECT first\_name FROM EMPLOYEES    WHERE employee\_id < var\_e\_id;  BEGIN    OPEN p\_cur\_RebellionRider (105);  LOOP      FETCH p\_cur\_RebellionRider INTO v\_name;      EXIT WHEN p\_cur\_RebellionRider%NOTFOUND;      DBMS\_OUTPUT.PUT\_LINE(v\_name );    END LOOP;    CLOSE p\_cur\_RebellionRider;  END;  / |

# Cursor For Loop With Simple Explicit Cursor In Oracle Database

As the name suggests Cursor For Loop is a type of For loop provided by oracle PL/SQL which makes working with cursors in oracle database a lot easier by executing OPEN, FETCH & CLOSE Cursor statements implicitly in the background for you.

**Definition:Cursor for Loop Is an Extension of the Numeric For Loop provided by Oracle PL/SQL which works on specified cursors and performs OPEN, FETCH & CLOSE cursor statements implicitly in the background.**

Suggested Reading: [Numeric For Loop In Oracle PL/SQL](http://www.rebellionrider.com/numeric-for-loop-in-pl-sql/)

### ****Syntax of Cursor For Loop.****

|  |  |
| --- | --- |
| 1  2  3  4 | FOR loop\_index IN cursor\_name      LOOP          Statements…      END LOOP; |

### ****Example 1: Cursor For Loop With Simple Explicit Cursor****

|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8  9  10  11  12 | SET SERVEROUTPUT ON;  DECLARE   CURSOR cur\_RebellionRider IS   SELECT first\_name, last\_name FROM employees   WHERE employee\_id >200;  BEGIN    FOR L\_IDX IN cur\_RebellionRider    LOOP      DBMS\_OUTPUT.PUT\_LINE(L\_IDX.first\_name||' '||L\_IDX.last\_name);    END LOOP;  END;  / |

Please watch the Video Tutorial on YouTube channel for detailed explanation of the above code.

### ****Example 2: Cursor For Loop With Inline Cursor.****

You can pass the cursor definition directly into the Cursor For Loop. The code for that is:

|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8  9 | SET SERVEROUTPUT ON;  BEGIN    FOR  L\_IDX  IN (SELECT first\_name, last\_name FROM employees    WHERE employee\_id >200)    LOOP       DBMS\_OUTPUT.PUT\_LINE (L\_IDX.first\_name||' '||L\_IDX.last\_name);    END LOOP;  END;  / |

As you can see in the above code, instead of declaring a cursor into a separate declaration section of PL/SQL block we can write the Cursor’s SELECT DML statement right inside the loop statement after IN keyword.

**Just remember:**

1. Directly write the SELECT statement without specifying the cursor name into the loop statement.
2. Enclose the SELECT statement into parenthesis.
3. Do not terminate the SELECT statement with a semicolon (;)

## ****How many times will Cursor For Loop execute?****

Unlike Numeric For Loop with Cursor For Loop we don’t have minimum or maximum range which will decide the number of execution. So how many times will this loop execute?

This loop will execute for each row returned by the specified cursor and it will terminate either when there is no row to return or there is an occurrence of an exception.