Exercises for Chapter 10: Exceptions: Advanced Concepts

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 10.1 RAISE_APPLICATION_ERROR

In this exercise, you calculate how many students are registered for a given section of a given course. You then display a message on the screen that contains the course number and the number of students registered for it. The original PL/SQL script will not contain any exception handlers, so you will be asked to add the `RAISE_APPLICATION_ERROR` statements.

Create the following PL/SQL script:

For Example *ch10_6a.sql*

```
DECLARE
   v_students NUMBER(3) := 0;
BEGIN
   SELECT COUNT(*)
     INTO v_students
     FROM enrollment e, section s
    WHERE e.section_id = s.section_id
      AND s.course_no  = 25
      AND s.section_id = 89;

   DBMS_OUTPUT.PUT_LINE ('Course 25, section 89 has '||v_students||' students');
END;
```

Execute the script, and then answer the following questions:

a) What output is produced by this script?

Answer: The output should look similar to the following:

```
Course 25, section 89 has 12 students
```

b) Modify this script so that if a section of a course has more than 10 students enrolled in it, an error message is displayed indicating that this course has too many students enrolled.
Answer: The script should look similar to the script shown. Newly added statements are highlighted in bold.

For Example *ch10_6b.sql*

```
DECLARE
   v_students NUMBER(3) := 0;
BEGIN
   SELECT COUNT(*)
     INTO v_students
     FROM enrollment e, section s
    WHERE e.section_id = s.section_id
      AND s.course_no  = 25
      AND s.section_id = 89;

   IF v_students > 10
   THEN
      RAISE_APPLICATION_ERROR
         (-20002, 'Course 25, section 89 has more than 10 students');
   END IF;

   DBMS_OUTPUT.PUT_LINE ('Course 25, section 89 has '||v_students||' students');
END;
```

Consider the result if you were to add an `IF` statement to this script, one in which the `IF` statement checks whether the value of the variable `v_students` exceeds 10. If the value of the variable does exceed 10, the `RAISE_APPLICATION_ERROR` statement executes, and the error message is displayed on the screen.

c) Execute the new version of the script. What output is produced by this version of the script?

Answer: The output should look similar to the following:

```
ORA-20002: Course 25, section 89 has more than 10 students
ORA-06512: at line 13
```

Note that in this case the error message generated by the script did not appear in the Dbms Output window of the Oracle SQL Developer. Instead, the error message was displayed in the Script Output window.

Lab 10.2 `EXCEPTION_INIT` Pragma

In this exercise, you insert a record in the `COURSE` table. The original PL/SQL script does not contain any exception handlers, so you are asked to define an exception and add the `EXCEPTION_INIT` pragma.

Create the following PL/SQL script:

For Example *ch10_7a.sql*

```
BEGIN
```

```
   INSERT INTO course
      (course_no, description, created_by, created_date)
   VALUES
      (COURSE_NO_SEQ.NEXTVAL, 'Test Course', USER, SYSDATE);
   COMMIT;
   DBMS_OUTPUT.PUT_LINE ('One course has been added');
END;
```

Execute the script, and then answer the following questions:

a) What output was generated by the script?

Answer: The output should look like the following:

```
ORA-01400: cannot insert NULL into ("STUDENT"."COURSE"."MODIFIED_BY")
ORA-06512: at line 2
```

b) Explain why the script does not execute successfully.

Answer: The script does not execute successfully because a `NULL` is inserted for the `MODIFIED_BY` and `MODIFIED_DATE` columns. The `MODIFED_BY` and `MODIFIED_DATE` columns have check constraints defined on them. These constraints can be viewed by querying one of the data dictionary tables.

