2

Declaring PL/SQL Variables

Objectives

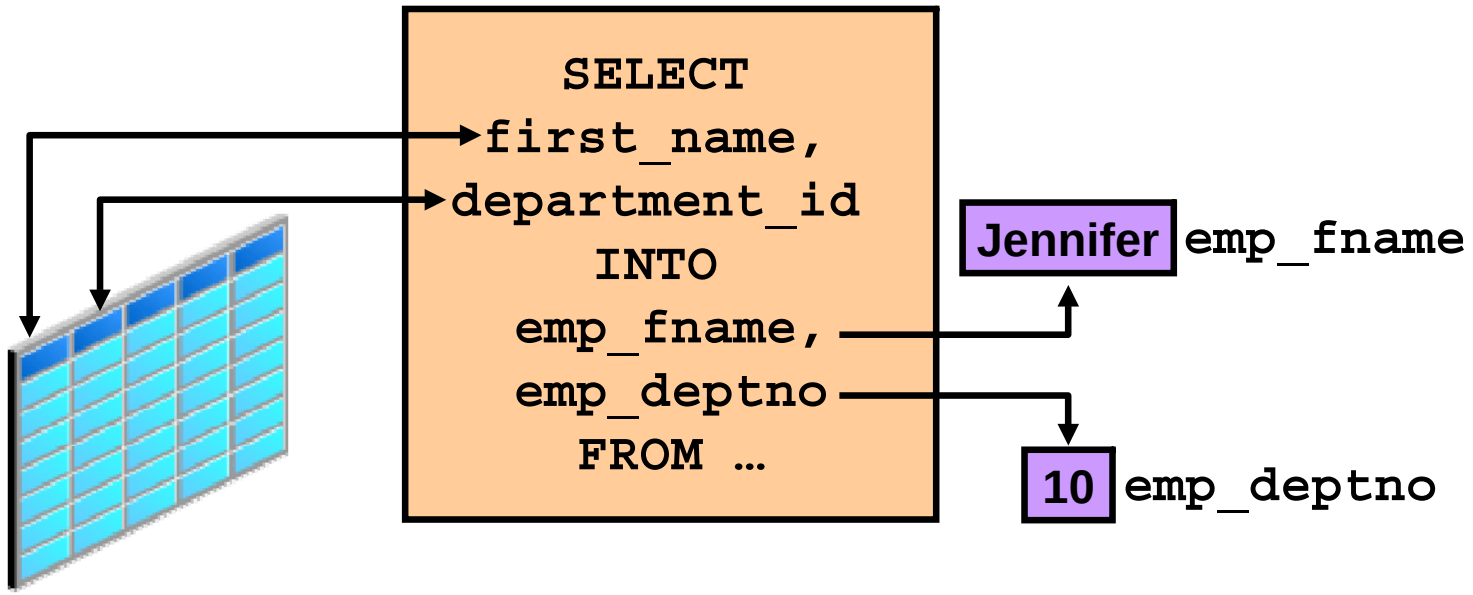
After completing this lesson, you should be able to do the following:

- Identify valid and invalid identifiers
- List the uses of variables
- Declare and initialize variables
- List and describe various data types
- Identify the benefits of using the `%TYPE` attribute
- Declare, use, and print bind variables

Use of Variables

Variables can be used for:

- Temporary storage of data
- Manipulation of stored values
- Reusability



Identifiers

Identifiers are used for:

- Naming a variable
- Providing conventions for variable names
  - Must start with a letter
  - Can include letters or numbers
  - Can include special characters (such as dollar sign, underscore, and pound sign)
  - Must limit the length to 30 characters
  - Must not be reserved words

Handling Variables in PL/SQL

Variables are:

- Declared and initialized in the declarative section
- Used and assigned new values in the executable section
- Passed as parameters to PL/SQL subprograms
- Used to hold the output of a PL/SQL subprogram

Declaring and Initializing PL/SQL Variables

Syntax

```
identifier [CONSTANT] datatype [NOT NULL]
     [:= | DEFAULT expr];
```

Examples

```
DECLARE
  emp_hiredate    DATE;
  emp_deptno      NUMBER(2) NOT NULL := 10;
  location        VARCHAR2(13) := 'Atlanta';
  c_comm          CONSTANT NUMBER := 1400;
```

