
ORACLE®
ACADEMY

Database Programming with PL/SQL

7-3
Trapping User-Defined Exceptions

ORACLE ACADEMY

Objectives

This lesson covers the following objectives:

- Write PL/SQL code to name a user-defined exception
- Write PL/SQL code to raise an exception
- Write PL/SQL code to handle a raised exception
- Write PL/SQL code to use `RAISE_APPLICATION_ERROR`

Purpose

- In addition to the predefined Oracle errors, programmers can create their own user-defined errors.
- User-defined errors are not automatically raised by the Oracle server, but are defined by the programmer and must be raised by the programmer when they occur.
- With a user-defined error, the programmer creates an error code and an error message.
- An example of a user-defined error might be `INVALID_MANAGER_ID`.

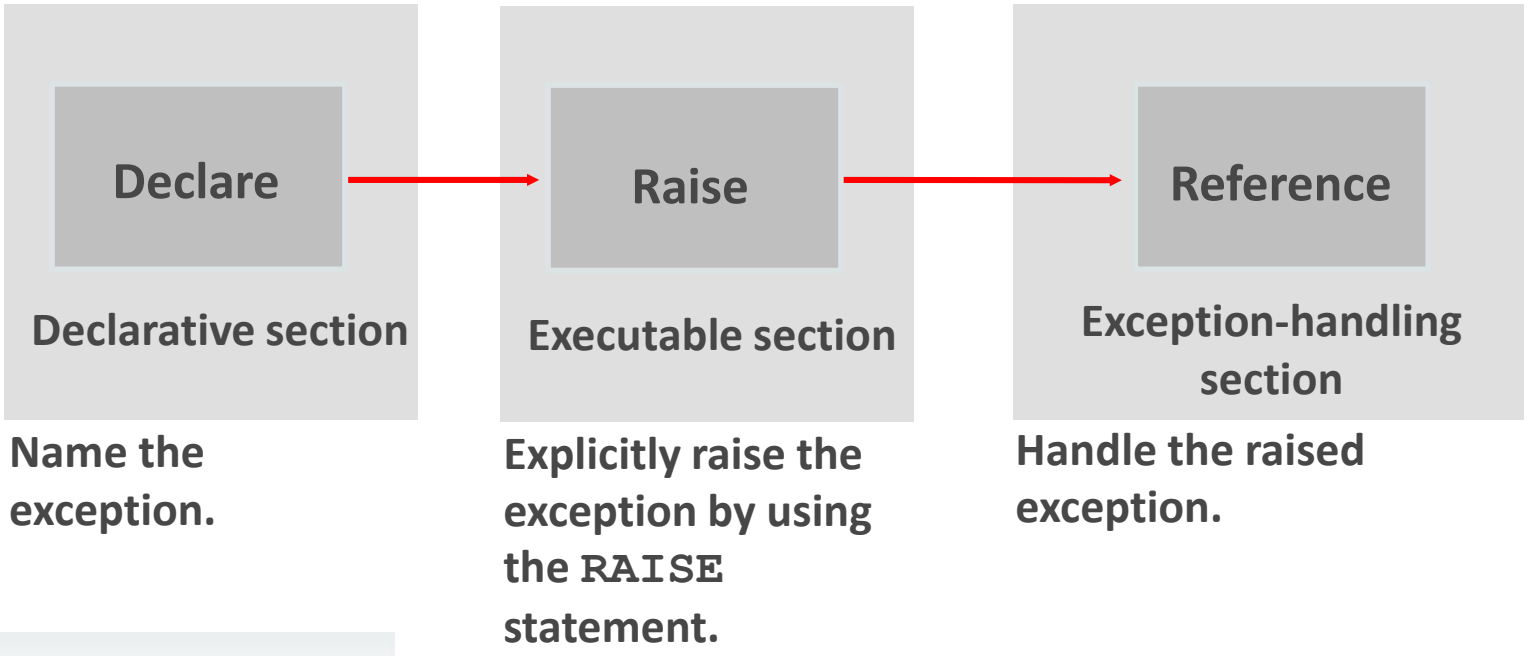
Exception Types

This lesson discusses user-defined errors.

