Exercises for Chapter 8: Error Handling and Built-In Exceptions

The Labs below provide you with exercises and suggested answers with discussion related to how those answers resulted. The most important thing to realize is whether your answer works. You should figure out the implications of the answers here and what the effects are from any different answers you may come up with.

Lab 8.1 Handling Errors

In this exercise, you will calculate the value of the square root of a number and display it on the screen. Create the following PL/SQL script:

For Example *ch08_5a.sql*

```
DECLARE
   v_num NUMBER := &sv_num;
BEGIN
   DBMS_OUTPUT.PUT_LINE ('Square root of '||v_num||' is '||SQRT(v_num));
EXCEPTION
   WHEN VALUE_ERROR
   THEN
      DBMS_OUTPUT.PUT_LINE ('An error has occurred');
END;
```

In the preceding script, the exception VALUE_ERROR, is raised when conversion or type mismatch errors occur. This exception is covered in greater detail in Lab 8.2 of this chapter.

In order to test this script fully, execute it two times. For the first run, enter a value of 4 for the variable v_num. For the second run, enter the value of –4 for the variable v_num.

Execute the script, and then answer the following questions:

a) What output was generated by the script (for both runs)?

   Answer: The first version of the output is produced when the value of the variable v_num is 4. The output should look like the following:

   ```
   Square root of 4 is 2
   ```

The second version of the output is produced when the value of the variable `v_num` is –4. The output should look like the following:

```
An error has occurred
```

b) Why do you think an error message was generated when the script was run a second time?

Answer: Error message "An error has occurred" was generated for the second run of example because a runtime error has occurred. The built-in function `SQRT` is unable to accept a negative number as its argument. Therefore, the exception `VALUE_ERROR` was raised, and the error message was displayed in the Dbms Output window.

c) Assume that you are not familiar with the exception `VALUE_ERROR`. How would you change this script to avoid this runtime error?

Answer: The new version of the script should look similar to the one below. All changes are shown in bold.

For Example *ch08_5b.sql*

```
DECLARE
   v_num NUMBER := &sv_num;
BEGIN
   IF v_num >= 0
   THEN
      DBMS_OUTPUT.PUT_LINE ('Square root of '||v_num||' is '||SQRT(v_num));
   ELSE
      DBMS_OUTPUT.PUT_LINE ('A number cannot be negative');
   END IF;
END;
```

Notice that in this version of the script, the value of the `v_num` variable is checked with the help of `IF-THEN-ELSE` statement. If the incoming value is negative, the message "A number cannot be negative" is displayed in the Dbms Output window. When the value of –4 is entered for the variable `v_num`, this script produces the following output:

```
A number cannot be negative
```

Lab 8.2 Built-In Exceptions

In this exercise, you will learn more about some built-in exceptions discussed in this chapter. Create the following PL/SQL script:

For Example *ch08_6a.sql*

```
DECLARE
   v_exists         NUMBER(1);
   v_total_students NUMBER(1);
   v_zip            CHAR(5):= '&sv_zip';
BEGIN
   SELECT count(*)
     INTO v_exists
     FROM zipcode
```

```
    WHERE zip = v_zip;

   IF v_exists != 0
   THEN
      SELECT COUNT(*)
        INTO v_total_students
        FROM student
       WHERE zip = v_zip;

      DBMS_OUTPUT.PUT_LINE ('There are '||v_total_students||' students');
   ELSE
      DBMS_OUTPUT.PUT_LINE (v_zip||' is not a valid zip');
   END IF;

EXCEPTION
   WHEN VALUE_ERROR OR INVALID_NUMBER
   THEN
      DBMS_OUTPUT.PUT_LINE ('An error has occurred');
END;
```

This script contains two exceptions, `VALUE_ERROR` and `INVALID_NUMBER`. However, only one exception handler is written for both exceptions. You can combine different exceptions in a single exception handler when you want to handle both exceptions in a similar way. Often the exceptions `VALUE_ERROR` and `INVALID_NUMBER` are used in a single exception handler because these Oracle errors refer to the conversion problems that may occur at runtime.