Declaring and Initializing PL/SQL Variables

1

```
SET SERVEROUTPUT ON
DECLARE
  Myname VARCHAR2(20);
BEGIN
  DBMS_OUTPUT.PUT_LINE('My name is: '||Myname);
  Myname := 'John';
  DBMS_OUTPUT.PUT_LINE('My name is: '||Myname);
END;
/
```

2

```
SET SERVEROUTPUT ON
DECLARE
  Myname VARCHAR2(20):= 'John';
BEGIN
  Myname := 'Steven';
  DBMS_OUTPUT.PUT_LINE('My name is: '||Myname);
END;
/
```

Delimiters in String Literals

```
SET SERVEROUTPUT ON
DECLARE
   event VARCHAR2(15);
BEGIN
  event := q'!Father's day!';
  DBMS_OUTPUT.PUT_LINE('3rd Sunday in June is :
   '||event);
  event := q'[Mother's day]';
  DBMS_OUTPUT.PUT_LINE('2nd Sunday in May is :
   '||event);
END;
/
```

```
anonymous block completed
3rd Sunday in June is : Father's day
2nd Sunday in May is : Mother's day
```

Types of Variables

- PL/SQL variables:
  - Scalar
  - Composite
  - Reference
  - Large object (LOB)
- Non-PL/SQL variables: Bind variables

Types of Variables

TRUE

25-JAN-01

The soul of the lazy man desires, and he has nothing; but the soul of the diligent shall be made rich.

256120.08

Atlanta

Guidelines for Declaring and Initializing PL/SQL Variables

- Follow naming conventions.
- Use meaningful names for variables.
- Initialize variables designated as `NOT NULL` and `CONSTANT`.
- Initialize variables with the assignment operator (`:=`) or the `DEFAULT` keyword:

```
Myname VARCHAR2(20):='John';
```

```
Myname VARCHAR2(20) DEFAULT 'John';
```

- Declare one identifier per line for better readability and code maintenance.

Guidelines for Declaring and Initializing PL/SQL Variables

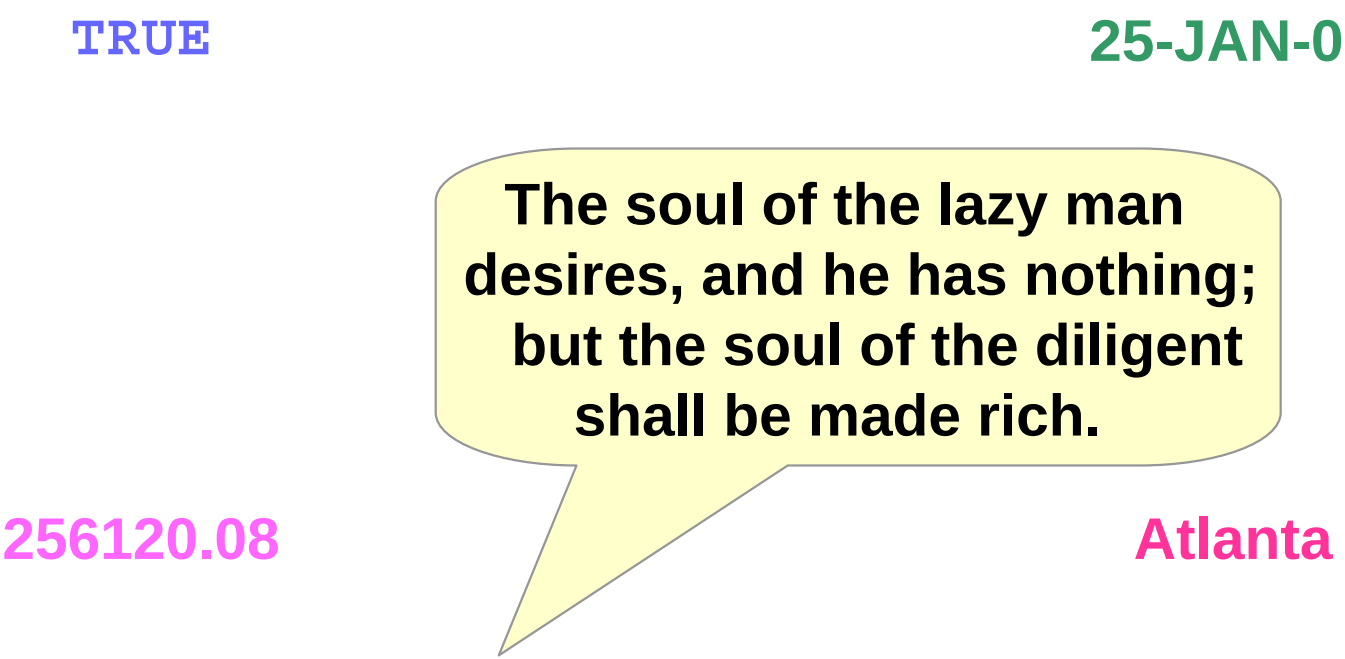
- Avoid using column names as identifiers.

```
DECLARE
  employee_id  NUMBER(6);
BEGIN
  SELECT    employee_id
  INTO      employee_id
  FROM      employees
  WHERE     last_name = 'Kochhar';
END;
/
```

