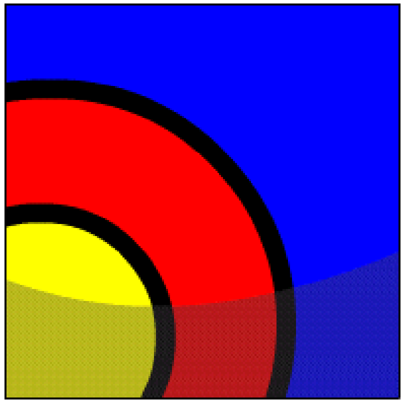
Nested Blocks and Variable Scope

What Will I Learn?



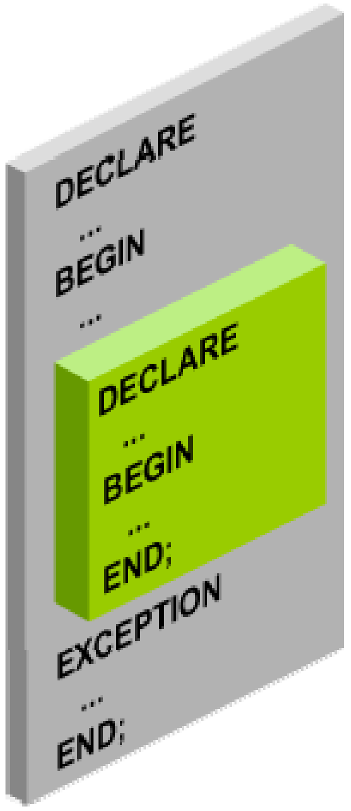
In this lesson, you will learn to:

- Understand the scope and visibility of variables
- Write nested blocks and qualify variables with labels
- Understand the scope of an exception
- Describe the effect of exception propagation in nested blocks

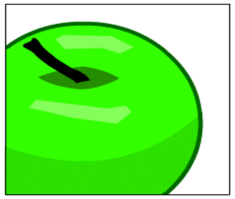
Why Learn It?

A large, complex block can be hard to understand. You can break it down into smaller blocks that are nested one inside the other, making the code easier to read and correct.

When you nest blocks, the declared variables might not be available depending on their scope and visibility. You can make invisible variables available by using labels.



Tell Me / Show Me



Nested Blocks

The example shown in the slide has an outer (parent) block (illustrated in normal text) and a nested (child) block (illustrated in bold text). The variable `v_outer_variable` is declared in the outer block and the variable `v_inner_variable` is declared in the inner block.

```
DECLARE
  v_outer_variable VARCHAR2(20):='GLOBAL VARIABLE';
BEGIN
  DECLARE
    v_inner_variable VARCHAR2(20):='LOCAL
VARIABLE';
  BEGIN
    DBMS_OUTPUT.PUT_LINE(v_inner_variable);
    DBMS_OUTPUT.PUT_LINE(v_outer_variable);
  END;
  DBMS_OUTPUT.PUT_LINE(v_outer_variable);
END;
```

Tell Me / Show Me

Variable Scope

The scope of a variable is the block or blocks in which the variable is accessible, that is, it can be named and used. In PL/SQL, a variable's scope is the block in which it is declared plus all blocks nested within the declaring block.

What are the scopes of the two variables declared in this example?

```
DECLARE
  v_outer_variable VARCHAR2(20):='GLOBAL VARIABLE';
BEGIN
  DECLARE
    v_inner_variable VARCHAR2(20):='LOCAL VARIABLE';
  BEGIN
    DBMS_OUTPUT.PUT_LINE(v_inner_variable);
    DBMS_OUTPUT.PUT_LINE(v_outer_variable);
  END;
  DBMS_OUTPUT.PUT_LINE(v_outer_variable);
END;
```

