

Experiment No.-5-6

AIM: The Run-time errors arise from design faults, coding mistakes, hardware failures, and many other sources. Although you cannot anticipate all the possible errors, you can plan to handle certain kinds of errors meaningful to your PL/SQL program. With PL/SQL mechanism called exception handling design “bulletproof” program so that it can continue operating in the presence of errors.

THEORY:

An exception is an error condition during a program execution. PL/SQL supports programmers to catch such conditions using EXCEPTION block in the program and an appropriate action is taken against the error condition.

There are mainly two types of exceptions:

1. System-defined exception
2. User-defined exception

System-defined Exception: This type of exceptions is pre-defined in the oracle. These are executed when any of the database rule is violated by a program. The various system-defined exceptions are as follows-

1. **NO_DATA_FOUND**- It is raised when a select into statement returns no rows.
2. **VALUE_ERROR**- It is raised when an arithmetic conversion, truncation or size constraint error occurs.
3. **ZERO_DIVIDE**- It is raised when an attempt is made to divide a number by zero.
4. **TOO_MANY_ROWS**- It is raised when a select into statement returns more than one row.
5. **CASE_NOT_FOUND**- It is raised when none of the choices in the when clause of a case statement is selected and there is no else clause.

Example 1- Write a program to check if the input is a valid number or not.

Declare

vish number:=:vish;

Roll no. : 1610991565

Name: Nidhi Mittal

Begin

```
Dbms_output.put_line('Number='|| vish);
```

Exception

```
when value_error then Dbms_output.put_line('Value entered is not a number');
```

End;

127.0.0.1:8080/apex/f?p=4500:1

:VISH

a

OUTPUT:

Results Explain Describe Saved SQL History

Entered value is not a number

Statement processed.

0.23 seconds

Example 2- Write a program to find the result of an operation performed on two numbers.

Declare

```
v number:=:v;
```

Roll no. : 1610991565

Name: Nidhi Mittal

```
vv number:=:vv;
```

```
o varchar(1):=:o;
```

```
r number:=0;
```

```
Begin
```

```
case o
```

```
when '+' then r:=v+vv;
```

```
when '-' then r:=v-vv;
```

```
when '*' then r:=v*vv;
```

```
when '/' then r:=v/vv;
```

```
when '%' then r:=v mod vv;
```

```
End case;
```

```
Dbms_output.put_line('Result= ' || r);
```

```
Exception
```

```
when zero_divide then Dbms_output.put_line('Division by zero not possible');
```

```
End;
```

127.0.0.1:8080/apex/f?p=

:v 3
:vv 0
:o /

OUTPUT:

```
Division by zero not possible
```

```
Statement processed.
```

```
- - - .
```

Example 3- Write a code to fetch the details of an employee who works in a particular department.

Declare

```
vish emp%rowtype;
```

```
dpt emp.deptno%type:=:dpt;
```

Begin

```
select * into vish from emp where deptno=dpt;
```

```
Dbms_output.put_line('Name: ' || vish.ename);
```

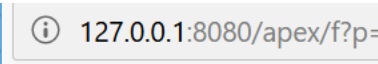
```
Dbms_output.put_line('Employee ID: ' || vish.empno);
```

```
Dbms_output.put_line('Designation: ' || vish.job);
```

Exception

```
when too_many_rows then Dbms_output.put_line('More than one employee works in the department');
```

End;



:DPT

OUTPUT:

Roll no. : 1610991565

Name: Nidhi Mittal

Results Explain Describe Saved SQL History

More than one employee works in the department.

Statement processed.

0.00 seconds

Example 4- Write a code to print the remarks on the basis of grades.

Declare

vish varchar(1):=:vish;

Begin

case vish

when 'A' then Dbms_output.put_line('Excellent');

when 'B' then Dbms_output.put_line('Very Good');

when 'C' then Dbms_output.put_line('Well Done');

when 'D' then Dbms_output.put_line('Passed');

when 'F' then Dbms_output.put_line('Fail');

End case;

Exception

when case_not_found then Dbms_output.put_line('No available grade');

End;

Roll no. : 1610991565

Name: Nidhi Mittal



:VISH Z

OUTPUT:

Results Explain Describe

No available grade

Statement processed.

0.00 seconds

Example 5- Write a code to retrieve the information of a department from the dept table.

Declare

vish dept%rowtype;

dn dept.deptno%type:=:dn;

Begin

select * into vish from dept where deptno=dn;

Dbms_output.put_line('Department name: ' || vish.dname);

Dbms_output.put_line('Location: ' || vish.loc);

Exception

when no_data_found then Dbms_output.put_line('Department not found');

End;

:DN 11

Roll no. : 1610991565

Name: Nidhi Mittal

OUTPUT:

Results	Explain	Describe	S
---------	---------	----------	---

Department not found

Statement processed.