- Use the `NOT NULL` constraint when the variable must hold a value.

Scalar Data Types

- Hold a single value
- Have no internal components



Base Scalar Data Types

- CHAR [(maximum_length)]
- VARCHAR2 (maximum_length)
- LONG
- LONG RAW
- NUMBER [(precision, scale)]
- BINARY_INTEGER
- PLS_INTEGER
- BOOLEAN
- BINARY_FLOAT
- BINARY_DOUBLE

Base Scalar Data Types

- DATE
- TIMESTAMP
- TIMESTAMP WITH TIME ZONE
- TIMESTAMP WITH LOCAL TIME ZONE
- INTERVAL YEAR TO MONTH
- INTERVAL DAY TO SECOND

BINARY_FLOAT and BINARY_DOUBLE

- Represent floating point numbers in IEEE 754 format
- Offer better interoperability and operational speed
- Store values beyond the values that the data type NUMBER can store
- Provide the benefits of closed arithmetic operations and transparent rounding

Declaring Scalar Variables

Examples

```
DECLARE
  emp_job          VARCHAR2(9);
  count_loop       BINARY_INTEGER := 0;
  dept_total_sal   NUMBER(9,2) := 0;
  orderdate        DATE := SYSDATE + 7;
  c_tax_rate       CONSTANT NUMBER(3,2) := 8.25;
  valid            BOOLEAN NOT NULL := TRUE;
  ...
```

%TYPE Attribute

The %TYPE attribute

- Is used to declare a variable according to:
  - A database column definition
  - Another declared variable
- Is prefixed with:
  - The database table and column
  - The name of the declared variable

Declaring Variables with the %TYPE Attribute

Syntax

```
identifier     table.column_name%TYPE;
```

Examples

```
...
  emp_lname       employees.last_name%TYPE;
  balance         NUMBER(7,2);
  min_balance     balance%TYPE := 1000;
...
```

Declaring Boolean Variables

- Only the values `TRUE`, `FALSE`, and `NULL` can be assigned to a Boolean variable.
- Conditional expressions use the logical operators `AND` and `OR` and the unary operator `NOT` to check the variable values.
- The variables always yield `TRUE`, `FALSE`, or `NULL`.
- Arithmetic, character, and date expressions can be used to return a Boolean value.

Bind Variables

Bind variables are:

- Created in the environment
- Also called *host* variables
- Created with the `VARIABLE` keyword
- Used in SQL statements and PL/SQL blocks
- Accessed even after the PL/SQL block is executed
- Referenced with a preceding colon

Printing Bind Variables

Example

```
VARIABLE emp_salary NUMBER
BEGIN
   SELECT salary  INTO :emp_salary
   FROM  employees WHERE employee_id = 178;
END;
/
PRINT emp_salary
SELECT first_name, last_name FROM employees
WHERE salary=:emp_salary;
```

Printing Bind Variables

Example

```
VARIABLE emp_salary NUMBER
SET AUTOPRINT ON
BEGIN
   SELECT salary  INTO :emp_salary
   FROM  employees WHERE employee_id = 178;
END;
/
```

