ORACLE®
ACADEMY

Database Programming with PL/SQL

7-2
Trapping Oracle Server Exceptions

Objectives

This lesson covers the following objectives:

- Describe and provide an example of an error defined by the Oracle server
- Describe and provide an example of an error defined by the PL/SQL programmer
- Differentiate between errors that are handled implicitly and explicitly by the Oracle server
- Write PL/SQL code to trap a predefined Oracle server error

Objectives

This lesson covers the following objectives:

- Write PL/SQL code to trap a non-predefined Oracle server error
- Write PL/SQL code to identify an exception by error code and by error message

Purpose

- PL/SQL error handling is flexible and allows programmers to handle Oracle server errors and errors defined by the programmer.
- This lesson discusses Oracle server errors.
- User/programmer-defined errors will be discussed in the next lesson.
- Oracle server errors can be either predefined or non-predefined.

Purpose

- Both types have an error code and a message.
- The predefined errors are the most common and they also have a name (ex., `NO_DATA_FOUND`, `TOO_MANY_ROWS`, etc.).

Exception Types

This lesson discusses both predefined and non-predefined Oracle server 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.

Handling Exceptions with PL/SQL

There are two methods for raising an exception:

- Implicitly (automatically) by the Oracle server:
  - An Oracle error occurs and the associated exception is raised automatically.
  - For example, if the error `ORA-01403` occurs when no rows are retrieved from the database in a `SELECT` statement, then PL/SQL raises the exception `NO_DATA_FOUND`.

Handling Exceptions with PL/SQL

- Explicitly by the programmer:
  - Depending on the business functionality your program is implementing, you might have to explicitly raise an exception.
  - You raise an exception explicitly by issuing the `RAISE` statement within the block.
  - The exception being raised can be either user-defined or predefined.
  - User-defined exceptions are explained in the next lesson.

Two Types of Oracle Server Errors

- When an Oracle server error occurs, the Oracle server automatically raises the associated exception, skips the rest of the executable section of the block, and looks for a handler in the exception section.
- As mentioned earlier, Oracle server errors can be predefined or non-predefined.

Two Types of Oracle Server Errors

Predefined Oracle server errors:

– Each of these errors has a predefined name, in addition to a standard Oracle error number (ORA-#####) and message.

– For example, if the error `ORA-01403` occurs when no rows are retrieved from the database in a `SELECT` statement, then PL/SQL raises the predefined exception `NO_DATA_FOUND`.

Two Types of Oracle Server Errors

Non-predefined Oracle server errors:

- Each of these errors has a standard Oracle error number (ORA-#####) and error message, but not a predefined name.
- You declare your own names for these so that you can reference these names in the exception section.

Trapping Predefined Oracle Server Errors

- Reference the predefined name in the exception handling routine.
- Sample predefined exceptions:
  - NO_DATA_FOUND
  - TOO_MANY_ROWS
  - INVALID_CURSOR
  - ZERO_DIVIDE
  - DUP_VAL_ON_INDEX

Trapping Predefined Oracle Server Errors

- For a partial list of predefined exceptions, refer to the short list available from the Student Resources in Section 0.
- For a complete list of predefined exceptions, see the *PL/SQL User's Guide and Reference*.

Trapping Predefined Oracle Server Errors

- The following example uses the TOO_MANY_ROWS predefined Oracle server error.
- Note that it is not declared in the DECLARATION section.

```
DECLARE
  v_lname VARCHAR2(15);
BEGIN
  SELECT last_name INTO v_lname
    FROM employees WHERE job_id = 'ST_CLERK';
  DBMS_OUTPUT.PUT_LINE('The last name of the ST_CLERK is: ' || v_lname);
EXCEPTION
  WHEN TOO_MANY_ROWS THEN
    DBMS_OUTPUT.PUT_LINE ('Your select statement retrieved multiple
      rows. Consider using a cursor.');
END;
```

Trapping Several Predefined Oracle Server Errors

- This example handles TOO_MANY_ROWS and NO_DATA_FOUND, with an OTHERS handler in case any other error occurs.

```
DECLARE
  v_lname VARCHAR2(15);
BEGIN
  SELECT last_name INTO v_lname
    FROM employees WHERE job_id = 'ST_CLERK';
  DBMS_OUTPUT.PUT_LINE('The last name of the ST_CLERK is: '||v_lname);
EXCEPTION
 WHEN TOO_MANY_ROWS THEN
    DBMS_OUTPUT.PUT_LINE ('Select statement found multiple rows');
  WHEN NO_DATA_FOUND THEN
    DBMS_OUTPUT.PUT_LINE ('Select statement found no rows');
  WHEN OTHERS THEN
    DBMS_OUTPUT.PUT_LINE ('Another type of error occurred');
END;
```

Trapping Non-Predefined Oracle Server Errors

- Non-predefined exceptions are similar to predefined exceptions, except they do not have predefined names.
- They do have a standard Oracle error number (ORA-#####) and error message.
- To use specific handlers (rather than handling through an `OTHERS` clause), you create your own names for them in the `DECLARE` section and associate the names with the specific ORA-##### numbers using the `PRAGMA EXCEPTION_INIT` function.