The data dictionary comprises tables owned by the user SYS. These tables provide the database with information that it uses to manage itself. Consider the following `SELECT` statement against one of Oracle's data dictionary tables, `USER_CONSTRAINTS`. This table contains information on various constraints defined on each table of the `STUDENT` schema.

```
SELECT constraint_name, search_condition
  FROM user_constraints
WHERE table_name = 'COURSE';

CONSTRAINT_NAME           SEARCH_CONDITION
------------------------  --------------------------
CRSE_CREATED_DATE_NNULL   "CREATED_DATE" IS NOT NULL
CRSE_MODIFIED_BY_NNULL    "MODIFIED_BY" IS NOT NULL
CRSE_MODIFIED_DATE_NNULL  "MODIFIED_DATE" IS NOT NULL
CRSE_DESCRIPTION_NNULL    "DESCRIPTION" IS NOT NULL
CRSE_COURSE_NO_NNULL      "COURSE_NO" IS NOT NULL
CRSE_CREATED_BY_NNULL     "CREATED_BY" IS NOT NULL
CRSE_PK
CRSE_CRSE_FK
```

Notice that the last two rows refer to the primary and foreign key constraints, so there are no search conditions specified.

Based on the results produced by the preceding `SELECT` statement, there are six columns having a `NOT NULL` constraint. However, the `INSERT` statement

```
INSERT INTO course
   (course_no, description, created_by, created_date)
VALUES
   (COURSE_NO_SEQ.NEXTVAL, 'Test Course', USER, SYSDATE);
```

has only four columns having `NOT NULL` constraints. The columns `MODIFIED_BY` and `MODIFIED_DATE` are not included in the `INSERT` statement. Any column of a table not listed in the `INSERT` statement has `NULL` assigned to it when a new record is added to the table. If a

column has a NOT NULL constraint and is not listed in the INSERT statement, the INSERT statement fails and causes an error.

c) Add a user-defined exception to the script, so that the error generated by the INSERT statement is handled.

Answer: The script should look similar to the script shown. All changes are shown in bold.

For Example *ch10_7b.sql*

```
DECLARE
   e_constraint_violation EXCEPTION;
   PRAGMA EXCEPTION_INIT(e_constraint_violation, -1400);
BEGIN
   INSERT INTO course
      (course_no, description, created_by, created_date)
   VALUES
      (COURSE_NO_SEQ.NEXTVAL, 'Test Course', USER, SYSDATE);
   COMMIT;
   DBMS_OUTPUT.PUT_LINE ('One course has been added');
EXCEPTION
   WHEN e_constraint_violation
   THEN
      DBMS_OUTPUT.PUT_LINE ('INSERT statement is violating a constraint');
END;
```

This version of the script contains user-defined exception, e_constraint_violation. This user-defined exception is associated with unnamed Oracle error by using the EXCEPTION_INIT pragma. In this case, the Oracle error number ORA-02290 is associated with the exception named e_constraint_violation. Such association enables processing of this error by the newly added exception-handling section of the block.

d) Run the new version of the script. Explain the output produced by the new version of the script.

Answer: The output should look similar to the following:

```
INSERT statement is violating a constraint
```

Once you define an exception and associate an Oracle error number with it, you can write an exception handler for it. As a result, as soon as the INSERT statement causes an error, control of the execution is transferred to the exception-handling section of the block. Then, the message "INSERT statement . . ." is displayed on the screen. Notice that once an exception is raised and processed, the execution of the program does not halt. The script completes successfully.

Lab 10.3 SQLCODE and SQLERRM

In this exercise, you add a new record to the ZIPCODE table. The original PL/SQL script does not contain any exception handlers. You are asked to add an exception-handling section to this script.

Create the following PL/SQL script:

For Example *ch10_8a.sql*

```
BEGIN
   INSERT INTO zipcode
```

```
      (zip, city, state, created_by, created_date, modified_by, modified_date)
   VALUES
      ('10027', 'NEW YORK', 'NY', USER, SYSDATE, USER, SYSDATE);
   COMMIT;
END;
```

Execute the script and answer the following questions:

a) Explain the output produced by the script?

Answer: The output should look like the following:

```
ORA-00001: unique constraint (STUDENT.ZIP_PK) violated
ORA-06512: at line 2
```

The `INSERT` statement

```
INSERT INTO zipcode
   (zip, city, state, created_by, created_date, modified_by, modified_date)
VALUES
   ('10027', 'NEW YORK', 'NY', USER, SYSDATE, USER, SYSDATE);
```

causes an error because a record with ZIP code 10027 already exists in the `ZIPCODE` table. Column ZIP of the `ZIPCODE` table has a primary key constraint defined on it. Therefore, when you try to insert another record with the value of ZIP already existing in the `ZIPCODE` table, the error message "ORA-00001: unique constraint . . ." is generated.

b) Modify the script so that the script completes successfully, and the error number and message are displayed on the screen.

