Database Programming with PL/SQL

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

Another kind of error handled by PL/SQL is a user-defined error.

These errors are not automatically raised by the Oracle server, but are defined by the programmer and are specific to the programmer's code.

An example of a programmer-defined error is `INVALID_MANAGER_ID`. You can define both an error code and an error message for user-defined errors.

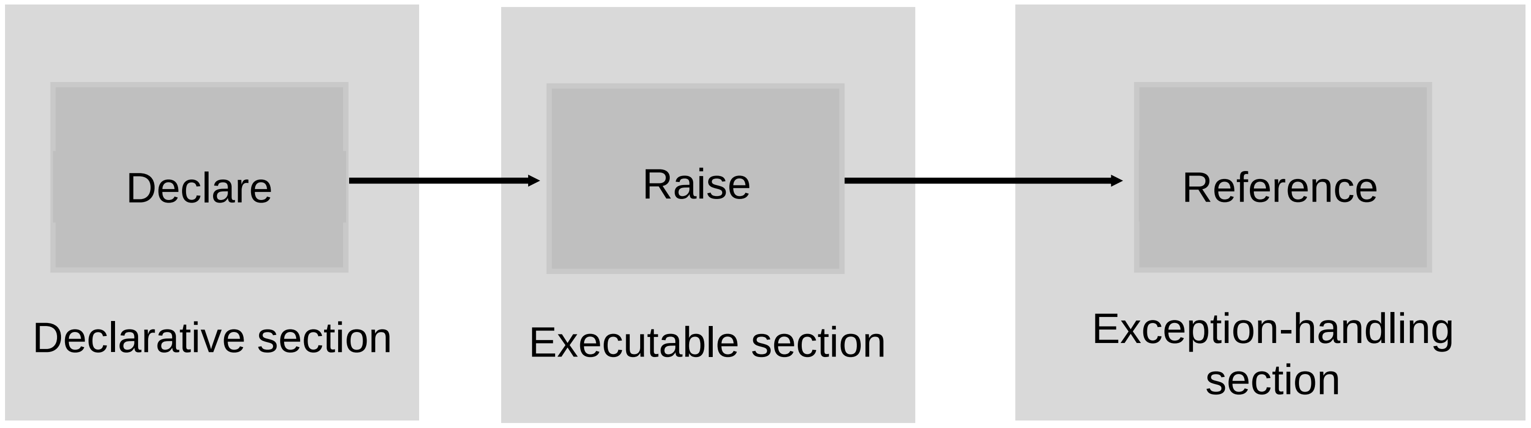
Exception Types

This lesson discusses user-defined errors.

Exception	Description	Instructions for Handling
Predefined Oracle server error	One of approximately 20 errors that occur most often in PL/SQL code	You need not declare these exceptions. They are predefined by the Oracle server and are raised implicitly (automatically).
Non-predefined Oracle server error	Any other standard Oracle server error	Declare within the declarative section and allow the Oracle Server to raise them implicitly (automatically).
User-defined error	A condition that the PL/SQL programmer decides is abnormal	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.



Name the exception.

Explicitly raise the exception by using the `RAISE` statement.

Handle the raised exception.

Trapping User-Defined Exceptions

One example of the need for a user-defined exception is during the input of data. Let's assume that your program prompts the user for a department number and name so that it can update the name of the department. What happens when the user enters an invalid department? The code doesn't produce an Oracle error. You need to define a predefined-user error to raise an error.

```
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 (cont.)

What happens when the user enters an invalid department? The code as written doesn't produce an Oracle error. You need to define a predefined-user error to raise an error. 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 (cont.)

3. Referencing the declared exception within the corresponding exception-handling routine.

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

Trapping User-Defined Exceptions (cont.)

The following is the completed code.

```
DECLARE
  e_invalid_department EXCEPTION;        1
  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;        2
  END IF;
  COMMIT;
EXCEPTION
  WHEN e_invalid_department        3
    THEN DBMS_OUTPUT.PUT_LINE('No such department id.');
         ROLLBACK;
END;
```

Trapping User-Defined Exceptions (cont.)

1. Declare the name of the user-defined exception within the declarative section. Syntax: *`exception`* `EXCEPTION;` where *`exception`* is the name of the exception.

```
DECLARE                                              1
  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                         2
    RAISE e_invalid_department;
  END IF;
  COMMIT;                                  3
EXCEPTION
  WHEN e_invalid_department
    THEN DBMS_OUTPUT.PUT_LINE('No such department id.');
         ROLLBACK;
END;
```

Trapping User-Defined Exceptions (cont.)

2. Use the `RAISE` statement to raise the exception explicitly within the executable section. Syntax:
   `RAISE` *`exception`*;
   where *`exception`* is the previously declared exception.

```
DECLARE
  e_invalid_department EXCEPTION;        1
  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;        2
  END IF;
  COMMIT;
EXCEPTION
  WHEN e_invalid_department        3
    THEN DBMS_OUTPUT.PUT_LINE('No such department id.');
         ROLLBACK;
END;
```

Trapping User-Defined Exceptions (cont.)

3. Reference the declared exception within the corresponding exception-handling routine.

```
DECLARE
  e_invalid_department EXCEPTION;          1
  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;          2
  END IF;
  COMMIT;
EXCEPTION
  WHEN e_invalid_department          3
    THEN DBMS_OUTPUT.PUT_LINE('No such department id.');
         ROLLBACK;
END;
```

The RAISE Statement

You can use the RAISE statement to raise a named exception. You can raise:

- An exception of your own (that is, 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;
```

- 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 main advantage of using `RAISE_APPLICATION_ERROR` instead of `RAISE` is that `RAISE_APPLICATION_ERROR` allows you to associate your own error number and meaningful message with the exception.
- The error numbers must fall between -20000 and -20999. Syntax:

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

The `RAISE_APPLICATION_ERROR` Procedure (cont.)

- *`error_number`* is a user-specified number for the exception between –20000 and –20999
- *`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` Procedure (cont.)

- `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 Procedure (cont.)

The number range -20000 to -20999 is reserved by Oracle for programmer use, and is never used for predefined Oracle Server errors.

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

The `RAISE_APPLICATION_ERROR` Procedure (cont.)

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('The employee who works for
           manager_id '||v_mgr||'  is: '||v_employee_id);
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