**Ex.No.11** **PL/SQL BLOCK – EXCEPTIONS AND TRIGGERS**

**AIM:**

To implement and execute PL/SQL exceptions and triggers

**DESCRIPTION:**

**EXCEPTIONS**

In PL/SQL, the user can catch certain runtime errors. Exceptions can be internally defined by Oracle or the user. Exceptions are used to handle errors that occur in your PL/SQL code. A PL/SQL block contains an EXCEPTION block to handle exception.

There are three types of exceptions:

1. Predefined Oracle errors
2. Undefined Oracle errors
3. User-defined errors

The different parts of the exception.

1. Declare the exception.
2. Raise an exception.
3. Handle the exception. An exception has four attributes:
   1. Name provides a short description of the problem.
   2. Type identifies the area of the error.
   3. Exception Code gives a numeric representation of the exception.
   4. Error message provides additional information about the exception.

The predefined divide-by-zero exception has the following values for the attributes:

1. Name = ZERO\_DIVIDE
2. Type = ORA (from the Oracle engine)
3. Exception Code = C01476

Error message = divisor is equal to zero

**EXCEPTION HANDLING**

PL/SQL provides a feature to handle the Exceptions which occur in a PL/SQL Block known as exception Handling. Using Exception Handling we can test the code and avoid it from exiting abruptly.When an exception occurs messages which explains its cause is received. PL/SQL

Exception message consists of three parts. **Type of Exception An Error Code A message**

**STRUCTURE OF EXCEPTION HANDLING**

**GENERAL SYNTAX FOR CODING THE EXCEPTION SECTION**

DECLARE

Declaration section

BEGIN

Exception section

EXCEPTION

WHEN ex\_name1 THEN -Error handling statements

WHEN ex\_name2 THEN -Error handling statements

WHEN Others THEN

-Error handling statements

END;

**Types of Exception**

There are 2 types of Exceptions.

1. System Exceptions
2. User-defined Exceptions

**a) System Exceptions**

System exceptions are automatically raised by Oracle, when a program violates a RDBMS rule. There are some system exceptions which are raised frequently, so they are pre-defined and given a name in Oracle which are known as Named System Exceptions.

**For example:** NO\_DATA\_FOUND and ZERO\_DIVIDE are called Named System exceptions.

**For Example:** Suppose a NO\_DATA\_FOUND exception is raised in a proc, we can write acode to handle the exception as given below.

BEGIN

Execution section EXCEPTION

WHEN NO\_DATA\_FOUND THEN

dbms\_output.put\_line ('A SELECT...INTO did not return any row.');

END;

/

**b) User-defined Exceptions**

PL/SQL allows us to define our own exceptions according to the need of our program. A user-defined exception must be declared and then raised explicitly, using a RAISE statement.

To define an exception we use EXCEPTION keyword as below: EXCEPTION\_NAME EXCEPTION;

To raise exception that we‟ve defined to use the RAISE statement as follows:

RAISE EXCEPTION\_NAME

**Raising Exceptions**

Exceptions are raised by the database server automatically whenever there is any internal database error, but exceptions can be raised explicitly by the programmer by using the command

**RAISE**. Following is the simple syntax of raising an exception:DECLARE

exception\_name EXCEPTION; BEGIN

IF condition THEN RAISE exception\_name;

END IF;

EXCEPTION

WHEN exception\_name THEN statement;

END;

**TRIGGERS**:

A trigger is a pl/sql block structure which is fired when a DML statements like Insert, Delete, Update is executed on a database table. A trigger is triggered automatically when an associated DML statement is executed. Sometimes a trigger is referred as a special kind of stored procedure in term of procedural code inside its body. The difference between a trigger and a stored procedure is that a trigger is activated or called when an event happens in a database table, a stored procedure must be called explicitly. For example you can have some business logic to do before or after inserting a new record in a database table.

## **Types of PL/SQL Triggers**:

There are two types of triggers based on which level it is triggered.  
**1) Row level trigger** - An event is triggered for each row upated, inserted or deleted.   
**2) Statement level trigger** - An event is triggered for each sql statement executed.

## **PL/SQL Trigger Execution Hierarchy**:

The following hierarchy is followed when a trigger is fired.

**1)** BEFORE statement trigger fires first.  
**2)** Next BEFORE row level trigger fires, once for each row affected.   
**3)** Then AFTER row level trigger fires once for each affected row. These events will alternates between BEFORE and AFTER row level triggers.  
**4)** Finally the AFTER statement level trigger fires.

