Nested Blocks and Variable Scope
Objectives

This lesson covers the following objectives:

• Understand the scope and visibility of variables
• Write nested blocks and qualify variables with labels
• Describe the rules for variable scope when a variable is nested in a block
• Recognize a variable scope issue when a variable is used in nested blocks
• Qualify a variable nested in a block with a label
Purpose

• 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, declared variables might not be available depending on their scope and visibility.

• You can make invisible variables available by using block labels.
Nested Blocks

• PL/SQL is a block-structured language.
• The basic units (procedures, functions, and anonymous blocks) are logical blocks, which can contain any number of nested sub-blocks.
• Each logical block corresponds to a problem to be solved.
Nested Blocks Illustrated

• Nested blocks are blocks of code placed within other blocks of code.

• There is an outer block and an inner block.

• You can nest blocks within blocks as many times as you need to; there is no practical limit to the depth of nesting Oracle allows.
Nested Block Example

• The example shown in the slide has an outer (parent) block (illustrated in blue text) and a nested (child) block (illustrated in red text).

• The variable \texttt{v\_outer\_variable} is declared in the outer block and the variable \texttt{v\_inner\_variable} is declared in the inner block.

```plsql
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;
```
Variable Scope

• The scope of a variable is the block or blocks in which the variable is accessible, that is, where it can be 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?

```sql
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;
```
Variable Scope Example

• Examine the following code.

• What is the scope of each of the variables?

```sql
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;
```
Local and Global Variables

- Variables declared in a PL/SQL block are considered local to that block and global to all blocks nested within it.

- `v_outer_variable` is local to the outer block but global to the inner block.

```plsql
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;
```
Local and Global Variables

• 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;
```
Local and Global Variables

• The \texttt{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.

```sql
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;
```
Local and Global Variables

- 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 into the child blocks.

```plsql
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;
```
Variable Scope Accessible to Outer Block

- 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.

```plsql
DECLARE
    v_father_name   VARCHAR2(20):='Patrick';
    v_date_of_birth DATE:='20-Apr-1972';
BEGIN
    DECLARE
        v_child_name  VARCHAR2(20):='Mike';
    ... 
```
Variable Scope Accessible to Outer Block

• The variable `v_child_name` is declared in the inner (nested) block.

• This variable is accessible only within the inner block and is not accessible in the outer block.

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

A Scoping Example

Why will this code not work correctly?

```sql
DECLARE
    v_first_name    VARCHAR2(20);
BEGIN
    DECLARE
        v_last_name     VARCHAR2(20);
    BEGIN
        v_first_name := 'Carmen';
        v_last_name  := 'Miranda';
        DBMS_OUTPUT.PUT_LINE
            (v_first_name || ' ' || v_last_name);
    END;
    DBMS_OUTPUT.PUT_LINE
        (v_first_name || ' ' || v_last_name);
END;
```
A Second Scoping Example

Will this code work correctly? Why or why not?

```sql
DECLARE
    v_first_name   VARCHAR2(20);
    v_last_name    VARCHAR2(20);
BEGIN
    BEGIN
        v_first_name := 'Carmen';
        v_last_name  := 'Miranda';
        DBMS_OUTPUT.PUT_LINE
            (v_first_name || ' ' || v_last_name);
    END;
    DBMS_OUTPUT.PUT_LINE
        (v_first_name || ' ' || v_last_name);
END;
```
Three Levels of Nested Block

What is the scope of each of these variables?

```sql
DECLARE       -- outer block
    v_outervar    VARCHAR2(20);
BEGIN
    DECLARE      -- middle block
        v_middlevar  VARCHAR2(20);
    BEGIN
        BEGIN      -- inner block
            BEGIN
                v_outervar := 'Joachim';
                v_middlevar := 'Chang';
                END;
            END;
        END;
    END;
END;
```
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 when one block is nested within the other block.

• The two items represented by the same name are distinct, and any change in one does not affect the other.
Are the following declarations valid?

```sql
DECLARE               -- outer block
  v_myvar    VARCHAR2(20);
BEGIN
  DECLARE               -- inner block
    v_myvar    VARCHAR2(15);
  BEGIN
    ...
  END;
END;
```

Example of Variable Naming
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.

```plsql
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);
    ...
Variable Visibility

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

```plsql
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);
        ...
```
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?

```sql
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: ' || 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;
```
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.

```sql
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';
    ...
```
Variable Visibility

- 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.
- What if you want to reference the outer block’s `v_date_of_birth` within the inner block?

```sql
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';
    ...
```
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.
• The outer block below is labeled <<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';
    ...
```

• Each nested inner block also can be labeled.
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 using code 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;
```

Statement processed.
Terminology

Key terms used in this lesson included:

• Block label
• Variable scope
• Variable visibility
Summary

In this lesson, you should have learned how to:

• Understand the scope and visibility of variables
• Write nested blocks and qualify variables with labels
• Describe the rules for variable scope when a variable is nested in a block
• Recognize a variable scope issue when a variable is used in nested blocks
• Qualify a variable nested in a block with a label