

PL/SQL Practical Guide

INDEX

Sr. No.	Contents	Page No.
1.	PL/SQL	02
2.	Composite Data Types	07
3.	Cursors	11
4.	Exception	14
5.	Types Of Named Sub Programs	15
6.	Functions	17
7.	Packages	19
8.	Database Triggers	22

1. PL / SQL:

- 1 PL/SQL is a procedural extension to a non - procedural language SQL.
- 2 PL/SQL is a Database Language restricted only to do the database activities. (Unlike Other General Purpose Languages like C, C++, Java, Etc)
- 3 PL/SQL Can Have Any Number of Statements, Which Reduces The Network Traffic.
- 4 PL/SQL Program can reside either at the front end or within Oracle Database Server as Stored Subprogram.
- 5 Named PL/SQL Sub-Programs that can be stored within the database are Procedures, Functions, Triggers and Packages.

Valid SQL Statements in PL/SQL:

- 1 All DML Statements (Insert, Update, Delete)
- 2 All TCL (Commit, Rollback)
- 3 All SQL Functions (Single & Group function)
- 4 All SQL Predicates (Where, Having, Group By, Order by)

Invalid SQL Statements in PL/SQL Are:

- 1 DDL Statements (Create, Alter, Etc)
- 2 DCL Statements (Grant, Revoke)

Types of Procedural Statements in PL/SQL:

- 1 Flow Control Statements: If, Exit, Goto, Raise.
- 2 Iterative Statements: Simple Loop, While Loop, For Loop.

Benefits of PL/SQL

- 1 Integration.
- 2 Modularized Application Development.
- 3 Improved Performance.
- 4 Provides Exception / Error Handling Capability.
- 5 Reusability of Code.
- 6 Provides Encapsulation, Overloading, etc.

PL/SQL Block

Declare (Optional)

Variables, Cursors and User defined Exception.

Begin (Mandatory)

SQL, PL/SQL Statements

Exception (Optional)

Action to perform when exception occurs.

End;

Data Types in PL/SQL:

Data Types

Scalar Data Types Composite Data Types

Scalar Data Types:

- Holds Single Value.
- Has No Internal Components.

- 1 Char [Max Length]
- 2 Varchar2 [Max Length]
- 3 Long
- 4 Long Raw
- 5 Number [Precision, Scale]
- 6 Binary_Integer
- 7 Boolean.

Note*: Data Types can be declared with Not Null Constraints. These must be initialized.

The %type Attribute:

- 1 %type Attribute is used to declare a variable as per the data type of an underlying table's column.
- 2 A Variable Declared with the %Type attribute contains the same data type as that of the columns upon which it is declared.

Composite Data Types:

- 1 Composite Data Types Have Internal Components.
- 2 Hence, Composite Data Types can store multiple values that can be manipulated individually.
- 3 Composite Data Types are also known as Collections.

Examples of Composite Data Type:

- 1 *Index by Table*
- 2 Record
- 3 Table of Records

Flow Control Statements:

1. If
2. If ... Elself
3. goto
4. Raise

Iterative Statements:

1. Simple Loop.
2. While Loop.
3. For Loop.

Guide Lines for Using Loops:

- Use the Simple Loop when the statements inside the blocks are to be executed at least once.
- Use the WHILE Loop if the condition needs to be evaluated before each iteration.
- Use the FOR Loop if the number of iterations is known.

Examples:

Declare

```
v_name Char(20);  
v_course Varchar2(20);  
v_duration Number(3)= 30;
```

Begin

```
v_name := 'Sachin';  
v_course := 'Oracle';  
dbms_output.put_line(v_name);  
dbms_output.put_line(v_course);  
dbms_output.put_line(v_duration);
```

End;

Declare

```
    v_name emp.ename%type;
    v_job   emp.job%type;
    v_sal   emp.sal%type;
Begin
    select ename,job,sal
    into v_name, v_job, v_sal
    from emp
    where empno =7902;
    dbms_output.put_line(v_name || ' ' || v_job || ' ' || v_sal);
End;
```

/* IF DEMO */

Declare

```
    a number(2) :=&value_of_a;
    b number(2) :=&value_of_b;
Begin
    if a<b then
        dbms_output.put_line(' Smaller Value is ' || a);
    elsif a>b then
        dbms_output.put_line(' Smaller Value is ' || b);
    else
        dbms_output.put_line(' Both no. are equal ');
    end if;
END;
```