Tell Me / Show Me

Variable Scope

Examine the following code. What is the scope of each of the variables?

```
DECLARE
  v_father_name   VARCHAR2(20):='Patrick';
  v_date_of_birth DATE:='20-Apr-1972';
BEGIN
  DECLARE
    v_child_name  VARCHAR2(20):='Mike';
  BEGIN
    DBMS_OUTPUT.PUT_LINE('Father''s Name: '||v_father_name);
    DBMS_OUTPUT.PUT_LINE('Date of Birth: '||v_date_of_birth);
    DBMS_OUTPUT.PUT_LINE('Child''s Name: '||v_child_name);
  END;
  DBMS_OUTPUT.PUT_LINE('Date of Birth: '||v_date_of_birth);
END;
```

Tell Me / Show Me

Local and Global Variables

Variables declared in a PL/SQL block are considered local to that block and global to all its subblocks. `v_outer_variable` is local to the outer block but global to the inner block. When you access this variable in the inner block, PL/SQL first looks for a local variable in the inner block with that name. If there are no similarly named variables, PL/SQL looks for the variable in the outer block.

```
DECLARE
  v_outer_variable VARCHAR2(20):='GLOBAL VARIABLE';
BEGIN
  DECLARE
    v_inner_variable VARCHAR2(20):='LOCAL
VARIABLE';
  BEGIN
    DBMS_OUTPUT.PUT_LINE(v_inner_variable);
    DBMS_OUTPUT.PUT_LINE(v_outer_variable);
  END;
  DBMS_OUTPUT.PUT_LINE(v_outer_variable);
END;
```

Tell Me / Show Me

Local and Global Variables

The `v_inner_variable` variable is local to the inner block and is not global because the inner block does not have any nested blocks. This variable can be accessed only within the inner block. If PL/SQL does not find the variable declared locally, it looks upward in the declarative section of the parent blocks. PL/SQL does not look downward in the child blocks.

```
DECLARE
  v_outer_variable VARCHAR2(20):='GLOBAL VARIABLE';
BEGIN
  DECLARE
    v_inner_variable VARCHAR2(20):='LOCAL
VARIABLE';
  BEGIN
    DBMS_OUTPUT.PUT_LINE(v_inner_variable);
    DBMS_OUTPUT.PUT_LINE(v_outer_variable);
  END;
  DBMS_OUTPUT.PUT_LINE(v_outer_variable);
END;
```

ORACLE Academy

Tell Me / Show Me

Variable Scope

The variables `v_father_name` and `v_date_of_birth` are declared in the outer block. They are local to the outer block and global to the inner block. Their scope includes both blocks.

```
DECLARE
  v_father_name   VARCHAR2(20):='Patrick';
  v_date_of_birth DATE:='20-Apr-1972';
BEGIN
  DECLARE
    v_child_name  VARCHAR2(20):='Mike';
   ...
```

The variable `v_child_name` is declared in the inner (nested) block. This variable is accessible only within the nested block and is not accessible in the outer block.

Tell Me / Show Me

Variable Naming

You cannot declare two variables with the same name in the same block. However, you can declare variables with the same name in two different blocks (nested blocks). The two items represented by the same name are distinct, and any change in one does not affect the other.

Tell Me / Show Me

Variable Visibility

What if the same name is used for two variables, one in each of the blocks? In this example, the variable `v_date_of_birth` is declared twice.

```
DECLARE
  v_father_name   VARCHAR2(20):='Patrick';
  v_date_of_birth DATE:='20-Apr-1972';
BEGIN
  DECLARE
    v_child_name    VARCHAR2(20):='Mike';
    v_date_of_birth DATE:='12-Dec-2002';
  BEGIN
    DBMS_OUTPUT.PUT_LINE('Date of Birth:'
                        ||v_date_of_birth);
    ...
```

Which `v_date_of_birth` is referenced in the `DBMS_OUTPUT.PUT_LINE` statement?