Answer: The script should resemble the script shown. Newly added exception handling section is shown in bold.

For Example *ch10_8b.sql*

```
BEGIN
   INSERT INTO zipcode
      (zip, city, state, created_by, created_date, modified_by, modified_date)
   VALUES
      ('10027', 'NEW YORK', 'NY', USER, SYSDATE, USER, SYSDATE);
   COMMIT;
EXCEPTION
   WHEN OTHERS
   THEN
      DECLARE
         v_err_code NUMBER := SQLCODE;
         v_err_msg VARCHAR2(100) := SUBSTR(SQLERRM, 1, 100);
      BEGIN
         DBMS_OUTPUT.PUT_LINE ('Error code: '||v_err_code);
         DBMS_OUTPUT.PUT_LINE ('Error message: '||v_err_msg);
      END;
END;
```

This version of the script has been extended with the exception-handling section and `OTHERS` exception handler. Notice that two variables `v_err_code` and `v_err_msg`, are declared, in

the exception-handling section of the block. In other words, the exception-handling section contains a nested (inner) PL/SQL block.

c) Run the new version of the script. Explain the output produced by the new version of the script.

Answer: The output should look similar to the following:

```
Error code: -1
Error message: ORA-00001: unique constraint (STUDENT.ZIP_PK) violated
```

Because the `INSERT` statement causes an error, control is transferred to the `OTHERS` exception handler. The `SQLCODE` function returns -1, and the `SQLERRM` function returns the text of the error corresponding to the error code -1. Once the exception-handling section completes its execution, control is passed to the host environment.

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. Modify the script created in this section in Chapter 9 (Question 1 of the Try It Yourself section). Raise a user-defined exception with the `RAISE_APPLICATION_ERROR` statement. Otherwise, display how many students there are in a section. Make sure your program is able to process all sections.

   Answer: Consider the version of the script created in Chapter 9 provided here for reference.

   For Example *ch09_12a.sql*

```
DECLARE
   v_section_id        NUMBER := &sv_section_id;
   v_total_students    NUMBER;
   e_too_many_students EXCEPTION;
BEGIN
   -- Calculate number of students enrolled
   SELECT COUNT(*)
     INTO v_total_students
     FROM enrollment
    WHERE section_id = v_section_id;

   IF v_total_students >= 10
   THEN
      RAISE e_too_many_students;
   ELSE
      DBMS_OUTPUT.PUT_LINE
         ('There are '||v_total_students||' students in section '||v_section_id);
   END IF;
EXCEPTION
   WHEN e_too_many_students
   THEN
      DBMS_OUTPUT.PUT_LINE
         ('There are too many students in section '||v_section_id);
END;
```

Next, consider modified version of this script. All changes are highlighted in bold.

For Example *ch10_9a.sql*

```
DECLARE
   v_section_id        NUMBER := &sv_section_id;
   v_total_students    NUMBER;
BEGIN
   -- Calculate number of students enrolled
   SELECT COUNT(*)
     INTO v_total_students
     FROM enrollment
    WHERE section_id = v_section_id;

   IF v_total_students >= 10
   THEN
      RAISE_APPLICATION_ERROR
         (-20000, 'There are too many students for section '||v_section_id);
   ELSE
      DBMS_OUTPUT.PUT_LINE
         ('There are '||v_total_students||' students in section '||v_section_id);
   END IF;
END;
```

In this version of the script, the RAISE_APPLICATON_ERROR statement is used to handle the following error condition: If the number of students enrolled for a particular section is equal to or greater than 10, the error is raised. It is important to remember that the RAISE_APPLICATION_ERROR statement works with the unnamed user-defined exceptions. Therefore, there is no reference to the exception e_too_many_students anywhere in this script. On the other hand, an error number has been associated with the error message.