/* SIMPLE LOOP */

DECLARE

```
    i NUMBER(2):= 1;
BEGIN
    LOOP
        dbms_output.put_line(i);
        EXIT WHEN i >= 10;
        i := i+1;
    END LOOP;
END;
```

/* WHILE LOOP */

DECLARE

 a number := 1;

BEGIN

 WHILE a<=10

 LOOP

 dbms_output.put_line(a);

 a:= a + 1;

 END LOOP;

END;

/* FOR DEMO*/

Begin

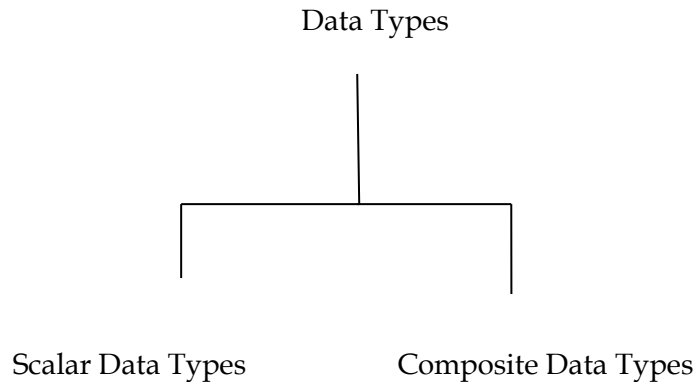
 for i in 1..10 Loop

 dbms_output.put_line(i);

 end Loop;

END;

2. DATA TYPES IN PL/SQL



Composite Data Types:

- Composite Data Types Have Internal Components.
- Hence, Composite Data Types can store multiple values that can be manipulated individually.
- Composite Data Types are also known as Collections.
-

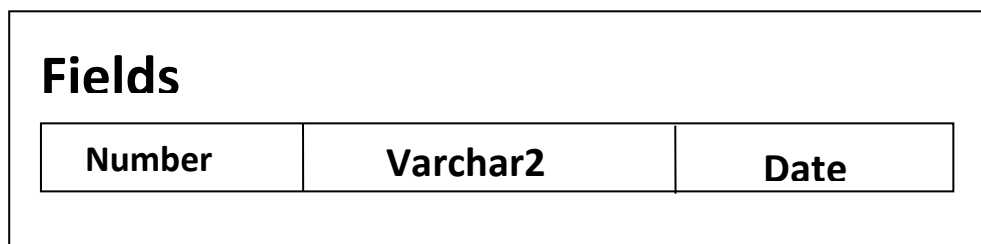
Examples of Composite Data Type:

- *Index by Table*
- Record
- Table of Records

Records:

A Record is a collection of logically related data items of dissimilar data types. It is similar to a row in a table or Structures in C Language.

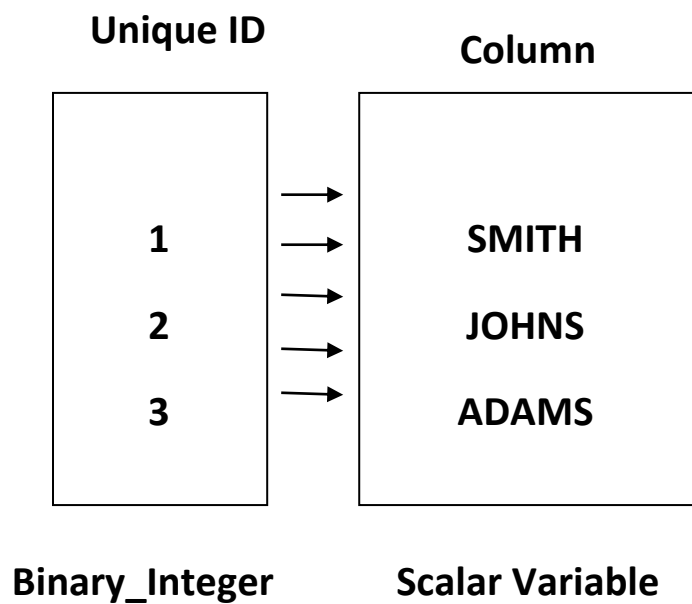
Record



The %RowType Attribute:

The %rowtype Attribute declares a variable according to the number and data types of a database table or view.