In order to test this script fully, execute it three times for the following ZIP code values: 07024, 00914, and 12345, and then answer the following questions:

a) What output was generated by the script (for all values of zip)?

Answer: The first version of the output is produced when the value of zip is 07024. The second version of the output is produced when the value of zip is 00914. The third version of the output is produced when the value of zip is 12345.

The first output should look like the following:

```
There are 9 students
```

When "07024" is entered for the variable `v_zip`, the first `SELECT INTO` statement is executed. This `SELECT INTO` statement checks whether the value of zip is valid, or, in other words, if a record exists in the `ZIPCODE` table for a given value of zip. Next, the value of the variable `v_exists` is evaluated with the help of the `IF` statement. For this run of the example, the `IF` statement evaluates to `TRUE`, and, as a result, the `SELECT INTO` statement against the `STUDENT` table is executed. Next, the `DBMS_OUTPUT.PUT_LINE` statement following the `SELECT INTO` statement is executed, and the message "There are 9 students" is displayed in the Dbms Output window.

The second output should look like the following:

```
There are 0 students
```

For the second run, the value "00914" is entered for the variable `v_zip`. Similarly to the above, the `SELECT INTO` statement against the `ZIPCODE` table is executed. Next, the variable `v_exists` is evaluated with the help of the `IF` statement. Because the `IF` statement evaluates to TRUE, the SELECT INTO statement against the STUDENT table is executed as well, and the message "There are 0 students" is displayed in the Dbms Output window.

Note that because the SELECT INTO statement against the STUDENT table uses group function, COUNT, there is no reason to use the exception NO_DATA_FOUND, because the COUNT function will always return data.

The third output should look like the following:

```
12345 is not a valid zip
```

For the third run, the value "12345" is entered for the variable v_zip. The SELECT INTO statement against the ZIPCODE table is executed. Next, the variable v_exists is evaluated with the help of the IF statement. Because the value of v_exists equals 0, the IF statement evaluates to FALSE. As a result, the ELSE part of the IF statement is executed, and the message "12345 is not a valid zip" is displayed in the Dbms Output window.

b) Explain why no exception has been raised for these values of the variable v_zip.

Answer: The exceptions VALUE_ERROR or INVALID_NUMBER have not been raised because there was no conversion or type mismatch error. Both variables, v_exists and v_total_students, have been defined as NUMBER(1).
The group function COUNT used in the SELECT INTO statements returns a NUMBER datatype. Moreover, on both occasions, a single digit number is returned by the COUNT function. As a result, neither exception has been raised.

c) Insert a record into the STUDENT table with a zip having the value of "07024" as indicated below:

```
INSERT INTO student (student_id, salutation, first_name, last_name, zip,
   registration_date, created_by, created_date, modified_by, modified_date)
VALUES (STUDENT_ID_SEQ.NEXTVAL, 'Mr.', 'John', 'Smith', '07024',
   SYSDATE, 'STUDENT', SYSDATE, 'STUDENT', SYSDATE);
```

Run the script again for the same value of zip ("07024"). What output was generated by the script? Why?

Answer: After a student has been added, the output should look like the following:

```
An error has occurred
```

Once a new record has been inserted into the STUDENT table with a ZIP having a value of "07024," the total number of students in this ZIP code changes from 9 to 10. As a result, the SELECT INTO statement against the STUDENT table causes an error, because the variable v_total_students has been defined as NUMBER(1). This means that only a single-digit number can be stored in this variable. The number 10 is a two-digit number, so the exception INVALID_NUMBER is raised. As a result, the message "An error has occurred" is displayed in the Dbms Output window.

Watch Out!

After completing this exercise remember to rollback your changes to the STUDENT table as newly added record may affect the outputs of the future examples and exercises.

Try It Yourself

The projects in this section are meant to have you use all of the skills that you have acquired throughout this chapter. Here are some exercises that will help you test the depth of your understanding

1) Create the following script: For a given student ID, display student's first and last names, and the number of courses that this student is registered for. Note that the script should contain appropriate exception handling.

