Database Programming with PL/SQL

7-2: Trapping Oracle Server Exceptions

Practice Activities

Vocabulary

Identify the vocabulary word for each definition below:

Predefined Oracle Server Errors	Each of these has a predefined name. 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 exceptionname NO_DATA_FOUND.
PRAGMA EXCEPTION_INIT	Tells the compiler to associate an exception name with an Oracle error number. That allows you to refer to any Oracle Server exception by name and to write a specific handler for it.
SQLERRM	Returns character data containing the message associated with the error number
Non-predefined Oracle Server Errors	Each of these has a standard Oracle error number (ORA-nnnnn) and error message, but not a predefined name. We declare our own names for these so that we can reference these names in the exception section.
SQLCODE	Returns the numeric value for the error code (You can assign it to a NUMBER variable.)

Try It / Solve It

1. What are the three types of exceptions that can be handled in a PL/SQL block?

Predefined Oracle server error

Non-predefined Oracle server error

User-defined error

2. What is the difference in how each of these three types of exceptions is handled in the PL/SQL block?

   En las predefinidas por Oracle no necesitas declarar las excepciones

   En las no predefinidas, tu las tienes que declarar y asignarle un nombre.

   En las definidas por el programador, tu las declaras y les asignas nombre.

3. Enter and run the following PL/SQL block. Look at the output and answer the following questions:

```
DECLARE   v_number
    NUMBER(6, 2) := 100;   v_region_id
    regions.region_id%TYPE;   v_region_name
regions.region_name%TYPE;
BEGIN
  SELECT region_id, region_name INTO v_region_id, v_region_name
    FROM regions
    WHERE region_id = 1;
  DBMS_OUTPUT.PUT_LINE('Region: ' || v_region_id || ' is: ' || v_region_name);
v_number := v_number / 0;
END;
```

A. What error message is displayed and why?

   Marca un error, porque no existe una región con el id 1;

B. Modify the block to handle this exception and re-run your code. Now what happens and why?

```
DECLARE
  v_number      NUMBER(6, 2) := 100;
  v_region_id   regions.region_id%TYPE;
  v_region_name  regions.region_name%TYPE;
BEGIN
   SELECT region_id, region_name INTO v_region_id, v_region_name
   FROM regions
   WHERE region_id = 1;
   DBMS_OUTPUT.PUT_LINE('Region: ' || v_region_id || ' is: ' || v_region_name);   v_number
:= v_number / 0;
   EXCEPTION
   WHEN NO_DATA_FOUND THEN
   dbms_output.put_line('No se encontro ninguna region con ese id ');
END;
```

C. Modify the block again to change the WHERE clause to region_id = 29. Re-run the block. Now what happens and why?

```
DECLARE

  v_number      NUMBER(6, 2) := 100;

  v_region_id   regions.region_id%TYPE;

  v_region_name  regions.region_name%TYPE;

BEGIN

   SELECT region_id, region_name INTO v_region_id, v_region_name

   FROM regions

   WHERE region_id = 29;

   DBMS_OUTPUT.PUT_LINE('Region: ' || v_region_id || ' is: ' || v_region_name);   v_number
:= v_number / 0;

   EXCEPTION

   WHEN NO_DATA_FOUND THEN
```

```
    dbms_output.put_line('No se encontro ninguna region con ese id ');
END;
```

Marca error porque se esta dividiendo una variable entre cero.

