Exception handling

In PL/SQL , a warning or error condition is called an exception.

Exceptions can be internally defined i.e predefined (by the run time system) or user defined.

Examples of internally defined exceptions include division by zero ,Duplicate Value On Index etc.

User-defined exceptions are raised to handle user-specified conditions.

Note: Unlike predefined exceptions, user defined exceptions must be explicitly raised.

Exception handling

When an error occurs, an exception is raised i.e normal execution stops and control transfers to the exception-handling part of PL/SQL block ,and if present, control searches for appropriate exception-handler, and if present, error gets trapped within it. If the pl/sql block has no exception block or no appropriate exception handler, then error propagates to the calling environment.

Internal (Predefined) exceptions are raised implicitly (automatically) where as user- defined exceptions must be raised explicitly using the RAISE statement.

Exception handling

The reason for explicitly handling the exceptions is to trap the errors within the PL/SQL block & to take corrective measures so that the error doesn’t propagate outside the PL/SQL block.

Ex:

```
begin
        --------------;
        --------------;
exception
   when exception_handler  then
                ---------------;
   when exception_handler  then
                ---------------;

end;
```

Internal exceptions (Predefined exceptions)

An internal exception is raised implicitly whenever the PL/SQL program violates an oracle rule or exceeds system dependent limit.

There are 2 types of predefined exceptions:

1. Named predefined exceptions
2. Unnamed predefined exceptions

Predefined exceptions

1. Named Predefined exceptions:

Every oracle error has a number such as *ORA-01403: No Data Found* , but exceptions must be handled by names as given below:

```
EXCEPTION
        When  exception_name Then
                ----------------;
```

So oracle has given names to the some of the commonly occurring error numbers so that they can be handled explicitly. These are called as named predefined exceptions.
Ex:NO_DATA_FOUND

Named Predefined exceptions

```
Declare
        emp_rec emp%rowtype;
Begin
        select * into emp_rec from emp where empno= &eno;
        dbms_output.put_line (emp_rec.empno|| ',' ||
                                emp_rec.ename || ',' ||
                                emp_rec.deptno);
Exception
        When no_data_found then
                dbms_output.put_line ('Invalid employee number');
End;
/
```

Named Predefined exceptions

```
Declare
        emp_rec emp%rowtype;
Begin
        select * into emp_rec from emp where deptno= &dno;
        dbms_output.put_line (emp_rec.empno|| ',' ||
                                emp_rec.ename || ',' ||
                                emp_rec.deptno);
Exception
        When no_data_found then
          dbms_output.put_line ('Invalid department number');

        When too_many_rows then
          dbms_output.put_line ('More than 1 employee in the dept. ');
End;
```

Note: PL/SQL declares predefined exceptions globally in a package STANDARD.

Unnamed Predefined exceptions

There are two ways of handling unnamed predefined exceptions

Method I: Using a pragma called EXCEPTION_INIT.

Note : Pragmas are pseudo instructions that are processed
at compile time, but not at run time.

In PL/SQL , the pragma, exception_init informs the compiler to associate an exception name with an oracle error number. This allows us to refer an unnamed predefined exception by name .

Syntax:

```
pragma  exception_init(exception_name , oracle_error_number);
```

Note: The above syntax has to be specified in the declarative part of PL/SQL block.

Unnamed Predefined exceptions

Ex.

```
declare
  child_found exception;
  pragma exception_init(child_found,-2292);
begin
  delete from  dept where deptno = &dno;
  dbms_output.put_line ('department deleted');
exception
  when child_found then
   dbms_output.put_line ('invalid department number');
end;
```

Unnamed Predefined exceptions

Method II: Using OTHERS handler

Note: To trap any exception that is not explicitly handled in the PL/SQL block , use OTHERS handler.

```
declare
           emp_rec emp%rowtype;
           v_total  emp.sal%type;
 begin
           select * into emp_rec from emp
           where empno =&eno;
           v_total := nvl(emp_rec.sal,0) + nvl(emp_rec.comm ,0);
           dbms_output.put_line ('Total pay : ' || v_total);
 exception
           when others then
                        dbms_output.put_line ('invalid employee number');
  end;
```

OTHERS Handler

```
declare
   child_found exception;
   pragma exception_init(child_found,-2292);
 begin
   delete from  dept where deptno = &dno;
   dbms_output.put_line ('department deleted');
 exception
   when others then
    dbms_output.put_line ('invalid department number');
 end;
```

Note: OTHERS handler guarantees that no exception will go unhandled.

Note: The OTHERS exception handler should be placed as the last handler.

OTHERS Handler

By calling SQLCODE & SQLERRM functions within the OTHERS handler, we can find out the type of exception that is raised.

```
declare
      dept_rec dept%rowtype;
      v_errno number;
      v_errtxt varchar2(50);
begin
      select * into dept_rec from dept where deptno=&dno;
      dbms_output.put_line (dept_rec.deptno || ',' || dept_rec.dname);
exception
      when others then
                  v_errno := sqlcode;
                  v_errtxt :=  substr(sqlerrm,1,50);
      dbms_output.put_line (v_errno  || ' , '  || v_errtxt);
end;
/
```

SQLCODE & SQLERRM functions

```
create table ora_errors_tab
        ( error_no number(10),
          error_text varchar2(100)
        );
```

```
declare
        v_errtext varchar2(100);
begin
        for i in 1..1000
        loop
                 v_errtext := substr(sqlerrm(-i),1,100);
                insert into ora_errors_tab values( -i , v_errtext);
        end loop;
        commit;
end;
```

User-defined exceptions

User-defined exceptions are raised explicitly within pl/sql block.