Index by Table:



- *Index by Table* is Similar to Arrays.
- Consist of two components:
- A Primary Key of Binary_Integer data type that indexes the Index by Table Elements.
- A Column of Scalar or Record Data Type that stores the elements value.
- Can Increase Dynamically.

Index by Table of Records:

- Records can store a Row from a Table.
- Index by Table can store a Column from a table.

Index by Table of Records is a combination of both. Hence it can Store an entire table.

Examples:

/* Record Demo */

Declare

```
TYPE emp_rec is RECORD
  ( name varchar2(20),
    hiredate date,
    sal number(7)
  );
```

```
v_emp emp_rec;
```

BEGIN

```
select ename, hiredate, sal into v_emp
from emp where empno=&Emp_No;
```

```
dbms_output.put_line(v_emp.name || ' ' || v_emp.sal || ' ' || v_emp.hiredate);
```

```
END;
```

/* Record %0type */

Declare

```
e emp%rowtype;
```

BEGIN

```
select * into e from emp
```

```
where empno=&Emp_No;
```

```
dbms_output.put_line(e.empno || ' ' || e.ename || ' ' || e.sal || ' ' || e.hiredate);
```

```
END;
```

/* Index by Table */

Declare

```
Type t_name is TABLE of VARCHAR2(20)
```

```
index by binary_integer;
```

```
v_name t_name;
```

```
dno number :=10;
```

BEGIN

```
FOR i in 1..4
```

```
LOOP
```

```
select dname into v_name(i)
```

```
from dept
```

```
where deptno = dno;
```

```
dno:=dno+10;
```

```
END LOOP;
```

```

FOR i in 1..4
  LOOP
    dbms_output.put_line(v_name(i));
  END LOOP;
END;

```

/* Index By Table of Records */

Declare

Type type_dept is TABLE of dept%rowtype
index by binary_integer;

v_dept type_dept;

dno number :=10;

BEGIN

FOR i in 1..4

LOOP

select * into v_dept(i)

from dept

where deptno = dno;

dno:=dno+10;

END LOOP;

FOR i in 1..4

LOOP

dbms_output.put_line(v_dept(i).deptno

||' '||v_dept(i).dname||' '||v_dept(i).loc);

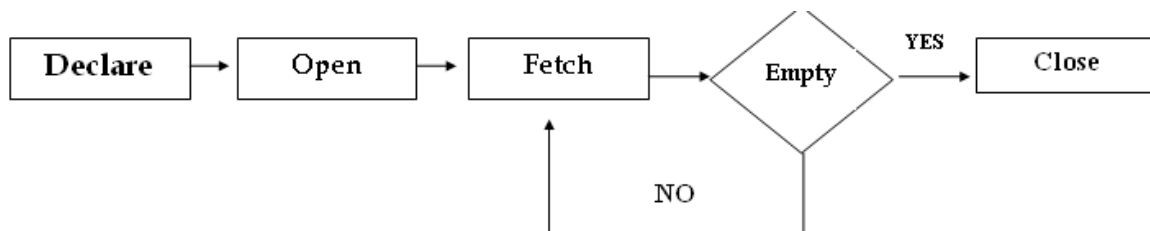
END LOOP;

END;

3. CURSORS

- Whenever you issue a SQL Statement, Oracle opens an area of memory in which the command is parsed and executed. This Area is called CONTEXT AREA.
- The information (Rows) retrieved from the database table, which is available in context area, is known as the ACTIVE SET.
- A Cursor is a pointer to the current row in the ACTIVE SET.
- There are two types of Cursors:
 1. Implicit Cursors: Created, Managed & Erased by Oracle Automatically.
 2. Explicit Cursors: Created & Named by the Programmer.