D. Modify the block again to handle the latest exception and re-run your code.

```
DECLARE
  v_number      NUMBER(6, 2) := 100;
  v_region_id    regions.region_id%TYPE;
  v_region_name  regions.region_name%TYPE;
BEGIN
   SELECT region_id, region_name INTO v_region_id, v_region_name
   FROM regions
   WHERE region_id = 21;
   DBMS_OUTPUT.PUT_LINE('Region: ' || v_region_id || ' is: ' || v_region_name);
   v_number := v_number / 0;
   EXCEPTION
   WHEN NO_DATA_FOUND THEN
   dbms_output.put_line('No se encontro ninguna region con ese id ');
   WHEN zero_divide THEN
   dbms_output.put_line('Se esta haciendo una division entre cero ');
END;
```

4. Enter and run the following PL/SQL block. Look at the output and answer the following questions:

```
DECLARE
   CURSOR regions_curs IS
   SELECT * FROM regions
     WHERE region_id < 20      ORDER BY
region_id;   regions_rec
```

```
        regions_curs%ROWTYPE;   v_count
    NUMBER(6);
    BEGIN
      LOOP
        FETCH regions_curs INTO regions_rec;
        EXIT WHEN regions_curs%NOTFOUND;
        DBMS_OUTPUT.PUT_LINE('Region: ' || regions_rec.region_id
                      || ' Name: ' || regions_rec.region_name);
    END LOOP;
      CLOSE regions_curs;
      SELECT COUNT(*) INTO v_count
        FROM regions
        WHERE region_id = 1;
      DBMS_OUTPUT.PUT_LINE('The number of regions is: ' || v_count);
    END;
```

A. What happens and why?

```
ORA-01001: invalid cursor
```

Falta abrir el cursor.

B. Modify the block to handle the exception and re-run your code.

```
DECLARE
   CURSOR regions_curs IS
   SELECT * FROM regions
     WHERE region_id < 20
 ORDER BY region_id;
regions_rec        regions_curs%ROWTYPE;
  v_count    NUMBER(6);
BEGIN
  LOOP
```

```
    FETCH regions_curs INTO regions_rec;
    EXIT WHEN regions_curs%NOTFOUND;
    DBMS_OUTPUT.PUT_LINE('Region: ' || regions_rec.region_id  || ' Name: ' ||
regions_rec.region_name);
END LOOP;
  CLOSE regions_curs;
  SELECT COUNT(*) INTO v_count
    FROM regions
    WHERE region_id = 1;
  DBMS_OUTPUT.PUT_LINE('The number of regions is: ' || v_count);

EXCEPTION
WHEN InVALID_CURSOR THEN
dbms_output.put_line('Cursor invalido');
END;
```

C. Modify the block again to add an OPEN statement for the cursor, and re-run your code. Now what happens and why? Remember that region_id = 1 does not exist.

```
DECLARE
   CURSOR regions_curs IS
   SELECT * FROM regions
     WHERE region_id < 20
 ORDER BY region_id;
regions_rec        regions_curs%ROWTYPE;
  v_count    NUMBER(6);
BEGIN
OPEN regions_curs;
  LOOP
   FETCH regions_curs INTO regions_rec;
   EXIT WHEN regions_curs%NOTFOUND;
   DBMS_OUTPUT.PUT_LINE('Region: ' || regions_rec.region_id  || ' Name: ' ||
regions_rec.region_name);
END LOOP;
  CLOSE regions_curs;
  SELECT COUNT(*) INTO v_count
   FROM regions
   WHERE region_id = 1;
  DBMS_OUTPUT.PUT_LINE('The number of regions is: ' || v_count);

EXCEPTION
WHEN InVALID_CURSOR THEN
dbms_output.put_line('Cursor invalido');
END;
```

El Código funciona.

5. Oracle Server Errors:

A. Add an exception handler to the following code to trap the following predefined Oracle Server errors: NO_DATA_FOUND, TOO_MANY_ROWS, and DUP_VAL_ON_INDEX.

```
DECLARE   v_language_id

            languages.language_id%TYPE;

v_language_name   languages.language_name%TYPE;
```

```
BEGIN
  SELECT language_id, language_name INTO v_language_id, v_language_name
    FROM languages
    WHERE LOWER(language_name) LIKE '<substring%>'; -- for example 'ab%'
  INSERT INTO languages(language_id, language_name)
VALUES(80, null);
EXCEPTION
WHEN NO_DATA_FOUND THEN
dbms_output.put_line('No se encontro ningun dato');

WHEN TOO_MANY_ROWS THEN
dbms_output.put_line('Demasiadas filas');

WHEN DUP_VAL_ON_INDEX THEN
dbms_output.put_line('ID duplicado en el indice');
END;
```