0.01 seconds

Experiment No.-7

AIM: PL/SQL gives you control to make your own exception base on oracle rules. User define exceptions must be declared yourself and RAISE statement is used to raise them explicitly. Use PL/SQL user defined exception to make your own exception.

THEORY:

We have majorly two kinds of exceptions in PL/SQL

Example 1- Write a code to check whether student is male otherwise raise exception.

declare

v_gender varchar2(10);

message varchar2(250);

my_exception exception;

begin

SELECT gender INTO v_gender from v_stu WHERE id = 6;

dbms_output.put_line('Gender is: '||v_gender);

IF v_gender = 'M' THEN

message := 'MALE';

dbms_output.put_line('Message: '||message);

ELSE

RAISE my_exception;

END IF;

exception

when my_exception then

dbms_output.put_line('Only males are allowed');

end;

OUTPUT:

Results Explain Describe

```
Gender is: F
Only males are allowed

Statement processed.
```

Example 2- Write a code to check the stocks of sports equipment.

declare

a number:=a;

b number:=b;

c number:=c;

ebat exception;

eball exception;

epad exception;

q_bat pls.quant%type;

q_ball pls.quant%type;

q_pad pls.quant%type;

begin

select quant into q_bat from pls where prod='bat';

select quant into q_ball from pls where prod='ball';

select quant into q_pad from pls where prod='pad';

if(a>q_bat)then

Roll no. : 1610991565

Name: Nidhi Mittal



```
raise ebat;

else

dbms_output.put_line('cost=||a*4000);

update pls set quant=quant-a where prod='bat';

if(b>q_ball)then

raise eball;

else

dbms_output.put_line('cost=||b*100);

update pls set quant=quant-b where prod='ball';

if(c>q_pad)then

raise epad;

else

dbms_output.put_line('cost=||c*50);

update pls set quant=quant-c where prod='pad';

end if;end if;end if;

exception

when ebat then

dbms_output.put_line('bat out of stock');

when eball then

dbms_output.put_line('ball out of stock');

when epad then

dbms_output.put_line('pad out of stock');

end;
```

Roll no. : 1610991565

Name: Nidhi Mittal

:A
:B
:C

OUTPUT:

Results Explain Describe

```
cost=4000  
cost=200  
pad out of stock  
  
Statement processed.
```

Example 3- Write a code to check if the user enters a valid id or not.

Declare

id number:=:id;

invalid exception;

Begin

if(id<0) then raise invalid;

else

Dbms_output.put_line('ID: '|| id);

Dbms_output.put_line('Valid ID');

End if;

Exception

when value_error then Dbms_output.put_line('Input ID is not a number');

when invalid then Dbms_output.put_line('Input ID is not valid');

Roll no. : 1610991565

Name: Nidhi Mittal



End;

OUTPUT:

Example 4- Write a code to check if the salary of an employee is always greater than 10000.

Declare

```
empName varchar2(10):=empName;
```

```
salary number:=:salary;
```

```
salInvalid exception;
```

Begin

```
if(salary<10000) then raise salInvalid;
```

```
else
```

```
  Dbms_output.put_line('Employee name: ' || empName);
```

```
  Dbms_output.put_line('Salary: ' || salary);
```

```
End if;
```

Exception

```
when salInvalid then Dbms_output.put_line('Salary for ' || empName || ' cannot be less than 10000');
```

End;

OUTPUT:

```
Salary for vishal cannot be less than 10000
```

```
Statement processed.
```

```
0.00 seconds
```

Example 5- Write a code that asks the user an option to book or enquire for an airline tickets. Using the user defined exceptions make the program user friendly that works favourably on any input given by the user.

Declare

```
book_enquiry varchar(1):=:book_enquiry;
```

```
airline_Name air.name%type:=:airline_Name;
```

```
tickets air.qty%type:=:tickets;
```

```
Availability exception;
```

```
airline_info air%rowtype;
```

Begin

```
select * into airline_info from air where name=:airline_Name;
```

```
case book_enquiry
```

```
when 'b' then
```

```
if(tickets>airline_info.qty) then raise Availability;
```

```
else
```

```
update air set qty=qty-tickets where name=:airline_Name;
```

Roll no. : 1610991565

Name: Nidhi Mittal



```
Dbms_output.put_line('Tickets booked. Thank You!');

End if;

when 'e' then

Dbms_output.put_line('Airline name: ' || airline_info.name);

Dbms_output.put_line('Seats available: ' || airline_info.qty);

Dbms_output.put_line('Thank You!');

End case;

Exception

when no_data_found then Dbms_output.put_line('No entry of airline ' || airline_Name || 'in the
airlines table. Thank You!');

when Availability then Dbms_output.put_line('No. of tikets requested are not available. Thank
You!');

End;
```

:BOOK_ENQUIRY	b
:AIRLINE_NAME	king fisher
:TICKETS	200

OUTPUT:

Results Explain Describe Saved SQL History

No. of tikets requested are not available. Thank You!