Controlling Explicit Cursor:



Steps Involved in Creating a Cursor:

1. Create the context area
Syntax: *CURSOR <cursor_name> IS <SQL Query>*;
2. Opening the CURSOR
Syntax: *OPEN <cursor_name>*;
3. Fetch the record into a cursor variable.
Syntax: *FETCH <cursor name> INTO <cursor variable>*;
4. CLOSE the cursor.
Syntax: *CLOSE <cursor name>*;

Cursor Attributes:

- <cursor name> %isopen
- <cursor name> %found
- <cursor name> %notfound
- <cursor name> %rowcount

```
/* CURSOR DEMO */
```

Declare

```
v_ename varchar2(10);
v_job varchar2(10);
v_sal number(4);
cursor c1 is
select ename,job,sal from emp;--Declareing Cursor
BEGIN
open c1; -- Opening Cursor.
loop
  fetch c1 into v_ename,v_job, v_sal;--Fetching
  exit when c1%notfound;
  dbms_output.put_line(v_ename || ' ' || v_job || ' ' || v_sal);
end loop;
close c1; --Closing.
end;
```

```
/* CURSOR WITH RECORD DATA TYPE */
```

Declare

```
cursor empcur is
select * from emp;
v_emp_cur empcur%rowtype;
BEGIN
  OPEN empcur;
  LOOP
    Fetch empcur into v_emp_cur;
    EXIT When empcur%notfound;
    dbms_output.put_line
      (v_emp_cur.empno || v_emp_cur.ename || v_emp_cur.job || v_emp_cur.sal);
  END LOOP;
  dbms_output.put_line(empcur%rowcount || ' records retrived');
CLOSE empcur;
END;
```

```
/* CURSOR WITH PARAMETERS */
```

Declare

```
cursor c1(dno number) is
  select ename,sal from emp where deptno=dno;
empcur c1%rowtype;
BEGIN
open c1(&dno);
loop
  fetch c1 into empcur;
```

```
exit when c1%notfound;
dbms_output.put_line(empcur.ename || ' ' || empcur.sal);
end loop;
close c1;

end;
```

```
/* CURSOR FOR LOOP */
```

Declare

```
cursor c1 is
select empno, job from emp;
```

```
BEGIN
```

```
for empcur in c1 --auto open and fetch
```

```
LOOP
```

```
dbms_output.put_line
```

```
(empcur.empno || ' ' || empcur.job);
```

```
END LOOP; -- auto close
```

```
END;
```

4. EXCEPTIONS

```
/* PRE DEFINED NAMED EXCEPTIONS */
```

Declare

```
    e emp%rowtype;
```

BEGIN

```
    select * into e from emp
    where empno=&empno;
```

EXCEPTION

```
    when no_data_found then
    dbms_output.put_line(' So such Emp Exist ');
    when invalid_column_name then
    dbms_output.put_line(' Datatype mismatch ');
    when others then
    dbms_output.put_line(' some error occured ');
```

END;

```
/* PRE DEFINED UN-NAMED EXCEPTION */
```

Declare

```
    exp_integrity exception;
    pragma exception_init
    (exp_integrity, -02292);
```

begin

```
    delete from dept where deptno=&deptno;
```

exception

```
    when exp_integrity then
    dbms_output.put_line(' cant delete dept records, child records exist ');
    when others then
    dbms_output.put_line(' Some Error Occured ');
```

end;

5. Types of Named Sub Programs

- 1> Procedures
- 2> Functions
- 3>

Procedures:

- A Procedure is a named PL/SQL Block, stored in the database.
- A Procedure is generally used to perform an action.
- A Procedure may or may not return a value.
- When a procedure is first created, it is compiled and stored with in the database in compiled form. This compiled code allows reusability and performance benefits.
- Parameter can have three modes in a procedure, IN, OUT & INOUT mode.

Privileges:

SQL> grant create procedure to user_name; (DBA)

SQL> Grant Execute on <procedure_name> to user_name (owner)

Data Dictionary Views:

- User_procedures (General Info)
 - User_source (the text of pl/sql procedure)
 - Desc procedure_name (IN, OUT, INOUT parameters list)
 - User_errors (to see all the compilation errors in a procedure).
- OR

SQL> show error;

Show err;

/* PROCEDURE TO ADD A Record in Dept Table */

create or replace procedure add_dept

(p_dno in number default 10,

p_name in varchar2 default 'IT',

p_loc in varchar2 default 'HYD')

as

begin

insert into dept values(p_dno, p_name, p_loc);

end;

```
/* PROC To Fetch Data From Emp Table */  
create or replace procedure get_emp  
    (p_eno in number,  
     p_name out varchar2,  
     p_job out varchar2,  
     p_sal out number  
    )  
is  
begin  
    select ename, job, sal into  
    p_name,p_job, p_sal from emp  
    where empno = p_eno;  
end;
```


6. FUNCTIONS

- Function is a named PL/SQL Block that returns a value.
- A Function can be stored in the database as a schema object for repeated execution.
- A function is called as part of an expression.
- Functions and Procedures are structured alike. Procedures are used to perform a task and Functions are used to compute values.

