

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 function | 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 exceptions, non-predefined exceptions, user-defined error.

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

Exceptiile predefinite sunt declarate de Oracle Server, cele non-predefinite sunt declarate in sectiunea de declaratii, nefiind declarate de Oracle Server, cele definite de utilizator asa cum le spune numele sunt declarate de programator in sectiunea de declaratii si sunt ridicate explicit.

Copyright © 2020, Oracle and/or its affiliates. All rights reserved. Oracle and Java are registered trademarks of Oracle and/or its affiliates. Other names may be trademarks of their respective owners.

1. 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;

1. What error message is displayed and why?

Apare tipul de eroare NO DATA FOUND pentru ca nu avem date pentru blocul dat.

1. 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 data found');

END;

1. 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 data found');

END;

**Apare o eroare de tipul divisor is equal to zero .**

1. 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 = 29;

DBMS\_OUTPUT.PUT\_LINE('Region: ' || v\_region\_id || ' is: ' || v\_region\_name);

v\_number := v\_number / 0;

EXCEPTION

WHEN ZERO\_DIVIDE THEN

DBMS\_OUTPUT.PUT\_LINE('No data found');

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;

1. What happens and why?

Apare o eroare de tipul invalid cursor fiindca acesta nu a fost deschis.

1. 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 invalid');

END;

1. 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.

ECLARE

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 invalid');

END;

Aici se vor afisa regiunile care au region\_id mai mic decat 20, iar apoi numarul de regiuni. Numarul de regiuni va fi 0 fiindca numaram regiunile care au region\_id = 1.

2

5. Oracle Server Errors:

1. 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;

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 'ab%'; -- for example 'ab%'

INSERT INTO languages(language\_id, language\_name)

VALUES(80, null);

EXCEPTION

WHEN NO\_DATA\_FOUND THEN

DBMS\_OUTPUT.PUT\_LINE('NO data found!');

WHEN TOO\_MANY\_ROWS THEN

DBMS\_OUTPUT.PUT\_LINE('TOO many rows!');

WHEN DUP\_VAL\_ON\_INDEX THEN

DBMS\_OUTPUT.PUT\_LINE('DUP\_VAL\_ON\_INDEX');

END;

1. 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.

Apare excepția “No data found!” și de asemenea și faptul că s-a inserat o linie.

1. 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, -1400);

BEGIN

SELECT language\_id, language\_name INTO v\_language\_id, v\_language\_name FROM languages

WHERE LOWER(language\_name) LIKE 'al%'; -- for example 'Al%'

INSERT INTO languages(language\_id, language\_name)

VALUES(80, null);

EXCEPTION

WHEN NO\_DATA\_FOUND THEN

DBMS\_OUTPUT.PUT\_LINE('NO data found!');

WHEN e\_null\_not\_allowed

THEN

DBMS\_OUTPUT.PUT\_LINE('ATENTIE EXCEPTION!');

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.

DECLARE

v\_language\_id languages.language\_id%TYPE;

v\_language\_name languages.language\_name%TYPE;

v\_sqlcode NUMBER(5);

v\_sqlerrm VARCHAR2(255);

BEGIN

SELECT language\_id, language\_name

INTO v\_language\_id, v\_language\_name

FROM languages

WHERE lower(language\_name) LIKE 'al%';

INSERT INTO languages(language\_id, language\_name)

VALUES(80, null);

EXCEPTION

WHEN OTHERS THEN

v\_sqlcode := SQLCODE;

v\_sqlerrm := SQLERRM;

INSERT INTO error\_log(who, when, error\_code, error\_message)

VALUES(USER, SYSDATE, v\_sqlcode, v\_sqlerrm);

END;

SELECT \*

FROM error\_log;

.

3