
ORACLE
ACADEMY

Database Programming with PL/SQL

7-4
Recognizing the Scope of Exceptions

ORACLE ACADEMY

Objectives

This lesson covers the following objectives:

- Describe the scope of an exception
- Recognize an exception-scope issue when an exception is within nested blocks
- Describe the effect of exception propagation in nested blocks

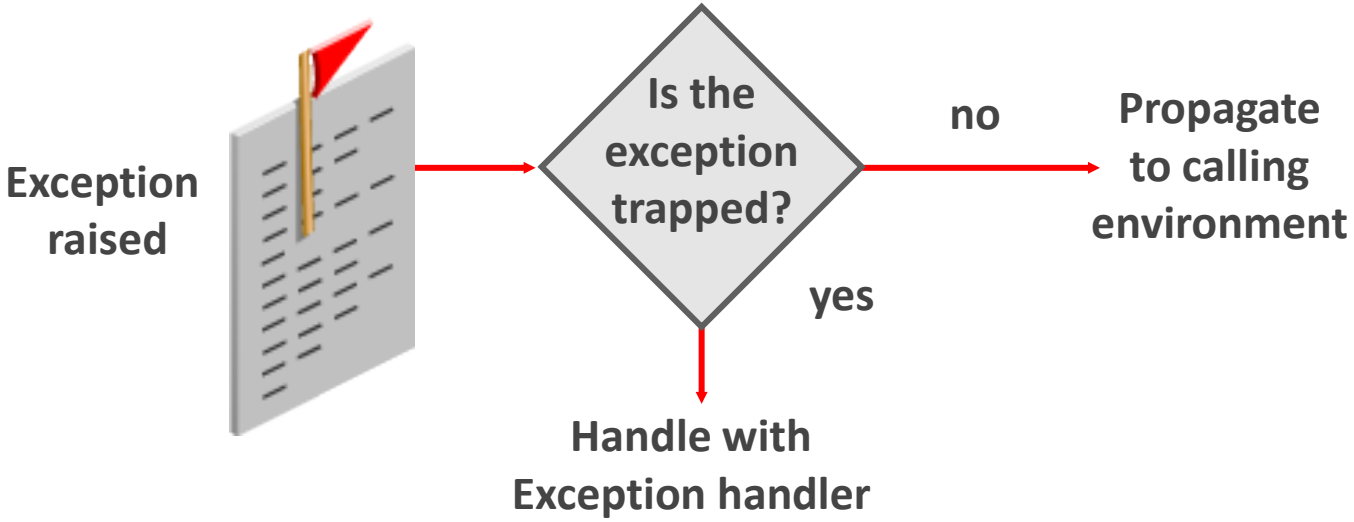
Purpose

- You learned about nested blocks and scope of variables in an earlier lesson.
- An exception is a PL/SQL variable; therefore, it follows the same scoping and visibility rules as any other kind of variable.
- To handle exceptions correctly, you must understand the scope and visibility of exception variables.
- This is particularly important when using nested blocks.

Exception Handling in Nested Blocks

You can deal with an exception by:

- Handling it ("trapping it") in the block in which it occurs, or
- Propagating it to the calling environment (which can be a higher-level block)