   Answer: The script should look similar to the following:

   For Example *ch08_7a.sql*

```
DECLARE
   v_student_id  NUMBER := &sv_student_id;
   v_name        VARCHAR2(30);
   v_enrollments NUMBER;
BEGIN
   SELECT first_name||' '||last_name
     INTO v_name
     FROM student
    WHERE student_id = v_student_id;

   SELECT COUNT(*)
     INTO v_enrollments
     FROM enrollment
    WHERE student_id = v_student_id;

   DBMS_OUTPUT.PUT_LINE
      ('Student '||v_name||' has '||v_enrollments||' enrollments');
EXCEPTION
   WHEN NO_DATA_FOUND
   THEN
      DBMS_OUTPUT.PUT_LINE ('This student does not exist');
END;
```

   This script above accepts a value for student's ID from a user. For a given student ID, it determines the student's name via the `SELECT INTO` statement. Then, it checks the number of enrollments that a given student has via the `SELECT INTO` statement against the `ENROLLMENT` table. Finally it displays this information it in the Dbms Output window.
   Note that this script has exception-handling section with a single exception, `NO_DATA_FOUND`. This exception is necessary because first `SELECT INTO` statement causes `NO_DATA_FOUND` exception when non-existent student ID is passed into the script. In such a case, the control of the execution is passed to the exception-handling section and the message "This student does not exist" is displayed in the Dbms Output window.
   Consider testing this script, for these student IDs: 230, 284, and 999. Each run produces output as indicated below:

```
Student George Kocka has 1 enrollments


Student Salewa Lindeman has 0 enrollments


This student does not exist
```

Note that for the second run (student ID 248), the `SELECT INTO` statement against the `ENROLLMENT` table completes successfully because it employs `COUNT` function which does not cause the `NO_DATA_FOUND` exception as it is a group function.

Next, consider a different approach that may be used to achieve the same result. Note that in this case even though the `SELECT INTO` statement contains `COUNT` function, it may cause `NO_DATA_FOUND` exception due to how the data is selected from the `STUDENT` and `ENROLLMENT` tables. All changes are highlighted in bold.

For Example *ch08_7b.sql*

```
DECLARE
   v_student_id  NUMBER := &sv_student_id;
   v_name        VARCHAR2(30);
   v_enrollments NUMBER;
BEGIN
   SELECT s.first_name||' '||s.last_name, COUNT(*)
     INTO v_name, v_enrollments
     FROM student s, enrollment e
    WHERE s.student_id = e.student_id
      AND s.student_id = v_student_id
   GROUP BY s.first_name||' '||s.last_name;

   DBMS_OUTPUT.PUT_LINE
      ('Student '||v_name||' has '||v_enrollments||' enrollments');
EXCEPTION
   WHEN NO_DATA_FOUND
   THEN
      BEGIN
         SELECT first_name||' '||last_name
           INTO v_name
           FROM student
          WHERE student_id = v_student_id;

        DBMS_OUTPUT.PUT_LINE ('Student '||v_name||' is not enrolled');
      EXCEPTION
         WHEN NO_DATA_FOUND
         THEN
            DBMS_OUTPUT.PUT_LINE ('This student does not exist');
      END;
END;
```

In this version of the script, there is a single `SELECT INTO` statement against the `STUDENT` and `ENROLLMENT` tables. As a result, for a given student ID it is possible to encounter a `NO_DATA_FOUND` exception in two cases:

- There is no student record in the `STUDENT` table
- There is no student record in the `ENROLLMENT` table

This is demonstrated further by the script output for the same student ID values used earlier (230, 284, and 999):

```
Student George Kocka has 1 enrollments
```

And

```
Student Salewa Lindeman has 0 enrollments
```

And

```
This student does not exist
```

To attain similar behavior, the exception-handling section in this version has been expanded to contain a nested PL/SQL block. This is done for error-reporting as when the `SELECT INTO` statement against the `STUDENT` and `ENROLLMENT` table causes `NO_DATA_FOUND` exception the reason for it is not known in advance.

