Database Programming

Error Handling & Exceptions

Sagara Samarawickrama

CSD 4204 - CPCM GP 1,2 & 3

Error Handling & Exceptions

In "PL/SQL Concepts," we encountered two types of errors that can be found in a program: compilation errors and runtime errors. You will recall that a special section in a PL/SQL block handles runtime errors. This is called the exception-handling section, and in it, runtime errors are called exceptions. The exception-handling section allows programmers to specify what actions should be taken when a specific exception occurs.

PL/SQL has two types of exceptions: built-in and user-defined. In this lesson, we will learn how to handle certain kinds of runtime errors with the help of built-in exceptions

Error Handling & Exceptions

The following example illustrates some of the differences between compilation and runtime errors:

```
DECLARE
   v_num1    INTEGER := &sv_num1;
   v_num2    INTEGER := &sv_num2;
   v_result NUMBER;
BEGIN
   v_result = v_num1 / v_num2;
   DBMS_OUTPUT.PUT_LINE ('v_result: '||v_result);
END;
```

This example is a very simple program. It has two variables, v_num1 and v_num2. A user supplies values for these variables. Next, v_num1 is divided by v_num2, and the result of this division is stored in the third variable, v_result. Finally, the value of v_result is displayed on the screen.

Error Handling & Exceptions

Now, assume that a user supplies values of 3 and 5 for the variables v_num1 and v_num2,respectively. As a result, the example produces the following output:

```
Enter value for sv_num1: 3
old   2:     v_num1 INTEGER := &sv_num1;
new   2:     v_num1 INTEGER := 3;
Enter value for sv_num2: 5
old   3:     v_num2 INTEGER := &sv_num2;
new   3:     v_num2 INTEGER := 5;
   v_result = v_num1 / v_num2;
              *
ERROR at line 6:
ORA-06550: line 6, column 13:
PLS-00103: Encountered the symbol "=" when expecting one of the
following:


:= . ( @ % ;
The symbol ":= was inserted before "=" to continue.
```

Error Handling & Exceptions

v_result = v_num1 / v_num2;

contains an equals-sign operator where an assignment operator should be used. The statement

should be rewritten as follows:

v_result := v_num1 / v_num2;

Next, if you change the values of the variables v_num1 and v_num2 to 4 and 0, respectively,the following output is produced:

```
Enter value for sv_num1: 4
old   2:     v_num1 integer := &sv_num1;
new   2:     v_num1 integer := 4;
Enter value for sv_num2: 0
old   3:     v_num2 integer := &sv_num2;
new   3:     v_num2 integer := 0;
DECLARE
*
ERROR at line 1:
ORA-01476: divisor is equal to zero
ORA-06512: at line 6
```

Error Handling & Exceptions

Even though this example does not contain syntax errors, it was terminated prematurely because the value entered for v_num2, the divisor, was 0. As you may recall, division by 0 is undefined and thus leads to an error. This example illustrates a runtime error that the compiler cannot detect

To handle this type of error in the program, you must add an exception handler. The exception handling section has the following structure:

```
EXCEPTION
   WHEN EXCEPTION_NAME THEN
      ERROR-PROCESSING STATEMENTS;
```

The exception-handling section is placed after the executable section of the block. The preceding example can be rewritten in the following manner:

Error Handling & Exceptions

```
DECLARE
   v_num1    INTEGER := &sv_num1;
   v_num2    INTEGER := &sv_num2;
   v_result NUMBER;
BEGIN
   v_result := v_num1 / v_num2;
   DBMS_OUTPUT.PUT_LINE ('v_result: '||v_result);
EXCEPTION
   WHEN ZERO_DIVIDE THEN
       DBMS_OUTPUT.PUT_LINE ('A number cannot be divided by zero.');
END;
```

The section of the example in bold shows the exception-handling section of the block

An exception-handling section allows a program to execute to completion, instead of terminating prematurely. Another advantage offered by the exception-handling section is isolation of error-handling routines

Error Handling & Exceptions

Built-in exceptions

As mentioned earlier, a PL/SQL block has the following structure:

```
DECLARE
   ...
BEGIN
   EXECUTABLE STATEMENTS;
EXCEPTION
   WHEN EXCEPTION_NAME THEN
      ERROR-PROCESSING STATEMENTS;
END;
```

When an error occurs that raises a built-in exception, the exception is said to be raised implicitly.In other words, if a program breaks an Oracle rule, control is passed to the exception handling section of the block. At this point, the error-processing statements are executed. It is important to realize that after the exception-handling section of the block has executed, the block terminates. Control does not return to the executable section of the block.

