Writing Executable Statements

Objectives

- After completing this lesson, you should be able to do the following:
  - Identify lexical units in a PL/SQL block
  - Use built-in SQL functions in PL/SQL
  - Describe when implicit conversions take place and when explicit conversions have to be dealt with
  - Write nested blocks and qualify variables with labels
  - Write readable code with appropriate indentations

Lexical Units in a PL/SQL Block

- Lexical units:
  - Are building blocks of any PL/SQL block
  - Are sequences of characters including letters, numerals, tabs, spaces, returns, and symbols
  - Can be classified as:
    - Identifiers
    - Delimiters
    - Literals
    - Comments

PL/SQL Block Syntax and Guidelines

- Literals:
  - Character and date literals must be enclosed in single quotation marks.

```
name := 'Henderson';
```

  - Numbers can be simple values or scientific notation.
- Statements can continue over several lines.

Commenting Code

- Prefix single-line comments with two hyphens (--).
- Place multiple-line comments between the symbols /* and */.

- Example

```
DECLARE
...
annual_sal NUMBER (9,2);
BEGIN    -- Begin the executable section

/* Compute the annual salary based on the
   monthly salary input from the user */
annual_sal := monthly_sal * 12;
END;     -- This is the end of the block
/
```

SQL Functions in PL/SQL

- Available in procedural statements:
  - Single-row number
  - Single-row character
  - Data type conversion
  - Date
  - Timestamp
  - `GREATEST` and `LEAST`
  - Miscellaneous functions
- Not available in procedural statements:
  - `DECODE`
  - Group functions

SQL Functions in PL/SQL: Examples

- Get the length of a string:

```
desc_size INTEGER(5);
prod_description VARCHAR2(70):='You can use this
product with your radios for higher frequency';

-- get the length of the string in prod_description
desc_size:= LENGTH(prod_description);
```

- Convert the employee name to lowercase:

```
emp_name:= LOWER(emp_name);
```

Data Type Conversion

- Convert data to comparable data types
- Are of two types:
  - Implicit conversions
  - Explicit conversions
- Some conversion functions:
  - `TO_CHAR`
  - `TO_DATE`
  - `TO_NUMBER`
  - `TO_TIMESTAMP`

Data Type Conversion

1
```
date_of_joining DATE:= '02-Feb-2000';
```

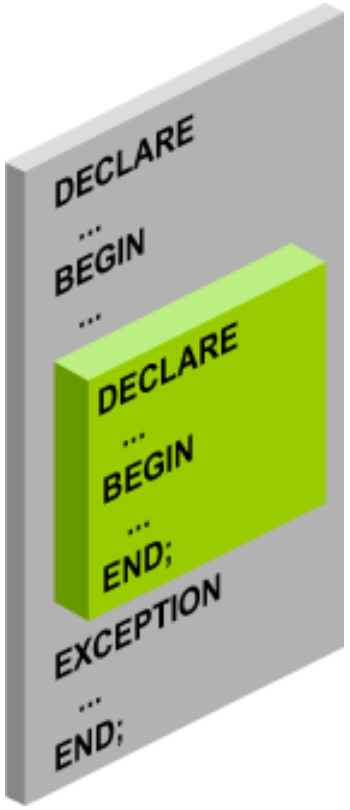
2
```
date_of_joining DATE:= 'February 02,2000';
```

3
```
date_of_joining DATE:= TO_DATE('February
02,2000','Month DD, YYYY');
```

Nested Blocks

PL/SQL blocks can be nested.

- **An executable section (`BEGIN` ... `END`) can contain nested blocks.**
- **An exception section can contain nested blocks.**