Tell Me / Show Me

Variable Visibility

The visibility of a variable is the portion of the program where the variable can be accessed without using a qualifier. What is the visibility of each of the variables?

```
DECLARE
  v_father_name   VARCHAR2(20):='Patrick';
  v_date_of_birth DATE:='20-Apr-1972';
BEGIN
  DECLARE
    v_child_name    VARCHAR2(20):='Mike';
    v_date_of_birth DATE:='12-Dec-2002';
  BEGIN
    DBMS_OUTPUT.PUT_LINE('Father''s Name: '||v_father_name);
1   DBMS_OUTPUT.PUT_LINE('Date of Birth:
   '||v_date_of_birth);
    DBMS_OUTPUT.PUT_LINE('Child''s Name: '||v_child_name);
2 END;
  DBMS_OUTPUT.PUT_LINE('Date of Birth: '||v_date_of_birth);
END;
```

Tell Me / Show Me

Variable Visibility

The `v_date_of_birth` variable declared in the outer block has scope even in the inner block. This variable is visible in the outer block. However, it is not visible in the inner block because the inner block has a local variable with the same name. The `v_father_name` variable is visible in the inner and outer blocks. The `v_child_name` variable is visible only in the inner block.

```
DECLARE
  v_father_name   VARCHAR2(20):='Patrick';
  v_date_of_birth DATE:='20-Apr-1972';
BEGIN
  DECLARE
    v_child_name    VARCHAR2(20):='Mike';
    v_date_of_birth DATE:='12-Dec-2002';
  …
```

What if you want to reference the outer block's `v_date_of_birth` within the inner block?

Tell Me / Show Me

Qualifying an Identifier

A qualifier is a label given to a block. You can use this qualifier to access the variables that have scope but are not visible. In this example, the outer block has the label, <<outer>>.

```
<<outer>>
DECLARE
  v_father_name   VARCHAR2(20):='Patrick';
  v_date_of_birth DATE:='20-Apr-1972';
BEGIN
  DECLARE
    v_child_name    VARCHAR2(20):='Mike';
    v_date_of_birth DATE:='12-Dec-2002';
  …
```

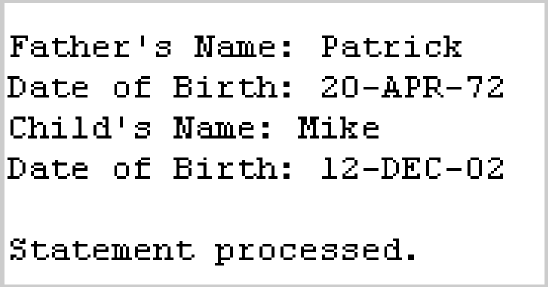
Labeling is not limited to the outer block; you can label any block.

Tell Me / Show Me

Qualifying an Identifier

Using the `outer` label to qualify the `v_date_of_birth` identifier, you can now print the father's date of birth in the inner block.

```
<<outer>>
DECLARE
  v_father_name VARCHAR2(20):='Patrick';
  v_date_of_birth DATE:='20-Apr-1972';
BEGIN
  DECLARE
    v_child_name VARCHAR2(20):='Mike';
    v_date_of_birth DATE:='12-Dec-2002';
  BEGIN
    DBMS_OUTPUT.PUT_LINE('Father''s Name: '||v_father_name);
    DBMS_OUTPUT.PUT_LINE('Date of Birth: '
    ||outer.v_date_of_birth);
    DBMS_OUTPUT.PUT_LINE('Child''s Name: '||v_child_name);
    DBMS_OUTPUT.PUT_LINE('Date of Birth: '||v_date_of_birth);
  END;
END;
```



