Exception Handling

In this section we will discuss about the following,   
1) What is Exception Handling.   
2) Structure of Exception Handling.   
3) Types of Exception Handling.

1) What is 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 a messages which explains its cause is recieved.   
PL/SQL Exception message consists of three parts.   
**1) Type of Exception  
2) An Error Code  
3) A message**

|  |  |
| --- | --- |
| By Handling the exceptions we can ensure a PL/SQL block does not exit abruptly. |  |

### 2) 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;*

#### General PL/SQL statments can be used in the Exception Block.

When an exception is raised, Oracle searches for an appropriate exception handler in the exception section. For example in the above example, if the error raised is 'ex\_name1 ', then the error is handled according to the statements under it. Since, it is not possible to determine all the possible runtime errors during testing fo the code, the 'WHEN Others' exception is used to manage the exceptions that are not explicitly handled. Only one exception can be raised in a Block and the control does not return to the Execution Section after the error is handled.

If there are nested PL/SQL blocks like this.

*DELCARE*

*Declaration section*

*BEGIN*

*DECLARE*

*Declaration section*

*BEGIN*

*Execution section*

*EXCEPTION*

*Exception section*

*END;*

*EXCEPTION*

*Exception section*

*END;*

In the above case, if the exception is raised in the inner block it should be handled in the exception block of the inner PL/SQL block else the control moves to the Exception block of the next upper PL/SQL Block. If none of the blocks handle the exception the program ends abruptly with an error.

### 3) Types of Exception.

There are 3 types of Exceptions.   
a) Named System Exceptions   
b) Unnamed System Exceptions   
c) User-defined Exceptions

### a) Named 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.

Named system exceptions are:   
1) Not Declared explicitly,   
2) Raised implicitly when a predefined Oracle error occurs,   
3) caught by referencing the standard name within an exception-handling routine.

|  |  |  |
| --- | --- | --- |
| **Exception Name** | **Reason** | **Error Number** |
| CURSOR\_ALREADY\_OPEN | When you open a cursor that is already open. | ORA-06511 |
| INVALID\_CURSOR | When you perform an invalid operation on a cursor like closing a cursor, fetch data from a cursor that is not opened. | ORA-01001 |
| NO\_DATA\_FOUND | When a SELECT...INTO clause does not return any row from a table. | ORA-01403 |
| TOO\_MANY\_ROWS | When you SELECT or fetch more than one row into a record or variable. | ORA-01422 |
| ZERO\_DIVIDE | When you attempt to divide a number by zero. | ORA-01476 |

**For Example:**Suppose a NO\_DATA\_FOUND exception is raised in a proc, we can write a code 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) Unnamed System Exceptions

Those system exception for which oracle does not provide a name is known as unamed system exception. These exception do not occur frequently. These Exceptions have a code and an associated message.

There are two ways to handle unnamed sysyem exceptions:   
1. By using the WHEN OTHERS exception handler, or   
2. By associating the exception code to a name and using it as a named exception.

We can assign a name to unnamed system exceptions using a **Pragma** called **EXCEPTION\_INIT.**   
**EXCEPTION\_INIT** will associate a predefined Oracle error number to a programmer\_defined exception name.

Steps to be followed to use unnamed system exceptions are   
• They are raised implicitly.   
• If they are not handled in WHEN Others they must be handled explicity.   
• To handle the exception explicity, they must be declared using Pragma EXCEPTION\_INIT as given above and handled referecing the user-defined exception name in the exception section.

The general syntax to declare unnamed system exception using EXCEPTION\_INIT is:

*DECLARE*

*exception\_name EXCEPTION;*

*PRAGMA*

*EXCEPTION\_INIT (exception\_name, Err\_code);*

*BEGIN*

*Execution section*

*EXCEPTION*

*WHEN exception\_name THEN*

*handle the exception*

*END;*

**For Example:** Lets consider the product table and order\_items table from sql joins.

Here product\_id is a primary key in product table and a foreign key in order\_items table.   
If we try to delete a product\_id from the product table when it has child records in order\_id table an exception will be thrown with oracle code number -2292.   
We can provide a name to this exception and handle it in the exception section as given below.

