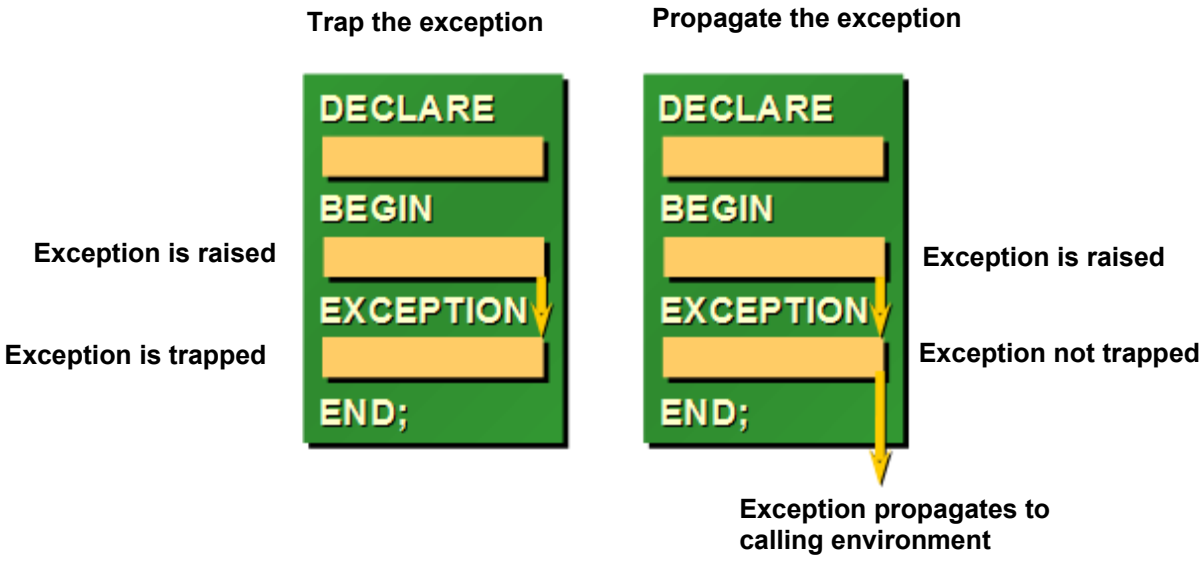
Oracle PL/SQL – Exception Handling

codingcircuit23@gmail.com

Introduction

- In PL/SQL, a warning or error condition is called an **exception**.
- To handle exceptions, separate routines or code can be developed which are called as **exception handler**.
- The exception handler mechanism allows to clearly separate error processing code from executable statements
- When an exception occurs or is raised, the normal execution stops and the control transfers to the exception handling part of the PL/SQL block.
- Exception are declared in the declarative section, raised in the executable section and handled in the exception section.



- PL/SQL exception consists of three parts :
  - Exception Type
  - Error Code
  - Error Message

Advantages of PL/SQL Exception handling:

- When exception handling is used, there is no need to check for execution errors each time a code is executed. Instead, the errors would be automatically handled. Thus it improves the reliability.
- Isolating error handling routines makes the rest of the program easier to read and understand.
- Potential errors from many statements can be handled with a single exception handler.
- It helps in identifying the errors in a easier way , especially errors which could be detected only testing with bad data.

Syntax :

```
DECLARE
        <declarations section>
BEGIN
        <executable command(s)>
EXCEPTION
        <exception handling here >
         WHEN exception1 THEN
                exception1-handling-statements
         WHEN exception2  THEN
                exception2-handling-statements
```

```
        WHEN exception3 THEN
                exception3-handling-statements
        ........
        WHEN others THEN
                exception3-handling-statements
END;
```

Types Of Exceptions

Exception are mainly of two types

- System defined Exception
- User defined Exception

System defined Exception :

- It is also referred as **internal** exception or **predefined** exception.
- A predefined exception is raised automatically when the PL/SQL program violates an Oracle rule or exceed a system dependent limit.
- Predefined Exception can be
    - Named Exception
    - Unnamed Exception

Named Exception:

- System exception which has a predefined name in Oracle is called Named System Exception.
- NO_DATA_FOUND and ZERO_DIVIDE are examples for named system exception.
- Named system exceptions are
    - declared implicitly. Explicit declaration not required.
    - raised implicitly when a predefined Oracle error occurs
    - caught by referencing the standard name within an exception

handling routine.