B. Test your block twice using each of the following language substrings: ba, ce. There are several language_names beginning with “Ba,” but none beginning with “Ce”.

```
 DECLARE   v_language_id              languages.language_id%TYPE;   v_language_name
languages.language_name%TYPE;
BEGIN
  SELECT language_id, language_name INTO v_language_id, v_language_name
    FROM languages
    WHERE LOWER(language_name) LIKE '<Ce%>'; -- for example 'ab%'
  INSERT INTO languages(language_id, language_name)     VALUES(80, null);
EXCEPTION
WHEN NO_DATA_FOUND THEN
dbms_output.put_line('No se encontro ningun dato');

WHEN TOO_MANY_ROWS THEN
```

```
dbms_output.put_line('Demasiadas filas');

WHEN DUP_VAL_ON_INDEX THEN
dbms_output.put_line('ID duplicado en el indice');
END;
```

Now test your block a third time using substring: al. There is exactly one language_name beginning with "Al". Note that language_id 80 (Arabic) already exists. Explain the output.

```
DECLARE   v_language_id              languages.language_id%TYPE;   v_language_name
languages.language_name%TYPE;

BEGIN

  SELECT language_id, language_name INTO v_language_id, v_language_name

    FROM languages

    WHERE LOWER(language_name) LIKE '<Al%>'; -- for example 'ab%'

  INSERT INTO languages(language_id, language_name)     VALUES(80, null);

EXCEPTION

WHEN NO_DATA_FOUND THEN

dbms_output.put_line('No se encontro ningun dato');


WHEN TOO_MANY_ROWS THEN

dbms_output.put_line('Demasiadas filas');


WHEN DUP_VAL_ON_INDEX THEN

dbms_output.put_line('ID duplicado en el indice');

END;
```

C. Now (keeping the substring as “al”), add a non_predefined exception handler to trap the ORA01400 exception. Name your exception e_null_not_allowed. Rerun the code and observe the results.

```
DECLARE
  v_language_id     languages.language_id%TYPE;
  v_language_name   languages.language_name%TYPE;

e_null_not_allowed EXCEPTION;
pragma exception_init(e_null_not_allowed,-01400);

BEGIN
  SELECT language_id, language_name INTO v_language_id, v_language_name
    FROM languages
    WHERE LOWER(language_name) LIKE 'Al%'; -- for example 'ab%'
  INSERT INTO languages(language_id, language_name)     VALUES(80, null);
EXCEPTION
WHEN e_null_not_allowed THEN
dbms_output.put_line('No se');



WHEN NO_DATA_FOUND THEN
dbms_output.put_line('No se encontro ningun dato');

WHEN TOO_MANY_ROWS THEN
dbms_output.put_line('Demasiadas filas');

WHEN DUP_VAL_ON_INDEX THEN
dbms_output.put_line('ID duplicado en el indice');

END;
```

Extension exercise

1. In preparation for this exercise, run the following SQL statement to create an error-logging table:

```
CREATE TABLE error_log
    (who            VARCHAR2(30),      when
            DATE,  error_code
    NUMBER(6),  error_message
    VARCHAR2(255));
```

Modify your PL/SQL block from question 5 to remove the four explicit exception handlers, replacing them with a single WHEN OTHERS handler. The handler should INSERT a row into the error_log table each time an exception is raised and handled. The row should consist of the Oracle username (who), when the error was raised (when), and the SQLCODE and SQLERRM of the exception. Test your block several times, with different data values to raise each of the four kinds of exceptions handled in the block. Finally, SELECT from the error-logging table to check that the rows have been inserted.