*DECLARE*

*Child\_rec\_exception EXCEPTION;*

*PRAGMA*

*EXCEPTION\_INIT (Child\_rec\_exception, -2292);*

*BEGIN*

*Delete FROM product where product\_id= 104;*

*EXCEPTION*

*WHEN Child\_rec\_exception*

*THEN Dbms\_output.put\_line('Child records are present for this product\_id.');*

*END;*

*/*

### c) User-defined Exceptions

Apart from sytem exceptions we can explicity define exceptions based on business rules. These are known as user-defined exceptions.

Steps to be followed to use user-defined exceptions:   
• They should be explicitly declared in the declaration section.   
• They should be explicitly raised in the Execution Section.   
• They should be handled by referencing the user-defined exception name in the exception section.

**For Example:** Lets consider the product table and order\_items table from sql joins to explain user-defined exception.   
Lets create a business rule that if the total no of units of any particular product sold is more than 20, then it is a huge quantity and a special discount should be provided.

*DECLARE*

*huge\_quantity EXCEPTION;*

*CURSOR product\_quantity is*

*SELECT p.product\_name as name, sum(o.total\_units) as units*

*FROM order\_tems o, product p*

*WHERE o.product\_id = p.product\_id;*

*quantity order\_tems.total\_units%type;*

*up\_limit CONSTANT order\_tems.total\_units%type := 20;*

*message VARCHAR2(50);*

*BEGIN*

*FOR product\_rec in product\_quantity LOOP*

*quantity := product\_rec.units;*

*IF quantity > up\_limit THEN*

*message := 'The number of units of product ' || product\_rec.name ||*

*' is more than 20. Special discounts should be provided.*

*Rest of the records are skipped. '*

*RAISE huge\_quantity;*

*ELSIF quantity < up\_limit THEN*

*v\_message:= 'The number of unit is below the discount limit.';*

*END IF;*

*dbms\_output.put\_line (message);*

*END LOOP;*

*EXCEPTION*

*WHEN huge\_quantity THEN*

*dbms\_output.put\_line (message);*

*END;*

*/*

### RAISE\_APPLICATION\_ERROR ( )

**RAISE\_APPLICATION\_ERROR** is a built-in procedure in oracle which is used to display the user-defined error messages along with the error number whose range is in between -20000 and -20999.

Whenever a message is displayed using RAISE\_APPLICATION\_ERROR, all previous transactions which are not committed within the PL/SQL Block are rolled back automatically (i.e. change due to INSERT, UPDATE, or DELETE statements).

RAISE\_APPLICATION\_ERROR raises an exception but does not handle it.

RAISE\_APPLICATION\_ERROR is used for the following reasons,   
a) to create a unique id for an user-defined exception.   
b) to make the user-defined exception look like an Oracle error.

The General Syntax to use this procedure is:

***RAISE\_APPLICATION\_ERROR (error\_number, error\_message);***

• The Error number must be between -20000 and -20999   
• The Error\_message is the message you want to display when the error occurs.

Steps to be folowed to use RAISE\_APPLICATION\_ERROR procedure:   
1. Declare a user-defined exception in the declaration section.   
2. Raise the user-defined exception based on a specific business rule in the execution section.  
3. Finally, catch the exception and link the exception to a user-defined error number in RAISE\_APPLICATION\_ERROR.

Using the above example we can display a error message using RAISE\_APPLICATION\_ERROR.

*DECLARE*

*huge\_quantity EXCEPTION;*

*CURSOR product\_quantity is*

*SELECT p.product\_name as name, sum(o.total\_units) as units*

*FROM order\_tems o, product p*

*WHERE o.product\_id = p.product\_id;*

*quantity order\_tems.total\_units%type;*

*up\_limit CONSTANT order\_tems.total\_units%type := 20;*

*message VARCHAR2(50);*

*BEGIN*

*FOR product\_rec in product\_quantity LOOP*

*quantity := product\_rec.units;*

*IF quantity > up\_limit THEN*

*RAISE huge\_quantity;*

*ELSIF quantity < up\_limit THEN*

*v\_message:= 'The number of unit is below the discount limit.';*

*END IF;*

*Dbms\_output.put\_line (message);*

*END LOOP;*

*EXCEPTION*

*WHEN huge\_quantity THEN*

*raise\_application\_error(-2100, 'The number of unit is above the discount limit.');*

*END;*

*/*