Exception	Description	Instructions for Handling
Predefined Oracle server error	Most common PL/SQL errors (about 20 or so that are named)	You need not declare these exceptions. They are predefined by the Oracle server and are raised implicitly (automatically).
Non-predefined Oracle server error	Other PL/SQL errors (no name)	Declare within the declarative section and allow the Oracle Server to raise them implicitly (automatically).
User-defined error	Defined by the programmer	Declare within the declarative section, and raise explicitly.

Trapping User-Defined Exceptions

- PL/SQL allows you to define your own exceptions.
- You define exceptions depending on the requirements of your application.



Trapping User-Defined Exceptions

- One example of the need for a user-defined exception is during the input of data.
- Assume your program prompts the user for a department number and name so it can update the name of the department.

```
DECLARE
  v_name    VARCHAR2(20):= 'Accounting';
  v_deptno NUMBER := 27;
BEGIN
  UPDATE  departments
    SET     department_name = v_name
    WHERE   department_id = v_deptno;
END;
```

Trapping User-Defined Exceptions

- What happens if the user enters an invalid department number?
- Oracle doesn't see this as an error.
- You will need a user-defined error to catch this situation.

```
DECLARE
  v_name   VARCHAR2(20):= 'Accounting';
  v_deptno NUMBER := 27;
BEGIN
  UPDATE  departments
    SET     department_name = v_name
    WHERE   department_id = v_deptno;
END;
```

Trapping User-Defined Exceptions

- What happens when the user enters an invalid department?
- The code as written doesn't produce an Oracle error.
- You need to create a user-defined error to handle this situation.
- You do this by:
  1. Declaring the name of the user-defined exception within the declarative section.

```
e_invalid_department EXCEPTION;
```

  2. Using the `RAISE` statement to raise the exception explicitly within the executable section.

```
IF SQL%NOTFOUND THEN RAISE e_invalid_department;
```

Trapping User-Defined Exceptions

- You do this by:
  3. Referencing the declared exception name within a `WHEN` clause in the exception-handling section.

```
EXCEPTION
  WHEN e_invalid_department  THEN
    DBMS_OUTPUT.PUT_LINE('No such department id.');
```

- These three "steps" are similar to what we did in the previous lesson with non-predefined Oracle errors.
- The differences are, no `PRAGMA EXCEPTION_INIT` is required and you must explicitly raise the exception using the `RAISE` command.

Trapping User-Defined Exceptions

The completed code with the "steps" indicated.

```
DECLARE
  e_invalid_department EXCEPTION;
  v_name VARCHAR2(20):='Accounting';
  v_deptno NUMBER := 27;
BEGIN
  UPDATE  departments
    SET     department_name = v_name
    WHERE   department_id = v_deptno;
  IF SQL%NOTFOUND THEN
    RAISE e_invalid_department;
  END IF;
EXCEPTION
  WHEN e_invalid_department
    THEN DBMS_OUTPUT.PUT_LINE('No such department id.');
END;
```

1

2

3

The RAISE Statement

- You can use the RAISE statement to raise exceptions.
- Raising a user-defined exception:

```
IF v_grand_total = 0 THEN
   RAISE e_invalid_total;
ELSE
  DBMS_OUTPUT.PUT_LINE(v_num_students / v_grand_total);
END IF;
```

- Raising an Oracle server error:

```
IF v_grand_total = 0 THEN
   RAISE ZERO_DIVIDE;
ELSE
   DBMS_OUTPUT.PUT_LINE(v_num_students / v_grand_total);
END IF;
```

The RAISE_APPLICATION_ERROR Procedure

- You can use the RAISE_APPLICATION_ERROR procedure to return user-defined error messages from stored subprograms.
- The following slides explain the syntax for using RAISE_APPLICATION_ERROR
- The main advantage of using this procedure instead of RAISE, is that RAISE_APPLICATION_ERROR allows you to associate your own error number and meaningful message with the exception.