Error Handling & Exceptions

Consider the following example :

```
DECLARE
   v_student_name VARCHAR2(50);
BEGIN
   SELECT first_name||' '||last_name
     INTO v_student_name
     FROM student
    WHERE student_id = 101;

    DBMS_OUTPUT.PUT_LINE ('Student name is '||v_student_name);
EXCEPTION
   WHEN NO_DATA_FOUND THEN
      DBMS_OUTPUT.PUT_LINE ('There is no such student');
END;
```

This example produces the following output:

There is no such student

PL/SQL procedure successfully completed.

Error Handling & Exceptions

Because there is no record in the STUDENT table with student ID 101, the SELECT INTO statement does not return any rows. As a result, control passes to the exception-handling section of the block, and the error message There is no such student is displayed. Even though there is a DBMS_OUTPUT.PUT_LINE statement right after the SELECT statement, it is not executed,because control has been transferred to the exception-handling section. Control never returns to the executable section of this block, which contains the first DBMS_OUTPUT.PUT_LINE statement.

Some common Oracle runtime errors are predefined in PL/SQL as exceptions

Error Handling & Exceptions

The following list describes some commonly used predefined exceptions and how they are raised:

NO_DATA_FOUND: This exception is raised when a SELECT INTO statement that makes no calls to group functions, such as SUM or COUNT, does not return any rows. For example, suppose you issue a SELECT INTO statement against the STUDENT table where student ID equals 101. If there is no record in the STUDENT table passing this criteria (student ID equals 101), the NO_DATA_FOUND exception is raised. When a SELECT INTO statement calls a group function, such as COUNT, the result set is never empty. When used in a SELECT INTO statement against the STUDENT table, function COUNT returns 0 for the value of student ID 123.

Error Handling & Exceptions

The following list describes some commonly used predefined exceptions and how they are raised:

NO_DATA_FOUND: This exception is raised when a SELECT INTO statement that makes no calls to group functions, such as SUM or COUNT, does not return any rows. For example, suppose you issue a SELECT INTO statement against the STUDENT table where student ID equals 101. If there is no record in the STUDENT table passing this criteria (student ID equals 101), the NO_DATA_FOUND exception is raised. When a SELECT INTO statement calls a group function, such as COUNT, the result set is never empty. When used in a SELECT INTO statement against the STUDENT table, function COUNT returns 0 for the value of student ID 123.

Error Handling & Exceptions

Hence, a SELECT statement that calls a group function never raises the NO_DATA_FOUND exception.

TOO_MANY_ROWS: This exception is raised when a SELECT INTO statement returns more than one row. By definition, a SELECT INTO can return only a single row. If a SELECT INTO statement returns more than one row, the definition of the SELECT INTO statement is violated. This causes the TOO_MANY_ROWS exception to be raised.

For example, suppose you issue a SELECT INTO statement against the STUDENT table for a specific zip code. There is a good chance that this SELECT statement will return more than one row, because many students can live in the same zip code area.

Error Handling & Exceptions

ZERO_DIVIDE: This exception is raised when a division operation is performed in the program and a divisor is equal to 0. An example in the previous lab of this chapter illustrated how this exception is raised.

LOGIN_DENIED: This exception is raised when a user tries to log in to Oracle with an invalid username or password.

PROGRAM_ERROR: This exception is raised when a PL/SQL program has an internal problem.

VALUE_ERROR: This exception is raised when a conversion or size mismatch error occurs. For example, suppose you select a student's last name into a variable that has been defined as VARCHAR2(5). If the student's last name contains more than five characters, the

Error Handling & Exceptions

the VALUE_ERROR exception is raised.

DUP_VALUE_ON_INDEX: This exception is raised when a program tries to store a duplicate value in the column or columns that have a unique index defined on them. For example, suppose you are trying to insert a record into the SECTION table for course number 25, section 1. If a record for the given course and section number already exists in the SECTION table, the DUP_VAL_ON_INDEX exception is raised, because these columns have a unique index defined on them.