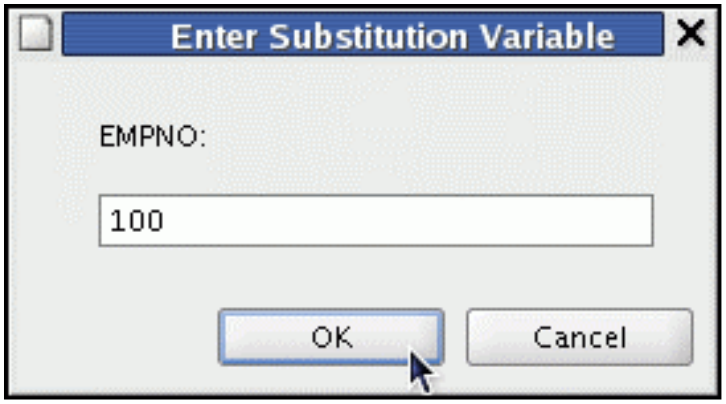
```
anonymous block completed
emp_salary
----
7000
```

Substitution Variables

- Are used to get user input at run time
- Are referenced within a PL/SQL block with a preceding ampersand
- Are used to avoid hard-coding values that can be obtained at run time

```
VARIABLE emp_salary NUMBER
SET AUTOPRINT ON
DECLARE
  empno NUMBER(6):=&empno;
BEGIN
  SELECT salary  INTO :emp_salary
  FROM  employees WHERE employee_id = empno;
END;
/
```

Substitution Variables



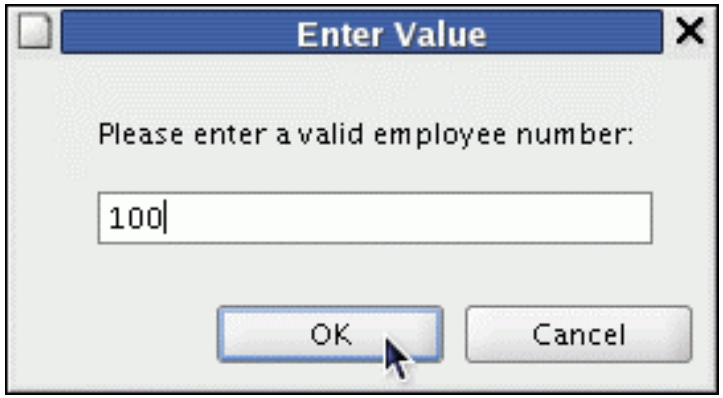
1

```
VARIABLE emp_salary NUMBER
SET AUTOPRINT ON
DECLARE
empno NUMBER(6):=100;
BEGIN
SELECT salary  INTO :emp_salary
FROM  employees WHERE employee_id = empno;
END;
anonymous block completed
emp_salary
-----
24000
```

2

Prompt for Substitution Variables

```
SET VERIFY OFF
VARIABLE emp_salary NUMBER
ACCEPT empno PROMPT 'Please enter a valid employee
number: '
SET AUTOPRINT ON
DECLARE
  empno NUMBER(6):= &empno;
BEGIN
  SELECT salary  INTO :emp_salary  FROM  employees
  WHERE employee_id = empno;
END;
/
```



Using DEFINE for a User Variable

Example

```
SET VERIFY OFF
DEFINE lname= Urman
DECLARE
  fname VARCHAR2(25);
BEGIN
  SELECT first_name INTO fname FROM employees
  WHERE last_name='&lname';
END;
/
```