Below table shows some of the commonly used named system exceptions

Exception Name	Error Number	Reason
CURSOR_ALREADY_OPEN	ORA-06511	A program tries to open a cursor which is already opened.
INVALID_CURSOR	ORA-01001	A program attempts a cursor operation that is not allowed, such as closing or fetching an unopened cursor.
NO_DATA_FOUND	ORA-01403	When a SELECT...INTO clause does not return any row from a table.
TOO_MANY_ROWS	ORA-01422	When more than one row is fetched or selected into a record or variable
ZERO_DIVIDE	ORA-01476	A program attends to divide a number by zero.
STORAGE_ERROR	ORA-06500	PL/SQL runs out of memory or memory has been corrupted.

Note :

To handle unexpected Oracle errors, OTHERS handler can be used. Within this handler, **SQLCODE** and **SQLERM** are used to return Oracle error code and message text.

Example:

```
SET SERVEROUTPUT ON;
DECLARE
       v_emp_name employee.emp_name%TYPE;
       v_designation employee.designation%TYPE;
BEGIN
       SELECT emp_name,designation INTO v_emp_name,v_designation
                                FROM employee WHERE emp_id =102;
       dbms_output.put_line('Employee Name :'||v_emp_name);
       dbms_output.put_line('Designation :'||v_designation);
EXCEPTION
         WHEN no_data_found THEN
              dbms_output.put_line('No customer selected');
         WHEN OTHERS THEN
              dbms_output.put_line('Error :'|| SQLCODE);
END;
/
```

Output :

It returns the below output since there was no employee record with ID =102 and thus no_data_found exception was raised and handled.

```
No customer selected

PL/SQL procedure successfully completed
```

Unnamed System Exception:

- System exception for which Oracle does not provide a name is known as unnamed system exception.
- These exception do not occur frequently. These exceptions have a code and an associated message.
- There are two ways to handle unnamed system exceptions
  - By using the WHEN OTHERS exception handler.
  - By associating the exception code to a name and using it as a

named exception.

- A name can be assigned to the unnamed system exceptions using a PRAGMA (compiler directive) called EXCEPTION_INIT.

- EXCEPTION_INIT will tell the compiler to associate an exception name with an Oracle predefined error number.

- Steps to be followed to use unnamed system exceptions are

  - They are raised implicitly. Explicit raising is not required.
  - If they are not handled in WHEN OTHERS they must be handled explicitly.
  - To handle the exception explicitly, they must be declared using EXCEPTION_INIT and handled using the user defined exception name

Syntax:

```
DECLARE
        <exception_name> EXCEPTION;
        PRAGMA
        EXCEPTION_INIT(exception_name,Err_code);
BEGIN
        <executable section>
EXCEPTION
         WHEN exception_name THEN
                 <exception handling statements>
END;
```

Example :

Consider a product table and order_items table where the product_id is a primary key in product table and a foreign key in order_items table. On deleting the 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. A name can be provided to this exception and can be handled as shown 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;
/
```

User- defined Exception

- Apart from system exceptions users can explicitly 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.

Syntax for declaring an exception :

```
DECLARE
 <exception_name> EXCEPTION;
```

Example for user defined exception

In the below example, designation and employee ID values are got from the user at run time. If there is no employee record with the given ID then the user defined exception will be raised and control passes to the exception handler.

```
SET SERVEROUTPUT ON ;
DECLARE
e_invalid_employee  EXCEPTION; -- declare the exception
BEGIN
   UPDATE employee SET designation = '&designation'  WHERE emp_id = &employee_number;
   IF SQL%NOTFOUND THEN
            RAISE e_invalid_employee; -- to raise the user defined exception
   END IF;
   COMMIT;
EXCEPTION
    WHEN e_invalid_employee THEN -- to handle the exception
    DBMS_OUTPUT.PUT_LINE('Invalid employee number.');
END;
/
```

Output :

```
Invalid employee number.

PL/SQL procedure successfully completed
```

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
- RAISE_APPLICATION_ERROR raises an exception but does not handle it.
- RAISE_APPLICATION_ERROR is used for the following reasons.
    - To create a unique id for an user-defined exception.
    - To make the user-defined exception look like an Oracle error.

Syntax:

```
RAISE_APPLICATION_ERROR(error_number, error_message);
```

Example:

```
SET SERVEROUTPUT ON ;
DECLARE
empid NUMBER := &eno;
BEGIN
IF empid <=0 THEN
raise_application_error (-20100,'Employee number must be> 0');
ELSE
DELETE FROM employee WHERE emp_id =empid;
END IF;
END;
/
```