The RAISE_APPLICATION_ERROR Syntax

- The *error_number* must fall between -20000 and -20999.
- This range is reserved by Oracle for programmer use, and is never used for predefined Oracle server errors.
- *message* is the user-specified message for the exception.
- It is a character string up to 2,048 bytes long.

```
RAISE_APPLICATION_ERROR (error_number,
                         message[, {TRUE | FALSE}]);
```

The RAISE_APPLICATION_ERROR Syntax

- TRUE | FALSE is an optional Boolean parameter.
- If TRUE, the error is placed on the stack of previous errors.
- If FALSE—the default—the error replaces all previous errors.

```
RAISE_APPLICATION_ERROR (error_number,
                         message[, {TRUE | FALSE}]);
```

The RAISE_APPLICATION_ERROR Usage

You can use the RAISE_APPLICATION_ERROR in two different places:

- Executable section
- Exception section

RAISE_APPLICATION_ERROR in the Executable Section

- When called, the RAISE_APPLICATION_ERROR procedure displays the error number and message to the user.
- This process is consistent with other Oracle server errors.

```
DECLARE
  v_mgr         PLS_INTEGER := 123;
BEGIN
  DELETE FROM employees
     WHERE  manager_id = v_mgr;
  IF SQL%NOTFOUND THEN
     RAISE_APPLICATION_ERROR(-20202,
  'This is not a valid manager');
  END IF;
END;
```

RAISE_APPLICATION_ERROR in the Exception Section

```
DECLARE
  v_mgr         PLS_INTEGER := 27;
  v_employee_id       employees.employee_id%TYPE;
BEGIN
  SELECT employee_id INTO v_employee_id
    FROM employees
    WHERE manager_id = v_mgr;
  DBMS_OUTPUT.PUT_LINE('Employee #' || v_employee_id ||
       ' works for manager #' ||     v_mgr || '.');
EXCEPTION
   WHEN NO_DATA_FOUND THEN
     RAISE_APPLICATION_ERROR(-20201,
   'This manager has no employees');
   WHEN TOO_MANY_ROWS THEN
     RAISE_APPLICATION_ERROR(-20202,
   'Too many employees were found.');
END;
```

Using the RAISE_APPLICATION_ERROR with a User-Defined Exception

```
DECLARE
  e_name EXCEPTION;
  PRAGMA EXCEPTION_INIT(e_name, -20999);
  v_last_name      employees.last_name%TYPE := 'Silly Name';
BEGIN
  DELETE FROM employees WHERE last_name = v_last_name;
  IF SQL%ROWCOUNT = 0 THEN
    RAISE_APPLICATION_ERROR(-20999, 'Invalid last name');
  ELSE
    DBMS_OUTPUT.PUT_LINE(v_last_name ||' deleted');
  END IF;
EXCEPTION
  WHEN e_name THEN
    DBMS_OUTPUT.PUT_LINE('Valid last names are: ');
    FOR c1 IN (SELECT DISTINCT last_name FROM employees)
 LOOP
         DBMS_OUTPUT.PUT_LINE(c1.last_name);
 END LOOP;
  WHEN OTHERS THEN
    DBMS_OUTPUT.PUT_LINE('Error deleting from employees');
END;
```

Terminology

Key terms used in this lesson included:

- `RAISE`
- `RAISE_APPLICATION_ERROR`
- User-defined error

Summary

In this lesson, you should have learned how to:

- Write PL/SQL code to name a user-defined exception
- Write PL/SQL code to raise an exception
- Write PL/SQL code to handle a raised exception
- Write PL/SQL code to use `RAISE_APPLICATION_ERROR`


ORACLE®
ACADEMY