### The Syntax for creating a trigger is:

CREATE [OR REPLACE] TRIGGER trigger\_name

{BEFORE | AFTER | INSTEAD OF }

{INSERT [OR] | UPDATE [OR] | DELETE}

[OF col\_name]

ON table\_name

[REFERENCING OLD AS o NEW AS n]

[FOR EACH ROW]

WHEN (condition)

BEGIN

--- sql statements

END;

* CREATE [OR REPLACE] TRIGGER trigger\_name - This clause creates a trigger with the given name or overwrites an existing trigger with the same name.
* {BEFORE | AFTER | INSTEAD OF } - This clause indicates at what time the trigger should get fired. i.e. for example: before or after updating a table. INSTEAD OF is used to create a trigger on a view. before and after cannot be used to create a trigger on a view.
* {INSERT [OR] | UPDATE [OR] | DELETE} *-* This clause determines the triggering event. More than one triggering events can be used together separated by OR keyword. The trigger gets fired at all the specified triggering event.
* [OF col\_name] *-* This clause is used with update triggers. This clause is used when you want to trigger an event only when a specific column is updated.
* CREATE [OR REPLACE] TRIGGER trigger\_name - This clause creates a trigger with the given name or overwrites an existing trigger with the same name.
* [ON table\_name] *-* This clause identifies the name of the table or view to which the trigger is associated.
* [REFERENCING OLD AS o NEW AS n] - This clause is used to reference the old and new values of the data being changed. By default, you reference the values as :old.column\_name or :new.column\_name. The reference names can also be changed from old (or new) to any other user-defined name. You cannot reference old values when inserting a record, or new values when deleting a record, because they do not exist.
* [FOR EACH ROW] - This clause is used to determine whether a trigger must fire when each row gets affected ( i.e. a Row Level Trigger) or just once when the entire sql statement is executed(i.e.statement level Trigger).
* WHEN (condition) - This clause is valid only for row level triggers. The trigger is fired only for rows that satisfy the condition specified.

**EXAMPLE PROGRAMS**

SQL> set serveroutput on

SQL> begin

2 dbms\_output.put\_line(1/0);

3 exception

4 when zero\_divide then

5 dbms\_output.put\_line('divide by zero');

6 end;

7

8 /

OUTPUT

divide by zero

PL/SQL procedure successfully completed.

NO DATA FOUND

SQL> select \* from customer\_1;

CUSID CNAME CADD

---------- ---------- ----------

1 aaa xxx

2 bbb yyy

SQL> declare

2 c\_id customer\_1.cusid%type;

3 c\_name customer\_1.cname%type;

4 c\_addr customer\_1.cadd%type;

5 begin

6 select cname ,cadd into c\_name,c\_addr from customer\_1 where cusid=c\_id;

7 dbms\_output.put\_line('name:'||c\_name);

8 dbms\_output.put\_line('add:'||c\_addr);

9 exception

10 when no\_data\_found then

11 dbms\_output.put\_line('no such customers');

12 when others then

13 dbms\_output.put\_line('error');

14 end;

15 /

OUTPUT

no such customers

PL/SQL procedure successfully completed.

INVALID NUMBER

begin

insert into emp\_5(empno)values('101x');

exception

when invalid\_number then

dbms\_output.put\_line('conversion of string to number failed');

end;

SQL> begin

2 insert into emp\_5(empno)values('101x');

3 exception

4 when invalid\_number then

5 dbms\_output.put\_line('conversion of string to number failed');

6 end;

7 /

OUTPUT

conversion of string to number failed

PL/SQL procedure successfully completed

OTHER EXCEPTION

begin

dbms\_output.put\_line(1/0);

exception

when others then

dbms\_output.put\_line('an exception occured');

end;

TRIGGER

create or replace trigger dis\_chan\_sal

before insert or update or delete on c3

for each row

when(new.ID>0)

declare

sal number;

begin

sal:=:new.salary-:old.salary;

dbms\_output.put\_line('old salary'||:old.salary);

dbms\_output.put\_line('new salary'||:new.salary);

dbms\_output.put\_line('salary diff'||sal);

end;

OUTPUT

SQL> update c3 set salary=40000 where id=102;

old salary40000

new salary40000

salary diff0

1 row updated.=102;

SQL> update c3 set salary=90000 where id=102;

old salary40000

new salary90000

salary diff50000

1 row updated.