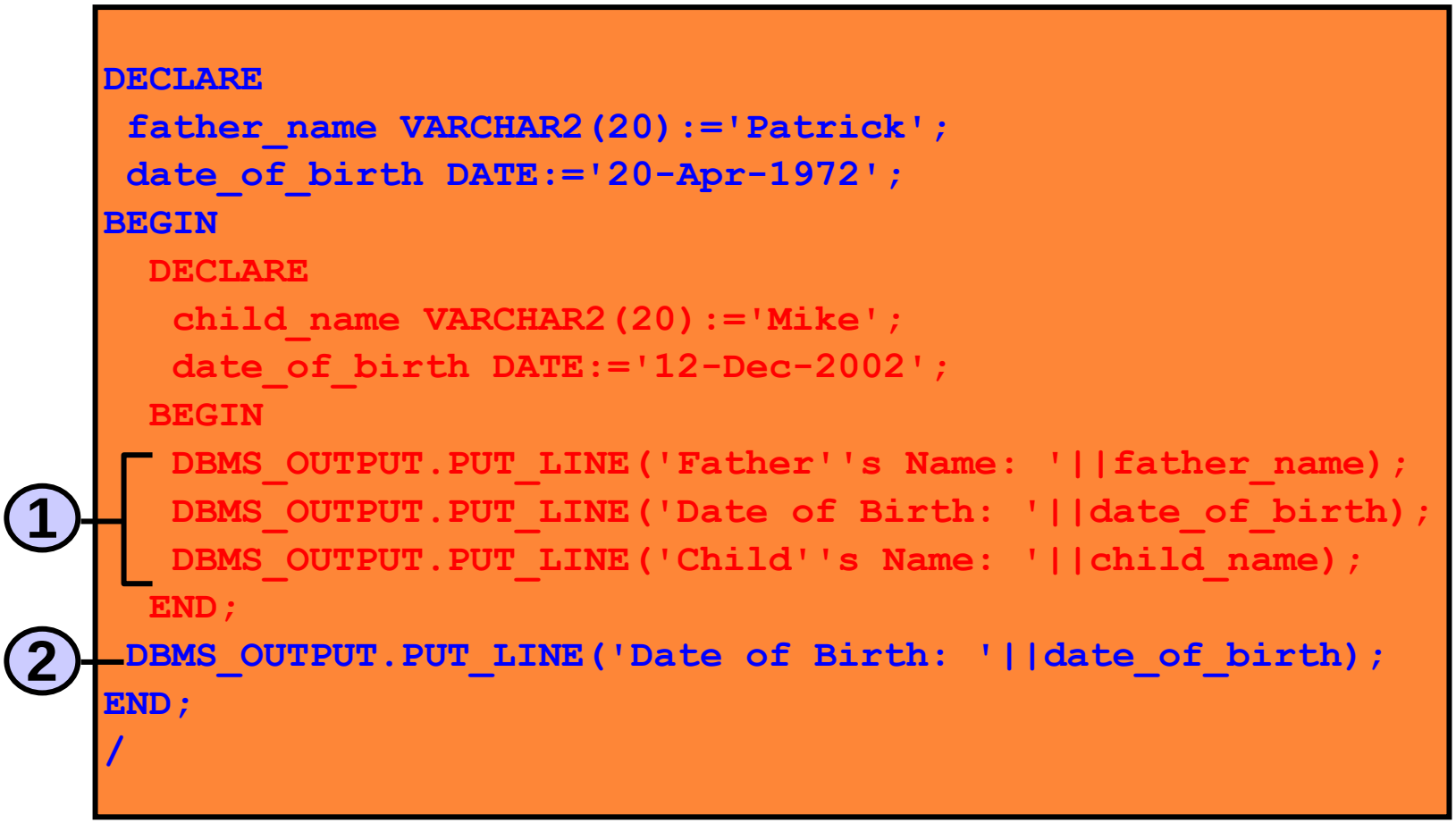


Nested Blocks

Example

```
DECLARE
 outer_variable VARCHAR2(20):='GLOBAL VARIABLE';
BEGIN
  DECLARE
   inner_variable VARCHAR2(20):='LOCAL VARIABLE';
  BEGIN
   DBMS_OUTPUT.PUT_LINE(inner_variable);
   DBMS_OUTPUT.PUT_LINE(outer_variable);
  END;
 DBMS_OUTPUT.PUT_LINE(outer_variable);
END;
/
```

Variable Scope and Visibility

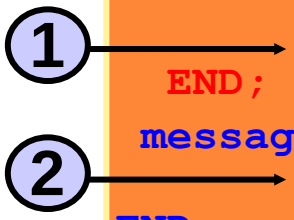


Qualify an Identifier

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

Determining Variable Scope

```
<<outer>>
DECLARE
  sal       NUMBER(7,2) := 60000;
  comm      NUMBER(7,2) := sal * 0.20;
  message   VARCHAR2(255) := ' eligible for commission';
BEGIN
  DECLARE
        sal          NUMBER(7,2) := 50000;
        comm         NUMBER(7,2) := 0;
        total_comp   NUMBER(7,2) := sal + comm;
  BEGIN
        message := 'CLERK not'||message;
        outer.comm := sal * 0.30;
  END;
 message := 'SALESMAN'||message;
END;
/
```



Operators in PL/SQL

- Logical
- Arithmetic
- Concatenation
- Parentheses to control order of operations

} **Same as in SQL**

- Exponential operator (**)

Operators in PL/SQL

- Examples
  - Increment the counter for a loop.

```
loop_count := loop_count + 1;
```

  - Set the value of a Boolean flag.

```
good_sal := sal BETWEEN 50000 AND 150000;
```

  - Validate whether an employee number contains a value.

```
valid  := (empno IS NOT NULL);
```

Programming Guidelines

- Make code maintenance easier by:
  - Documenting code with comments
  - Developing a case convention for the code
  - Developing naming conventions for identifiers and other objects
  - Enhancing readability by indenting

Indenting Code

- For clarity, indent each level of code.
- Example:

```
BEGIN
   IF x=0 THEN
      y:=1;
   END IF;
END;
/
```

```
DECLARE
   deptno        NUMBER(4);
   location_id   NUMBER(4);
BEGIN
   SELECT   department_id,
            location_id
   INTO     deptno,
            location_id
   FROM     departments
   WHERE    department_name
            = 'Sales';
...
END;
/
```

Summary

- In this lesson, you should have learned how to:
  - Use built-in SQL functions in PL/SQL
  - Write nested blocks to break logically related functionalities
  - Decide when to perform explicit conversions
  - Qualify variables in nested blocks

Practice 3: Overview

- This practice covers the following topics:
  - Reviewing scoping and nesting rules
  - Writing and testing PL/SQL blocks