Trapping Non-Predefined Oracle Server Errors

- You can trap a non-predefined Oracle server error by declaring it first.
- The declared exception is raised implicitly. In PL/SQL, the `PRAGMA EXCEPTION_INIT` tells the compiler to associate an exception name with a specific Oracle error number.
- This allows you to refer to any Oracle Server exception by a name and to write a specific handler for it.

Non-Predefined Error

- Examine the following example.

```
BEGIN
 INSERT INTO departments
   (department_id, department_name) VALUES (280, NULL);
END;
```

- The code above results in the error message below.

```
ORA-01400: cannot insert NULL into
("US_1217_S19_PLSQL"."DEPARTMENTS"."DEPARTMENT_NAME")
```

Non-Predefined Error

- The `INSERT` statement tries to insert the value `NULL` for the `department_name` column of the `departments` table.
- However, the operation is not successful because `department_name` is a `NOT NULL` column.
- There is no predefined error name for violating a `NOT NULL` constraint.
- The following slides will demonstrate how to "handle" non-predefined exceptions.

Non-Predefined Error

- Declare the name of the exception in the declarative section.

```
DECLARE
  e_insert_excep EXCEPTION;
  PRAGMA EXCEPTION_INIT(e_insert_excep, -01400);
BEGIN
  INSERT INTO departments
    (department_id, department_name)
    VALUES (280, NULL);
EXCEPTION
  WHEN e_insert_excep
    THEN
      DBMS_OUTPUT.PUT_LINE('INSERT FAILED');
END;
```

1

```
Syntax:

exception_name EXCEPTION;
```

Non-Predefined Error

- Associate the declared exception name with the standard Oracle server error number using the PRAGMA EXCEPTION_INIT function.

```
DECLARE
  e_insert_excep EXCEPTION;
  PRAGMA EXCEPTION_INIT(e_insert_excep, -01400);
BEGIN
  INSERT INTO departments
    (department_id, department_name)
    VALUES (280, NULL);
EXCEPTION
  WHEN e_insert_e
    THEN
      DBMS_OUTPUT
END;
```

2

```
Syntax:

PRAGMA EXCEPTION_INIT(exception_name, -number);
```

Non-Predefined Error

- Reference the declared exception name within a WHEN clause in the exception-handling section.

```
DECLARE
  e_insert_excep EXCEPTION;
  PRAGMA EXCEPTION_INIT(e_insert_excep, -01400);
BEGIN
  INSERT INTO departments
    (department_id, department_name)
    VALUES (280, NULL);
EXCEPTION
  WHEN e_insert_excep
    THEN
      DBMS_OUTPUT.PUT_LINE('INSERT FAILED');
END;
```

3

Functions for Trapping Exceptions

- When an exception occurs, you can retrieve the associated error code or error message by using two functions.
- Based on the values of the code or the message, you can decide which subsequent actions to take.
  - `SQLERRM` returns character data containing the message associated with the error number.
  - `SQLCODE` returns the numeric value for the error code. (You can assign it to a `NUMBER` variable.)

Functions for Trapping Exceptions

SQLCODE Value	Description
0	No exception encountered
1	User defined exception
+100	`NO_DATA_FOUND` exception
Negative number	Another Oracle Server error number

Functions for Trapping Exceptions

- You cannot use SQLCODE or SQLERRM directly in an SQL statement.
- Instead, you must assign their values to local variables, then use the variables in the SQL statement, as shown in the following example:

```
DECLARE
  v_error_code      NUMBER;
  v_error_message   VARCHAR2(255);
BEGIN           ...
EXCEPTION
  WHEN OTHERS THEN
    ROLLBACK;
    v_error_code    := SQLCODE;
    v_error_message := SQLERRM;
    INSERT INTO error_log(e_user, e_date, error_code, error_message)
          VALUES(USER, SYSDATE, v_error_code, v_error_message);
END;
```

Terminology

Key terms used in this lesson included:

- Non-predefined Oracle server errors
- Predefined Oracle server errors
- `PRAGMA EXCEPTION_INIT`
- `SQLERRM`
- `SQLCODE`

Summary

In this lesson, you should have learned how to:

- Describe and provide an example of an error defined by the Oracle server.
- Describe and provide an example of an error defined by the PL/SQL programmer
- Differentiate between errors that are handled implicitly and explicitly by the Oracle server
- Write PL/SQL code to trap a predefined Oracle server error

Summary

In this lesson, you should have learned how to:

- Write PL/SQL code to trap a non-predefined Oracle server error
- Write PL/SQL code to identify an exception by error code and by error message

ORACLE® ACADEMY