

# Oracle for Developers (PL/SQL)

## Exception Handling



To understand the following topics:

- Error Handling
  - Declaring Exceptions
    - Predefined Exceptions
    - Non-Predefined Exceptions
    - User Defined Exceptions
  - Raising Exceptions
  - Control passing to Exception Handlers
  - RAISE\_APPLICATION\_ERROR
- } Implicitly Raised Exception  
Explicitly Raised Exception





## Error Handling:

- In PL/SQL, a warning or error condition is called an “exception”.
  - Exceptions can be internally defined (by the run-time system) or user defined.
  - Examples of internally defined exceptions:
    - division by zero
    - out of memory
  - Some common internal exceptions have predefined names, namely:
    - ZERO\_DIVIDE
    - STORAGE\_ERROR
  - The other exceptions can be given user-defined names.
  - Exceptions can be defined in the declarative part of any PL/SQL block, subprogram, or package. These are Non-Predefined or user-defined exceptions.



Exception is an error that is defined by the program.

- It could be an error with the data, as well.

There are three types of exceptions in Oracle:

- |   |   |                             |
|---|---|-----------------------------|
| <ul style="list-style-type: none"><li>▪ Predefined exceptions</li><li>▪ Non-Predefined exceptions</li></ul> | } | Implicitly Raised Exception |
| <ul style="list-style-type: none"><li>▪ User defined exceptions</li></ul>                                   |   | Explicitly Raised Exception |



Predefined Exceptions correspond to the most common Oracle errors.

- They are always available to the program. Hence there is no need to declare them.
- They are automatically raised by ORACLE whenever that particular error condition occurs.
- Examples: NO\_DATA\_FOUND, CURSOR\_ALREADY\_OPEN, PROGRAM\_ERROR



Exception	Oracle Error
ACCESS_INTO_NULL	ORA-06530
CASE_NOT_FOUND	ORA-06592
COLLECTION_IS_NULL	ORA-06531
CURSOR_ALREADY_OPEN	ORA-06511
DUP_VAL_ON_INDEX	ORA-00001
INVALID_CURSOR	ORA-01001
INVALID_NUMBER	ORA-01722
LOGIN_DENIED	ORA-01017
NO_DATA_FOUND	ORA-01403
NOT_LOGGED_ON	ORA-01012
PROGRAM_ERROR	ORA-06501
ROWTYPE_MISMATCH	ORA-06504
SELF_IS_NULL	ORA-30625
STORAGE_ERROR	ORA-06500
SUBSCRIPT_BEYOND_COUNT	ORA-06533
SUBSCRIPT_OUTSIDE_LIMIT	ORA-06532
SYS_INVALID_ROWID	ORA-01410
TIMEOUT_ON_RESOURCE	ORA-00051
TOO_MANY_ROWS	ORA-01422
VALUE_ERROR	ORA-06502
ZERO_DIVIDE	ORA-01476



- In the following example, the built in exception is handled

```
DECLARE
    v_staffno staff_master.staff_code%type;
    v_name staff_master.staff_name%type;
BEGIN
    SELECT staff_name into v_name FROM staff_master
    WHERE staff_code=&v_staffno;
    dbms_output.put_line(v_name);
EXCEPTION
    WHEN NO_DATA_FOUND THEN
        dbms_output.put_line('Not Found');
END;
/
```



An exception name can be associated with an ORACLE error.

- This gives us the ability to trap the error specifically to ORACLE errors
- This is done with the help of “compiler directives” –  
PRAGMA EXCEPTION\_INIT





## PRAGMA EXCEPTION\_INIT:

- A PRAGMA is a compiler directive that is processed at compile time, not at run time. It is used to name an exception.
- In PL/SQL, the PRAGMA EXCEPTION\_INIT tells the compiler to associate an exception name with an Oracle error number.
  - This arrangement lets you refer to any internal exception(error) by name, and to write a specific handler for it.
- When you see an error stack, or sequence of error messages, the one on top is the one that you can trap and handle.



