ORACLE Academy

academy.oracle.com

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

1.
Predefined oracle server error

Non-predefined oracle server error

User-defined error

2.
Predefined oracle server errors are errors that have names and can be called in the exception block.

Non-predefined oracle server errors are errors that do not have names but have codes and messages and can be given names by the user in the declaration section and called in the exception block.

User-defined errors are either non-predefined oracle server errors named by user or not errors but situations that the user considers as an error and that shouldn't happen.

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? Because we try to divide v_number by 0. When there's an error that is not handled, the entire program is not executed.

ORA-01476: divisor is equal to zero

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

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

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

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
We didn't open the cursor so we can't fetch and close it.

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

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.

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);
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”.

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.

   C. Now (keeping the substring as “al”), add a non_predefined exception handler to trap the ORA-01400 exception. Name your exception e_null_not_allowed. Rerun the code and observe the results.

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.