Location to call User-Defined Functions:

- Select Command.
- Where, Group by, Having & Order by Clauses.
- In an Insert Statement.
- In Update Statement.

Restrictions On Functions:

- Functions Called from a SQL Statements cannot have DML statements.
- Functions called from an update / delete statement on a table XYZ cannot perform DML on the same table XYZ.
- Functions called from a DML statement on a table cannot query the same table.
- Functions called from a SQL statement cannot contain COMMIT or ROLLBACK statement.

Getting Function Info:

USER_OBJECTS:

```
SQL> Select object_name from user_objects where object_type = 'FUNCTION';
```

USER_SOURCE:

```
SQL> Select text from user_source where name = 'FUNC_NAME';
```

Dropping a Function:

```
SQL > Drop Function Function_Name;
```

```
CREATE OR REPLACE FUNCTION get_annsal (p_id number) return number
```

```
as
```

```
    v_salary number(10);
```

```
BEGIN
```

```
    select sal*12 into v_salary from emp
```

```
    where empno = p_id;
```

```
    return v_salary;
```

```
END;
```

```
CREATE OR REPLACE FUNCTION tax (p_sal number) return number
```

```
as
```

```
    v_tax number(8,2) :=0;
```

```
BEGIN
```

```
    if p_sal between 0 and 2000 then
```

```
        v_tax := p_sal * 0.10;
```

```
    elsif p_sal between 2001 and 4000 then
```

```
        v_tax := p_sal *0.15;
```

```
    else
```

```
        v_tax := p_sal * 0.25;
```

```
    end if;
```

```
    return v_tax;
```

```
END;
```

```
CREATE OR REPLACE FUNCTION emp_exp (p_eno number) return number
```

```
as
```

```
    hdate date;
```

```
    e number;
```

```
BEGIN
```

```
    select hiredate into hdate
```

```
    from emp
```

```
    where empno = p_eno;
```

```
    e := months_between(sysdate, hdate) / 12;
```

```
    return round(e);
```

```
end;
```

7. PACKAGES

- Packages are used to bundle together a group of logically related Sub-Programs.
- A Package Consist of two parts:
 - Package Specification &
 - Package Body
- Both of which are stored independently in the Data Dictionary.
- The Constructs (sub programs) mentioned in the package specification are PUBLIC constructs. The Constructs described in the package body, but not mentioned in the specification are PRIVATE constructs.
- A Package itself can't be invoked, parameterizes or nested.
- When one Sub-Program from the packages is called, the entire package is loaded in the memory providing faster access to other Sub-Programs.

Advantages of Package:

- Modularity
- Encapsulation
- Overloading
- Better Performance

Privileges:

Create Procedure (DBA).

Execute (owner)

Data Dictionary View:

User_procedures, User_objects, User_source.

CREATE OR REPLACE PACKAGE my_pack is

FUNCTION get_annsal(p_id number) return number;

FUNCTION tax(p_id number) return number;

PROCEDURE get_emp(p_eno in number,

p_name out varchar2,

p_job out varchar2,

p_sal out number);

end my_pack;

```

CREATE OR REPLACE PACKAGE BODY my_pack
is
  /* GET_ANNSAL FUNCTION */
  FUNCTION get_annsal(p_id number)
  return number
  as
    v_salary emp.sal%type;
  BEGIN
    select sal*12 into v_salary from emp
    where empno = p_id;
    return v_salary;
  END;
  /* TAX FUNCTION */
  FUNCTION tax(p_id number)
  return number
  as
    v_tax number(8,2) :=0;
    v_salary number(6);
  BEGIN
    select sal*12 into v_salary from emp
    where empno=p_id;
    if v_salary between 0 and 2000 then
      v_tax := v_salary * 0.10;
    elsif v_Salary between 2001 and 4000 then
      v_tax := v_salary *0.15;
    else
      v_tax := v_salary * 0.25;
    end if;
    return v_tax;
  END;
  /* GET_EMP */
  PROCEDURE get_emp(p_eno in number,
    p_name out varchar2,
    p_job out varchar2,
    p_sal out number
  )
is
begin
  select ename, job, sal into
    p_name,p_job, p_sal from emp
  where empno = p_eno;
end;
end my_pack;