When run, this exercise produces the following output (the same section IDs are used for this script as well: 101, 116, and 999).

When 101 is provided for the section ID (this section has more than 10 students), this script produces output as follows:

```
ORA-20000: There are too many students for section 101
ORA-06512: at line 13
```

Note that the error message generated by the RAISE_APPLICATION_ERROR statement appears in the Script Output window and not in the Dbms Output window when this script is run in Oracle SQL Developer.

When 116 is provided for the section ID (this section has less than 10 students), this script produces different output:

```
There are 8 students in section 116
```

For non-existent section ID, the script produces this output:

```
There are 0 students in section 999
```

2. Create the following script. Try to add a record to the INSTRUCTOR table without providing values for the columns CREATED_BY, CREATED_DATE, MODIFIED_BY, and MODIFIED_DATE. Define an exception and associate it with the Oracle error number, so that the error generated by the INSERT statement is handled.

Answer: Consider the following script. Notice that this initial version does not have any exception handlers:

For Example *ch10_10a.sql*

```
BEGIN
   INSERT INTO instructor
      (instructor_id, salutation, first_name, last_name, street_address, zip, phone)
   VALUES
      (INSTRUCTOR_ID_SEQ.NEXTVAL, 'Mr', 'John', 'Smith', '123 Main St.', '00914'
      ,'1234567890');
   COMMIT;
END;
```

When run, this version of the script causes error message as shown below:

```
ORA-01400: cannot insert NULL into ("STUDENT"."INSTRUCTOR"."CREATED_BY")
ORA-06512: at line 2
```

This error message states that a `NULL` value cannot be inserted in to the column `CREATED_BY` of the `INSTRUCTOR` table. Therefore, in order for the script to terminate gracefully (i.e. handle this error), an exception section must be added as shown. All newly added statements are highlighted in bold.

For Example *ch10_10b.sql*

```
DECLARE
   e_non_null_value EXCEPTION;
   PRAGMA EXCEPTION_INIT(e_non_null_value, -1400);
BEGIN
   INSERT INTO instructor
      (instructor_id, salutation, first_name, last_name, street_address, zip, phone)
   VALUES
      (INSTRUCTOR_ID_SEQ.NEXTVAL, 'Mr', 'John', 'Smith', '123 Main St.', '00914'
      ,'1234567890');
   COMMIT;
EXCEPTION
   WHEN e_non_null_value
   THEN
      DBMS_OUTPUT.PUT_LINE
         ('A NULL value cannot be inserted. '||
          'Check constraints on the INSTRUCTOR table.');
END;
```

In this version of the script, a new user-defined exception, `e_non_null_value`, is associated with Oracle error number. As a result, an exception-handling section is able to trap it when this error generated by Oracle. When run, the new version produces the following output:

```
A NULL value cannot be inserted. Check constraints on the INSTRUCTOR table.
```

3. Modify the script created in the previous exercise. Instead of declaring a user-defined exception, add the `OTHERS` exception handler to the exception-handling section of the block. Then display the error number and the error message on the screen.

Answer: The script should look similar to the following. All changes are shown in bold letters.

For Example *ch10_10c.sql*

```
BEGIN
   INSERT INTO instructor
      (instructor_id, salutation, first_name, last_name, street_address, zip, phone)
   VALUES
      (INSTRUCTOR_ID_SEQ.NEXTVAL, 'Mr', 'John', 'Smith', '123 Main St.', '00914'
      ,'1234567890');
   COMMIT;
EXCEPTION
   WHEN OTHERS
   THEN
      DBMS_OUTPUT.PUT_LINE ('Error code: '||SQLCODE);
      DBMS_OUTPUT.PUT_LINE ('Error message: '||SUBSTR(SQLERRM, 1, 200));
END;
```

Notice that as long as the `OTHERS` exception handler is used, there is no need associate an Oracle error number with a user-defined exception. When run, this version of the script produces output as shown:

```
Error code: -1400
Error message: ORA-01400: cannot insert NULL into
("STUDENT"."INSTRUCTOR"."CREATED_BY")
```