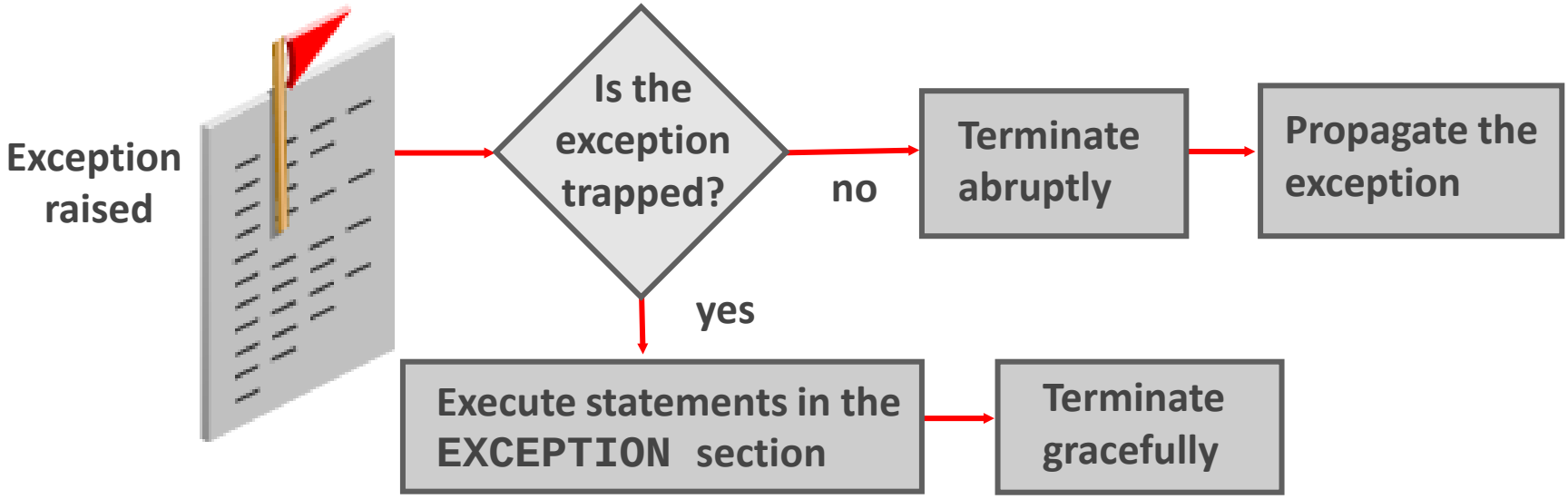
Handling Exceptions in an Inner Block

- In this example, an error occurs during the execution of the inner block.
- The inner block's EXCEPTION section deals with the exception successfully, and PL/SQL considers that this exception is now finished.
- The outer block resumes execution as normal.

```
BEGIN  -- outer block
  ...
  BEGIN -- inner block
   ...  -- exception_name occurs here
   ...
  EXCEPTION
    WHEN exception_name THEN -- handled here
    ...
  END; -- inner block terminates successfully
  ...  -- outer block continues execution
END;
```

Propagating Exceptions to an Outer Block

If the exception is raised in the executable section of the inner block and no corresponding exception handler exists, the PL/SQL block terminates with failure and the exception is propagated to an enclosing block.



Propagating Exceptions to an Outer Block

- In this example, an exception occurs during the execution of the inner block.
- The inner block's EXCEPTION section does not deal with the exception.

```
DECLARE    -- outer block
  e_no_rows     EXCEPTION;
BEGIN
  BEGIN    -- inner block
    IF ... THEN RAISE e_no_rows; -- exception occurs here
    ...
  END;   -- Inner block terminates unsuccessfully
  ...    -- Remaining code in outer block's executable
  ...    -- section is skipped
  EXCEPTION
    WHEN e_no_rows THEN – outer block handles the exception
    ...
END;
```

Propagating Exceptions to an Outer Block

- The inner block terminates unsuccessfully and PL/SQL passes (propagates) the exception to the outer block.
- The outer block's EXCEPTION section successfully handles the exception.

```
DECLARE    -- outer block
  e_no_rows     EXCEPTION;
BEGIN
  BEGIN    -- inner block
    IF ... THEN RAISE e_no_rows; -- exception occurs here
    ...
  END;   -- Inner block terminates unsuccessfully
  ...    -- Remaining code in outer block's executable
  ...    -- section is skipped
  EXCEPTION
    WHEN e_no_rows THEN – outer block handles the exception
    ...
END;
```

Propagating Exceptions from a Sub-Block

- If a PL/SQL raises an exception and the current block does not have a handler for that exception, the exception propagates to successive enclosing blocks until it finds a handler.
- When the exception propagates to an enclosing block, the remaining executable actions in that block are bypassed.
- One advantage of this behavior is that you can enclose statements that require their own exclusive error handling in their own block, while leaving more general exception handling (for example WHEN OTHERS) to the enclosing block.
- The next slide shows an example of this.