Steps:

1) declare user-defined exception within the declarative part of the pl/sql block.

Syntax:

```
exception_name  exception;
```

User-defined exceptions

2) raise the exception within the pl/sql block based on some condition.

Syntax:

```
if cond then
        raise exception_name;
end if;
```

3) handle the exception within the exception clause.

Syntax:

```
Exception
        when exception_name then
        -------------;
```

About Exceptions

Note:
If an exception raised (implicitly/explicitly) within the pl/sql block is not handled, then DML statements executed within the pl/sql block will be rolled back.

If an exception raised (implicitly/explicitly) within the pl/sql block is trapped then DML statements executed within the pl/sql block will not be rolled back.

User-defined exceptions

Write a anonymous pl/sql block that updates the salary of given empno by user-specified amount. If employee’s new salary exceeds 50000, roll back the transaction using the user-defined exceptions .

```
Declare
        v_sal emp.sal%type;
        v_empno emp.empno%type:=&enum;
        v_amount real :=  &amount;
        salary_exceeded  EXCEPTION;
```

User-defined exceptions

```
 Begin
                    select nvl(sal,0) into v_sal from emp
                     where empno =  v_empno;
                     v_sal := v_sal + v_amount;
                    update emp set sal = v_sal where empno= v_empno;
                     if v_sal > 50000 then
                       raise salary_exceeded;
                    end if;
                    commit;
 Exception
       when salary_exceeded then
            dbms_output.put_line ('salary cannot exceed Rs.50000');
            rollback;
          when others then
            null;
End;
```

User-defined exceptions

Propagation of exceptions in nested pl/sql blocks

When an exception is raised , if pl/sql cannot find an handler in the current block , the exception propagates. i.e. the exception reproduces itself in successive enclosing blocks until a handler is found or there are no more blocks to search. In latter case, error propagates to the host environment.

Re-raising an exception

To re-raise an exception, place a RAISE statement in the local handler.

User-defined exceptions

Write anonymous pl/sql block that increment’s the commission of given empno by 10% of current salary. If employee’s new commission exceeds current salary then raise user-defined exception, else commit the changes.

```
Declare
   v_sal emp.sal%type;
   v_comm  emp.comm%type;
   v_empno   emp.empno%type:=&enum;
   comm_exceeded exception;
Begin
       Begin
                      select nvl(sal,0),comm into v_sal, v_comm from emp
                       where empno = v_empno;
                       v_comm := v_comm + .10 * v_sal ;
                      update emp set comm = v_comm where empno= v_empno;
                       if v_comm > v_sal then
                         raise comm_exceeded;
                      else
                          commit;
                      end if;
                   Exception
                      when comm_exceeded then
                         dbms_output.put_line ('Commission Cannot exceed Salary');
                         raise;
                      when others then
                         null;
       End;
```

User-defined exceptions

```
Exception
        when comm_exceeded  then
                rollback;
        when others  then
                null;
End;
```

Exceptions at declare & exception section of a PL/SQL Block

Exceptions that occur within the declarative part and within the exception section of a pl/sql block cannot be trapped within the same block.

They have to be trapped within the outer block.

```
Declare
 v_amount number(3) := 2500;
Begin
 update emp set sal = sal + v_amount
 where empno = &enum;
Exception
 when others then
    dbms_output.put_line ('Error trapped within the same block');
End;
```

```
ERROR at line 1:
ORA-06502: PL/SQL: numeric or value error: number precision too large
ORA-06512: at line 2
```

Exceptions at declare & exception section of a PL/SQL Block

```
Begin
 Declare
  v_amount number(3) := 2500;
 Begin
  update emp set sal = sal + v_amount
  where empno = &enum;
 Exception
  when others then
     dbms_output.put_line ('Error trapped within the same block');
 End;
 Exception
  when others then
     dbms_output.put_line ('Error trapped in the outer block');
 End;
```

```
Error trapped in the outer block

PL/SQL procedure successfully completed.
```

Exceptions at declare & exception section of a PL/SQL Block

```
SQL> execute :b_empno1 := 1111;
SQL> execute :b_empno2 := 1112;
SQL>    Declare
        v_sal emp.sal%type;
        Begin
        select sal into v_sal from emp
        where empno = :b_empno1;
        dbms_output. put_line(v_sal);
        Exception
         when no_data_found then
                dbms_output.put_line ('Invalid empnno');
                select sal into v_sal from emp
                where empno = :b_empno2;
                dbms_output.put_line(v_sal);
        when others then
                null;
        End;
```

```
Invalid empnno
Declare
*
ERROR at line 1:
ORA-01403: no data found
ORA-06512: at line 10
ORA-01403: no data found
```

Exceptions at declare & exception section of a PL/SQL Block

```
        Begin
        Declare
        v_sal emp.sal%type;
        Begin
         select sal into v_sal from emp
         where empno = :b_empno1;
         dbms_output. put_line(v_sal);
        Exception
         when no_data_found then
          dbms_output.put_line ('Invalid empnno');
          select sal into v_sal from emp
          where empno = :b_empno2;
          dbms_output.put_line(v_sal);
         when others then
          null;
        End;
        Exception
         When others then
                dbms_output.put_line ('Error trapped in outer block');
        End;
/
```

```
Invalid empnno
Error trapped in outer block

PL/SQL procedure successfully completed.
```