```

```

/* Function Overloading in a Package */
create or replace package operation
is
function add(x number, y number) return number;
function add(x varchar2, y varchar2) return varchar2;
function add(x date, y number) return date;
end operation;
create or replace package body operation
is
function add(x number, y number) return number
is
    v_ans number;
Begin
    v_ans := x + y;
    return v_ans;
End;
function add(x varchar2, y varchar2) return varchar2
is
    v_ans varchar2(40);
Begin
    v_ans := x || y;
    return v_ans;
End;
function add(x date, y number) return date
is
    v_ans date;
Begin
    v_ans := x + y;
    return v_ans;
End;
end operation;
/* An Anonymous PL/SQL Block To Call Operation.add */
Declare
    a number;
    b varchar2(30);
    c date;
begin
    a := operation.add(23,3);
    dbms_output.put_line('the value of a is ' || a);
    b :=operation.add('Active','Net');
    dbms_output.put_line('the value of b is ' || b);
    c :=operation.add(sysdate,8);
    dbms_output.put_line('the value of c is ' || c);
end;

```

8. DATABASE TRIGGERS

- A Database Trigger is a PL/SQL Block, which is associated with a table, view, schema or the entire database.
- Executes Implicitly (Automatically) whenever a particular event takes place.
- Can be of Two types:
 1. Schema Level Trigger: Fires of each event (DML) for that particular user.
 2. System Trigger: Fires for each event for all users.

Schema Level Triggers:

Based on Tables and Views in a Schema.

Triggering Event:

Event upon which the trigger will be fired

i.e. body of the trigger will be executed.

Eg: Insert, update, Delete, Instead of (views).

Trigger Timing:

When should the trigger fire. Before or After the EVENT.

Trigger Types:

Statement Level: (default for tables)

Executed once for the Entire DML Operation.

Row Level: (default for views)

Executed once for each row affected by the event.

Note: Triggers Cannot Contain Commit, Savepoint or Rollback Statements.

```
SQL> Alter Trigger Trigger_Name Disable | Enable;
```

```
SQL> Alter Table EMP Disable | Enable All Triggers;
```

```
SQL> Drop Trigger Trigger_Name;
```

```
SQL> DESC user_triggers;
```

Note: When a table is dropped all trigger on that table are also dropped.

INSTEAD OF Trigger

- A View consisting of Group Function, Group by Clause, Join Condition, etc is called a complex view.
- DML Operations cannot be performed directly on a Complex view.
- Def: To Perform DML operations through a complex view, we can use an INSTEAD of Trigger. The DML operation is targeted at the Base Table the view refers to.
- Instead of Triggers can only be ROW LEVEL Triggers.

Database Trigger / System Triggers

1. Create. (DB or Schema Level)
2. Alter. “
3. Drop. “
4. Log on. “
5. Log Off. “
6. Startup. (DB Level Only).
7. Shut Down. “
8. A Specific Error or Any Error Being Raised. “

```
/* Dept Backup Trigger */
create or replace trigger dept_backup
before delete
on dept
for each row
Begin
    insert into dept_backup values
    (:old.deptno, :old.dname, :old.loc);
end;
/* Sal Check Trigger - With User Defined Exception */
create or replace trigger sal_check
before update
on emp
for each row
begin
    if :new.sal < :old.sal then
        raise_application_error(-20006,'You Cannot Decrease an emp's Sal');
    end if;
End;

/* Day & Time Check Trigger - With User Defined Exception */

CREATE or REPLACE TRIGGER day_time_check
BEFORE
```

```

INSERT OR UPDATE OR DELETE
ON EMP
DECLARE
    d varchar2(3);
    t number(2);
BEGIN
    d :=to_char(sysdate,'DY');
    t := to_char(sysdate, 'HH24');

    if d in ('SAT','SUN') then
        raise_application_error
            (-20005, ' Today is Saturday / Sunday. Transactions are not allowed on weekends.
');
    end if;

    if t NOT between 09 AND 17 Then
        raise_application_error
            (-20006, ' Tx Allowed Between 09 AM Till 6 PM Only ');
    end if;

end;

/* Database Level Logon Trigger - To Be Create By Sys */
create or replace trigger logon_trig
after
logon ON Database
Declare

begin

    insert into log values(user, sysdate);
end;

```