2) Create the following script: For a given course ID, display course name, number of sections this course has, and total number of students enrolled in this course. Note that the script should contain appropriate exception handling.

   Answer: The script should look similar to the following:

   For Example *ch08_8a.sql*

```
DECLARE
   v_course_no NUMBER := &sv_course_no;
   v_name      VARCHAR2(50);
   v_sections  NUMBER;
   v_students  NUMBER;
 BEGIN
   SELECT description
     INTO v_name
     FROM course
    WHERE course_no = v_course_no;

   -- check how many sections are offered for a given course
   SELECT COUNT(*)
     INTO v_sections
     FROM section
    WHERE course_no = v_course_no;

  -- check how many students are enrolled in a given course
  SELECT COUNT(e.student_id)
    INTO v_students
    FROM section s
        ,enrollment e
   WHERE s.section_id = e.section_id
     AND s.course_no = v_course_no;

   DBMS_OUTPUT.PUT_LINE
      ('Course '||v_course_no||', '||v_name||', has '||v_sections||' section(s)');
   DBMS_OUTPUT.PUT_LINE (v_students||' students are enrolled in this course');
EXCEPTION
   WHEN NO_DATA_FOUND
   THEN
      DBMS_OUTPUT.PUT_LINE (v_course_no||' is not a valid course');
END;
```

This script accepts a value for a course number at a run time. For a given course number, it selects course description from the `COURSE` table. If provided course number does not exists in the `COURSE` table, the control of the execution is passed to the exception-handling section of the block, where the NO_DATA_FOUND exception is raised. As a result, the message ‘This is not a valid course’ is displayed in the Dbms Output window. On the other hand, if the value provided is a valid course number, the second SELECT INTO statement determines how many sections are offered for a given course number, and the third SELECT INTO statement determines how many students in total are registered for this course.

To test this script fully, consider running it twice. For the first run, the course number provided at the run time is a valid course, and for the second run, the course number provided at the run time should be for non-existent course. This is further demonstrated by the outputs below for course numbers 25 (valid course number) and 999 (invalid course number):

```
Course 25, Intro to Programming, has 9 section(s)
45 students are enrolled in this course
```

And

```
999 is not a valid course
```

Next consider another version of the script where the SELECT INTO statements against the SECTION and ENROLLMENT tables are combined into a single SELECT statement as demonstrated below (changes are shown in bold):

For Example *ch08_8b.sql*

```
DECLARE
   v_course_no NUMBER := &sv_course_no;
   v_name      VARCHAR2(50);
   v_sections  NUMBER;
   v_students  NUMBER;
 BEGIN
   SELECT description
     INTO v_name
     FROM course
    WHERE course_no = v_course_no;

   -- check how many sections are offered for a given course and
   -- how many students are enrolled in a given course
   SELECT COUNT(UNIQUE e.section_id), COUNT(e.student_id)
     INTO v_sections, v_students
    FROM section    s
        ,enrollment e
   WHERE s.section_id = e.section_id
     AND s.course_no = v_course_no;

   DBMS_OUTPUT.PUT_LINE
      ('Course '||v_course_no||', '||v_name||', has '||v_sections||' section(s)');
   DBMS_OUTPUT.PUT_LINE (v_students||' students are enrolled in this course');
EXCEPTION
   WHEN NO_DATA_FOUND
   THEN
      DBMS_OUTPUT.PUT_LINE (v_course_no||' is not a valid course');
END;
```

When this version of the script is executed for the course number 25, it produces slightly different output:

```
Course 25, Intro to Programming, has 8 section(s)
```

```
45 students are enrolled in this course
```

This occurs because one of the sections does not have any students enrolled in it, and as a result does not have any corresponding records in the `ENROLLMENT` table. *Essentially, this version of the example contains a logical run time error that does not cause any exceptions and produces incorrect result.*