Propagating Predefined Oracle Server Exceptions from a Sub-Block

- `Employee_id` 999 does not exist.
- What is displayed when this code is executed?

```
DECLARE
  v_last_name    employees.last_name%TYPE;
BEGIN
  BEGIN
    SELECT last_name INTO v_last_name
      FROM employees WHERE employee_id = 999;
    DBMS_OUTPUT.PUT_LINE('Message 1');
  EXCEPTION
    WHEN TOO_MANY_ROWS THEN
      DBMS_OUTPUT.PUT_LINE('Message 2');
  END;
  DBMS_OUTPUT.PUT_LINE('Message 3');
EXCEPTION
  WHEN OTHERS THEN
    DBMS_OUTPUT.PUT_LINE('Message 4');
END;
```

Propagating User-named Exceptions: Ex. 1

What happens when this code is executed?

```
BEGIN
  DECLARE
    e_myexcep    EXCEPTION;
  BEGIN
    RAISE e_myexcep;
    DBMS_OUTPUT.PUT_LINE('Message 1');
  EXCEPTION
    WHEN TOO_MANY_ROWS THEN
      DBMS_OUTPUT.PUT_LINE('Message 2');
  END;
  DBMS_OUTPUT.PUT_LINE('Message 3');
EXCEPTION
  WHEN e_myexcep THEN
    DBMS_OUTPUT.PUT_LINE('Message 4');
END;
```

Scope of Exception Names

- Predefined Oracle server exceptions, such as NO_DATA_FOUND, TOO_MANY_ROWS, and OTHERS are not declared by the programmer.
- They can be raised in any block and handled in any block.
- User-named exceptions (non-predefined Oracle server exceptions and user-defined exceptions) are declared by the programmer as variables of type EXCEPTION.
- They follow the same scoping rules as other variables.

Scope of Exception Names

- Therefore, a user-named exception declared within an inner block cannot be referenced in the exception section of an outer block.
- To avoid this, always declare user-named exceptions in the outermost block.

Propagating User-named Exceptions: Ex. 2

Now what happens when this code is executed?

```
DECLARE
  e_myexcep    EXCEPTION;
BEGIN
  BEGIN
    RAISE e_myexcep;
    DBMS_OUTPUT.PUT_LINE('Message 1');
  EXCEPTION
    WHEN TOO_MANY_ROWS THEN
      DBMS_OUTPUT.PUT_LINE('Message 2');
  END;
  DBMS_OUTPUT.PUT_LINE('Message 3');
EXCEPTION
  WHEN e_myexcep THEN
    DBMS_OUTPUT.PUT_LINE('Message 4');
END;
```

Propagating User-named Exceptions: Ex. 3

What happens when this code is executed?

```
DECLARE
  e_myexcep    EXCEPTION;
BEGIN
  BEGIN
    RAISE e_myexcep;
    DBMS_OUTPUT.PUT_LINE('Message 1');
  EXCEPTION
    WHEN TOO_MANY_ROWS THEN
      DBMS_OUTPUT.PUT_LINE('Message 2');
  END;
  DBMS_OUTPUT.PUT_LINE('Message 3');
EXCEPTION
  WHEN NO_DATA_FOUND THEN
    DBMS_OUTPUT.PUT_LINE('Message 4');
END;
```

Propagating Unhandled Exceptions to the Calling Environment

- If a raised exception is not handled in any block, the outermost block is exited with the exception still raised.
- The calling environment, for example Application Express, must then try to handle the exception.
- Because Application Express is Oracle software and therefore understands PL/SQL exceptions, Application Express will display an error message.

Propagating Unhandled Exceptions to the Calling Environment

- But other applications cannot always do this, and may fail with unexpected errors.
- To avoid this, always handle exceptions within PL/SQL.
- One way to guarantee this is to always include a WHEN OTHERS handler in the outermost block.

Terminology

Key terms used in this lesson included:

- Exception scope
- Exception visibility
- Propagation of exceptions

Summary

In this lesson, you should have learned how to:

- Describe the scope of an exception
- Recognize an exception-scope issue when an exception is within nested blocks
- Describe the effect of exception propagation in nested blocks


ORACLE
ACADEMY