1 row(s) updated.

0.02 seconds

Example 6- Write a code to either create or break the fixed deposit for a customer of the bank_customer table. Also raise an exception when felt required

Declare

```
cust_name bank.name%type:=:cust_name;
```

```
create_break varchar(1):=:create_break;
```

```
customer bank%rowtype;
```

```
amount bank.fd%type:=:amount;
```

```
noFD exception;
```

Begin

```
select * into customer from bank where name=cust_name;
```

```
case create_break
```

```
when 'c' then
```

```
Dbms_output.put_line('FD created. Thank You!');
```

```
update bank set fd=fd+amount where name=cust_name;
```

```
when 'b' then
```

```
if(customer.fd=null) then raise noFD;
```

```
else
```

```
Dbms_output.put_line('FD brokeed. Thank You!');
```

```
update bank set fd=null where name=cust_name;
```

```
End if;
```

```
End case;
```

Exception

```
when noFD then Dbms_output.put_line('Customer does not have an FD. Thank You!');
```

Roll no. : 1610991565

Name: Nidhi Mittal

End;

:CUST_NAME	vishal
:CREATE_BREAK	c
:AMOUNT	5000

OUTPUT:

Results Explain Describe Sa

FD created. Thank You!

1 row(s) updated.

0.01 seconds

EXPERIMENT NO.: 08 - 09

AIM: Create a temporary work area in the system memory whenever a SQL statement is executed. This temporary work area will be used to store the data retrieved from the database, and manipulate this data. Create A cursor that can hold more than one row, but can process only one row at a time.

THEORY:

A cursor is a temporary work area created in the system memory when a SQL statement is executed. A cursor contains information on a select statement and the rows of data accessed by it.

This temporary work area is used to store the data retrieved from the database, and manipulate this data. A cursor can hold more than one row, but can process only one row at a time. The set of rows the cursor holds is called the *active set*.

There are two types of cursors in PL/SQL:

- Implicit Cursors
- Explicit Cursors

Implicit cursors

These are created by default when DML statements like, INSERT, UPDATE, and DELETE statements are executed. They are also created when a SELECT statement that returns just one row is executed.

Oracle provides few attributes called as implicit cursor attributes to check the status of DML operations. The cursor attributes available are **%FOUND**, **%NOTFOUND**, **%ROWCOUNT**, and **%ISOPEN**.

Attributes	Return Value	Example
%FOUND	The return value is TRUE, if the DML statements like INSERT, DELETE and UPDATE affect at least one row and if SELECT ...INTO statement return at least	SQL%FOUND

	one row.	
	The return value is FALSE, if DML statements like INSERT, DELETE and UPDATE do not affect row and if SELECT....INTO statement do not return a row.	
%NOTFOUND	The return value is FALSE, if DML statements like INSERT, DELETE and UPDATE at least one row and if SELECTINTO statement return at least one row.	SQL%NOTFOUND
	The return value is TRUE, if a DML statement like INSERT, DELETE and UPDATE do not affect even one row and if SELECTINTO statement does not return a row.	
%ROWCOUNT	Return the number of rows affected by the DML operations INSERT, DELETE, UPDATE, SELECT	SQL%ROWCOUNT

Example - Program to update the balance of an employee

declare

begin

update bank set bal=bal+(0.1*bal) where name='vish';

```
if(SQL%notfound) then  
    dbms_output.put_line('no updation');  
elsif (SQL%found) then  
    dbms_output.put_line(SQL%rowcount);  
end if;  
end;
```

OUTPUT:

Results	Explain	Describe	Save
---------	---------	----------	------

2

Statement processed.

0.01 seconds

Explicit cursors

They must be created when you are executing a SELECT statement that returns more than one row. Even though the cursor stores multiple records, only one record can be processed at a time, which is called as current row. When you fetch a row the current row position moves to next row.

Both implicit and explicit cursors have the same functionality, but they differ in the way they are accessed.