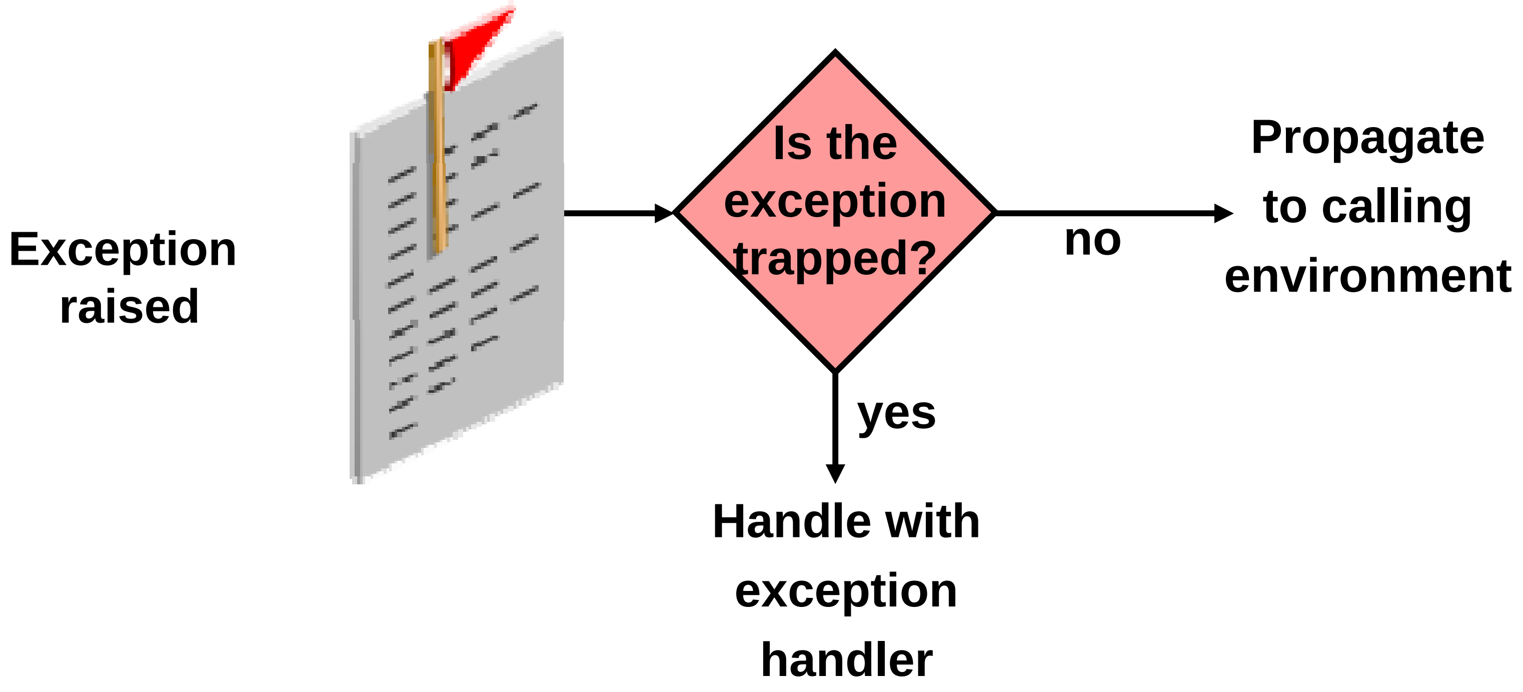
```
Father's Name: Patrick
Date of Birth: 20-APR-72
Child's Name: Mike
Date of Birth: 12-DEC-02

Statement processed.
```

Tell Me / Show Me

Scope of Exceptions in Nested Blocks

You can deal with an exception by:

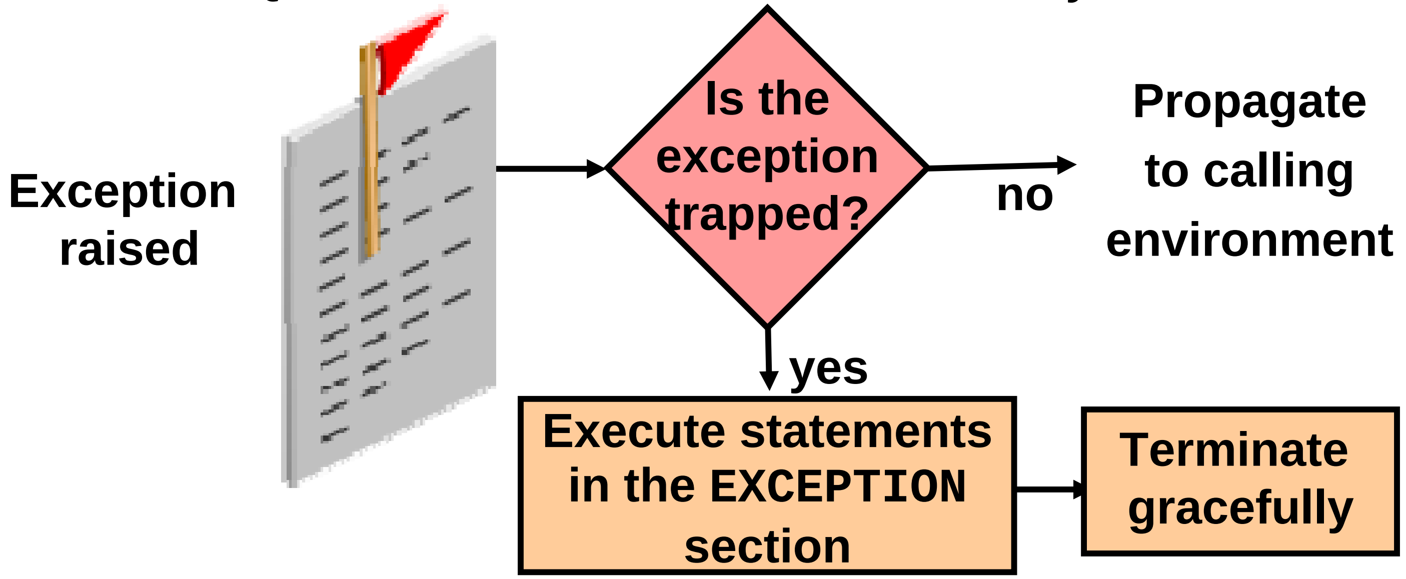
- Handling it (“trapping it”) in the block in which it occurs, or
- Propagating it to the calling environment


Exception raised
Is the exception trapped?
no
Propagate to calling environment
yes
Handle with exception handler

Tell Me / Show Me

Trapping Exceptions with a Handler

Include an `EXCEPTION` section in your PL/SQL program to trap exceptions. If the exception is raised in the executable section of the block, processing is handled by the corresponding exception handler in the exception section of the same block. If PL/SQL successfully handles the exception, then the exception does not propagate to the enclosing block or to the calling environment. The PL/SQL block terminates successfully.


Exception raised
Is the exception trapped?
no
Propagate to calling environment
yes
Execute statements in the EXCEPTION section
Terminate gracefully

Tell Me / Show Me

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.