Composite Data Types

TRUE	23-DEC-98	ATLANTA	

PL/SQL table structure

1	SMITH
2	JONES
3	NANCY
4	TIM

VARCHAR2

PLS_INTEGER

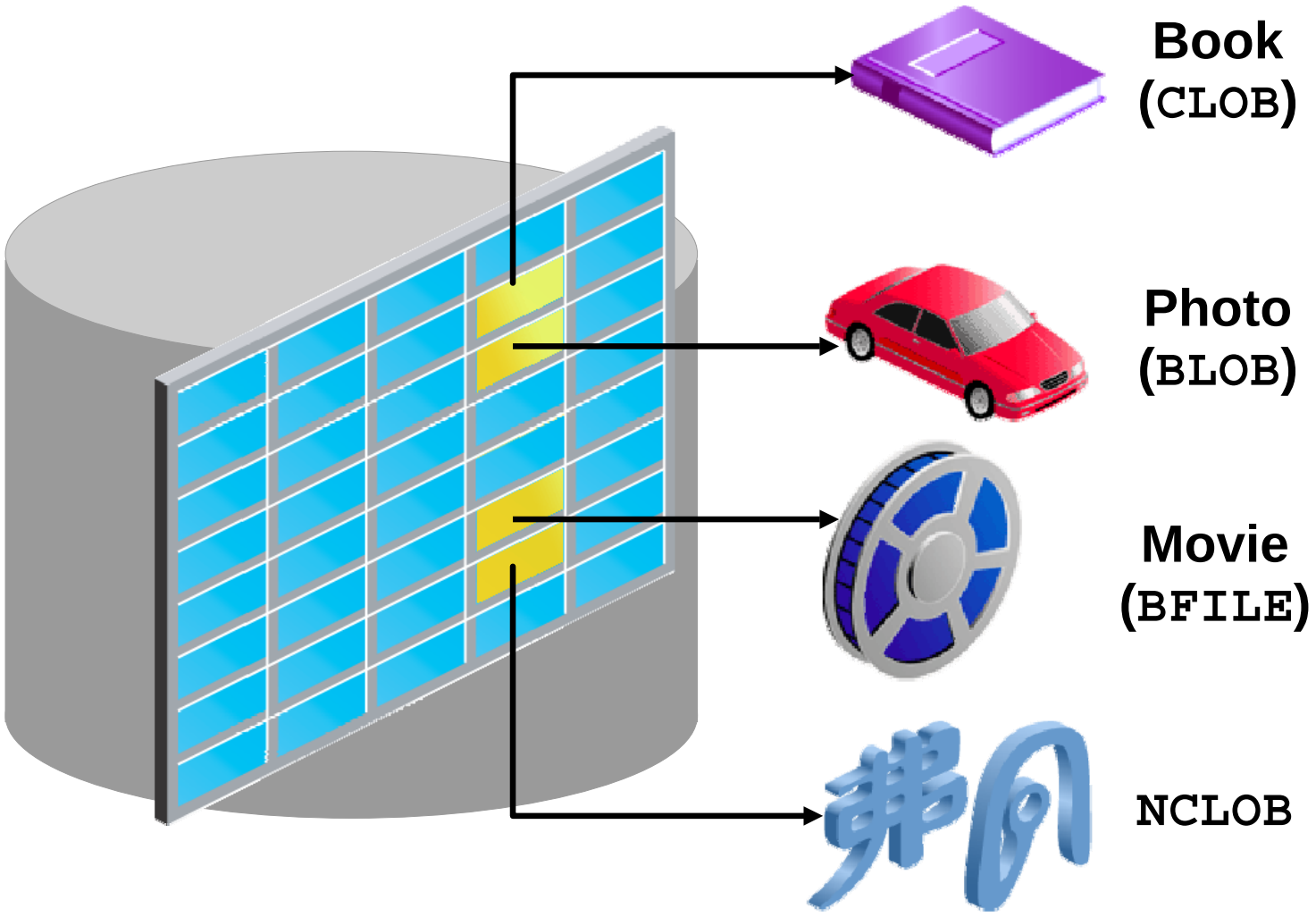
PL/SQL table structure

1	5000
2	2345
3	12
4	3456

NUMBER

PLS_INTEGER

LOB Data Type Variables



Summary

In this lesson, you should have learned how to:

- Recognize valid and invalid identifiers
- Declare variables in the declarative section of a PL/SQL block
- Initialize variables and use them in the executable section
- Differentiate between scalar and composite data types
- Use the `%TYPE` attribute
- Use bind variables

Practice 2: Overview

This practice covers the following topics:

- Determining valid identifiers
- Determining valid variable declarations
- Declaring variables within an anonymous block
- Using the `%TYPE` attribute to declare variables
- Declaring and printing a bind variable
- Executing a PL/SQL block