EXAMPLE- Program to print the name of an employee providing his ID

declare

 cursor c1(n number) is select * from office where id=n;

 vemp office%rowtype;

begin

 open c1(:n);

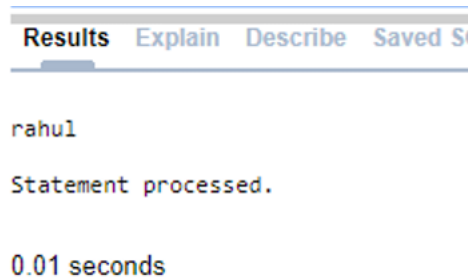
 while true loop

Roll no. : 1610991565

Name: Nidhi Mittal

```
        fetch c1 into vemp;  
        if(c1%notfound) then  
            exit;  
        end if;  
        dbms_output.put_line(vemp.name);  
    end loop;  
end;
```

Output:



The screenshot shows the SQL Developer interface with the 'Results' tab selected. It displays the output of the PL/SQL block, which is 'rahul'. Below the output, it states 'Statement processed.' and '0.01 seconds'.

EXAMPLE- Program to update the quantity of items check whether they are available in stock or not.

```
declare  
cursor c1 is select * from pro;  
vpro pro%rowtype;  
n1 number:=:n1;  
n2 number:=:n2;  
n3 number:=:n3;  
lap pro.qnt%type;  
mob pro.qnt%type;  
ip pro.qnt%type;  
out_of_stock exception;  
begin
```

Roll no. : 1610991565

Name: Nidhi Mittal

```
open c1;
fetch c1 into vpro;
mob:=vpro.qnt;
fetch c1 into vpro;
lap:=vpro.qnt;
fetch c1 into vpro;
ip:=vpro.qnt;
if(mob<n1) then
    raise out_of_stock;
else
    update pro set qnt=qnt-n1 where prod='mobile';
end if;
if(lap<n2) then
    raise out_of_stock;
else
    update pro set qnt=qnt-n2 where prod='laptop';
end if;
if(ip<n3) then
    raise out_of_stock;
else
    update pro set qnt=qnt-n3 where prod='ipad';
end if;
exception
when out_of_stock then
    if(mob<n1 and lap<n2 and ip<n3) then
        dbms_output.put_line('all out of stock');
```

Roll no. : 1610991565

Name: Nidhi Mittal

```
elseif (mob<n1 and lap<n2) then
    dbms_output.put_line('mobile,laptop out of stock');

elseif(mob<n1 and ip<n3) then
    dbms_output.put_line('mobile, ipad out of stock');
elseif(lap<n2 and ip<n3) then
    dbms_output.put_line('laptop, ipad out of stock');
elseif(mob<n1) then
    dbms_output.put_line('mobile out of stock');
elseif(lap<n2) then
    dbms_output.put_line('laptop out of stock');
else
    dbms_output.put_line('ipad out of stock');
end if;
end;
```

OUTPUT:

Results Explain Describe Sa

mobile out of stock

1 row(s) updated.

0.15 seconds

EXPERIMENT NO.: 10 – 11

AIM: Creating and calling a standalone function. The function should return the total number of CUSTOMERS in the customers table. Use the CUSTOMERS table, with different columns like name, salary, department, designation, DOJ etc.

Functions:

A standalone function is created using the **CREATE FUNCTION** statement. The simplified syntax for the **CREATE OR REPLACE PROCEDURE** statement is as follows –

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

Where,

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

Calling a Function:

While creating a function, you give a definition of what the function has to do. To use a function, you will have to call that function to perform the defined task. When a program calls a function, the program control is transferred to the called function.

A called function performs the defined task and when its return statement is executed or when the **last end statement** is reached, it returns the program control back to the main program.

EXAMPLE- Program to add and subtract two numbers using functions

Declare

a number;

b number;

c number;

d number;

Function add(x in number , y in number) return number is z number;

Begin

c:=x+y;

return c;

end;

Function subtract(x in number , y in number) return number is z number;

Begin

d:=x-y;

return d;

end;

Begin

a:=30;

b:=20;

c:=add(a,b);

d:=subtract(a,b);

Dbms_output.put_line(c);

Dbms_output.put_line(d);

end;

Roll no. : 1610991565

Name: Nidhi Mittal

OUTPUT:

Results	Explain	Describe	Saved SQL
50			
10			
Statement processed.			
0.00 seconds			

EXAMPLE- Program to print the total count of employees in an office. Create function counting to return number is total_count number.

Begin

select count(*) into total_count from office;

return total_count;

end;

declare

counter number;

begin

counter:=counting2();

Dbms_output.put_line(counter);

end;

Output:

Results	Explain	Describe	Saved SQ
21			
Statement processed.			
0.05 seconds			