- The exception is declared in Declaration section.
- It is valid within the PL/SQL blocks only.
- Syntax is:

```
PRAGMA EXCEPTION_INIT(Exception Name, Error_Number);
```



A PL/SQL block to handle Numbered Exceptions

```
DECLARE
    v_bookno number := 10000008;
    child_rec_found EXCEPTION;
    PRAGMA EXCEPTION_INIT (child_rec_found, -2292);
BEGIN
    DELETE from book_master
    WHERE book_code = v_bookno;
EXCEPTION
    WHEN child_rec_found THEN
        INSERT into error_log
        VALUES ('Book entries exist for book:' || v_bookno);
END;
```



User-defined Exceptions are:

- declared in the Declaration section,
- raised in the Executable section with RAISE command, and
- handled in the Exception section



Here is an example of User Defined Exception:

```
DECLARE
    E_Balance_Not_Sufficient EXCEPTION;
    E_Comm_Too_Large EXCEPTION;
    ...
BEGIN
    NULL;
END;
```



## Raising Exceptions:

- Internal exceptions are raised implicitly by the run-time system, as are user-defined exceptions that are associated with an Oracle error number using `EXCEPTION_INIT`.
- Other user-defined exceptions must be raised explicitly by `RAISE` statements.
  - The syntax is:

```
RAISE Exception_Name;
```



An exception is defined and raised as shown below:

```
DECLARE
    ...
    retired_emp EXCEPTION ;
BEGIN
    pl/sql_statements ;
    if error_condition then
        RAISE retired_emp ;
    pl/sql_statements ;
EXCEPTION
    WHEN retired_emp THEN
        pl/sql_statements ;
END ;
```



## User Defined Exception Handling:

```
DECLARE
    dup_deptno EXCEPTION;
    v_counter binary_integer;
    v_department number(2) := 50;
BEGIN
    SELECT count(*) into v_counter FROM department_master
    WHERE dept_code=50;
    IF v_counter > 0 THEN
        RAISE dup_deptno ;
    END IF;
    INSERT into department_master
    VALUES (v_department , 'new name');
EXCEPTION
    WHEN dup_deptno THEN
        INSERT into error_log
        VALUES ('Dept: ' || v_department || ' already exists');
END ;
```





## OTHERS Exception Handler:

- The optional OTHERS exception handler, which is always the last handler in a block or subprogram, acts as the handler for all exceptions that are not specifically named in the Exception section.
- A block or subprogram can have only one OTHERS handler.



- To handle a specific case within the OTHERS handler, predefined functions SQLCODE and SQLERRM are used.
  - SQLCODE returns the current error code. And SQLERRM returns the current error message text.
  - The values of SQLCODE and SQLERRM should be assigned to local variables before using it within a SQL statement.



```
DECLARE
  v_dummy varchar2(1);
  v_designation number(3) := 109;
BEGIN
  SELECT 'x' into v_dummy FROM designation_master
  WHERE design_code= v_designation;
  INSERT into error_log
  VALUES ('Designation: ' || v_designation || 'already exists');
EXCEPTION
  WHEN no_data_found THEN
    insert into designation_master values
(v_designation,'newdesig');
  WHEN OTHERS THEN
    Err_Num := SQLCODE;
    Err_Msg :=SUBSTR( SQLERRM, 1, 100);
    INSERT into errors VALUES( err_num, err_msg );
END;
```



## RAISE\_APPLICATION\_ERROR:

- The procedure RAISE\_APPLICATION\_ERROR lets you issue user-defined ORA- error messages from stored subprograms.
- In this way, you can report errors to your application and avoid returning unhandled exceptions.
- RAISE\_APPLICATION\_ERROR exceptions can be named with error number between -20000 and -20999.
- Syntax:

```
RAISE_APPLICATION_ERROR( Error_Number, Error_Message);
```

- where:
  - Error\_Number is a parameter between -20000 and -20999
  - Error\_Message is the text associated with this error



Here is an example of Raise Application Error:

```
DECLARE
    /* VARIABLES */
BEGIN
    .....
    .....
EXCEPTION
    WHEN OTHERS THEN
        -- Will transfer the error to the calling environment
        RAISE_APPLICATION_ERROR( -20999, 'Contact          DBA');
END;
```



In this lesson, you have learnt about:

- Exception Handling
  - User-defined Exceptions
  - Predefined Exceptions
- Control passing to Exception Handler
- OTHERS exception handler
- Association of Exception name to Oracle errors
- RAISE\_APPLICATION\_ERROR procedure





Question 1: The procedure \_\_\_\_ lets you issue user-defined ORA-error messages from stored subprograms.

Question 2: The \_\_\_\_ tells the compiler to associate an exception name with an Oracle error number.

Question 3: \_\_\_\_ returns the current error code. And \_\_\_\_ returns the current error message text.