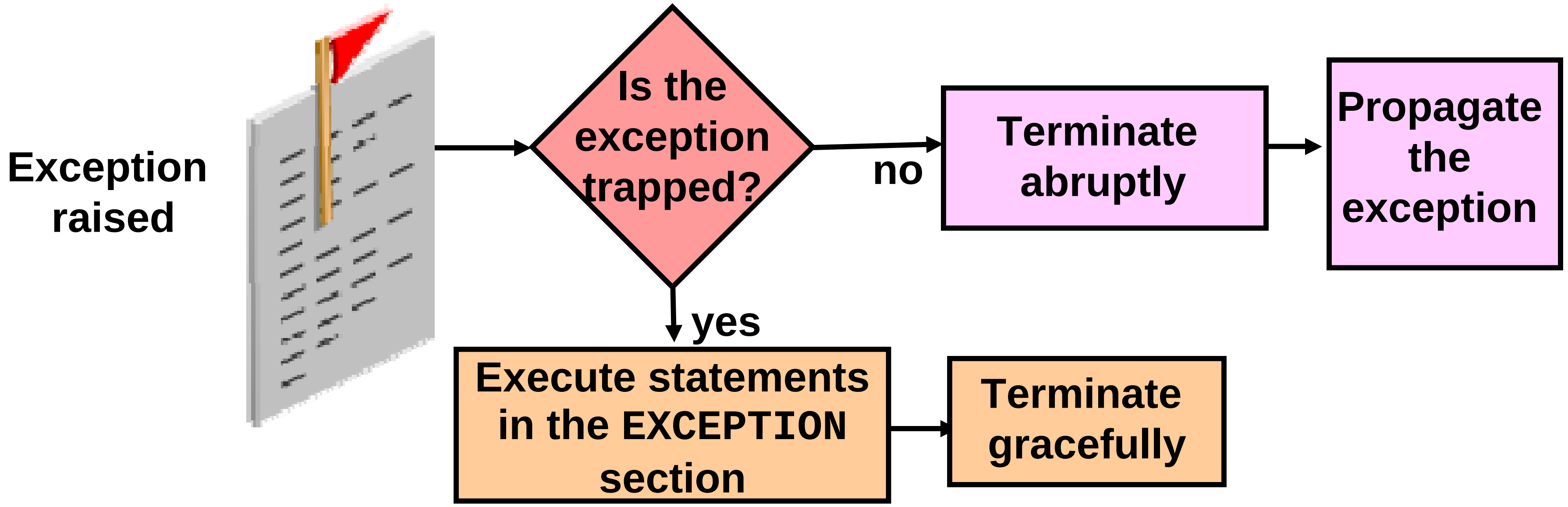
```
BEGIN  -- outer block
  ...

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

  ...  -- outer block continues execution
END;
```

ORACLE Academy

Tell Me / Show Me

Propagating Exceptions to an Outer Block

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



Tell Me / Show Me

Propagating Exceptions to an Outer Block

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

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

```
BEGIN  -- outer block
  ...
  BEGIN -- inner block
   ...   -– exception_name occurs here
   ...
  END; -- inner block terminates unsuccessfully

  ...   -- Remaining code in outer block’s executable
  ...   -- section is skipped
  EXCEPTION
    WHEN exception_name THEN – outer block handles the exception
    ...
END;
```

Tell Me / Show Me

Propagating Exceptions in a Subblock

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 to the enclosing block.

If none of these blocks handle the exception, an unhandled exception occurs in the host environment (for example Application Express).

Tell Me / Show Me

Terminology

Key terms used in this lesson include:

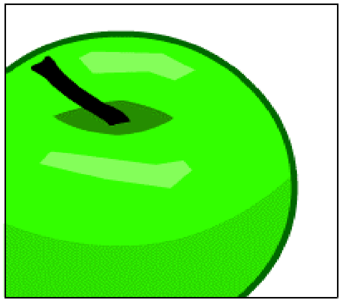
Variable scope

Variable visibility

Qualifier

Exception handling

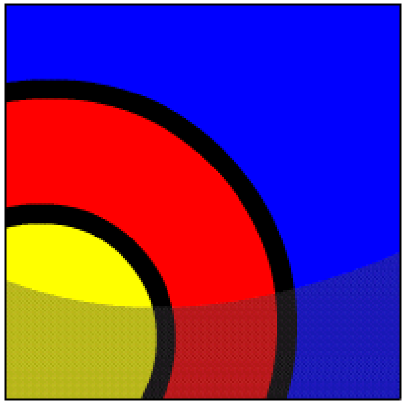
Exception propagating



Summary

In this lesson, you have learned how to:

- Understand the scope and visibility of variables
- Write nested blocks and qualify variables with labels
- Understand the scope of an exception
- Describe the effect of exception propagation in nested blocks



Try It / Solve It

The exercises in this lesson cover the following topics:

- Understanding the scope and visibility of variables
- Writing nested blocks and qualifying variables with labels
- Understanding the scope of an exception
- Describing the effect of exception propagation in nested blocks