SECTION 2 LESSON 1 - Using Variables in PL/SQL

Slide 1: Using Variables in PL/SQL
No instructor notes for this slide

Slide 2: What Will I Learn?
No instructor notes for this slide

Slide 3: Why Learn It?
No instructor notes for this slide

Slide 4: Tell Me / Show Me – Use of Variables
Variables are mainly used for the storage of data and manipulation of stored values. Consider the SQL statement shown in the slide. The statement is retrieving the first_name and department_id from the table. If you have to manipulate the first_name or the department_id, then you have to store the retrieved value. Variables are used to temporarily store the value. You can use the value stored in these variables for processing or manipulating the data. Therefore, the variables are used for storing and manipulating data. Variables can store any PL/SQL object such as variables, types, cursors, and subprograms.

Reusability is another advantage of declaring variables. After they are declared, variables can be used repeatedly in an application by referring to them in the statements.

Slide 5: Tell Me / Show Me – Handling Variables in PL/SQL
No instructor notes for this slide

Slide 6: Tell Me / Show Me – Declaring and Initializing PL/SQL Variables
Point out that a variable is simply a name or label for a value stored in a piece of computer memory.

Slide 7: Tell Me / Show Me – Declaring and Initializing Variables: Syntax
In addition to variables, you can also declare cursors and exceptions in the declarative section. You will learn how to declare cursors and exceptions later in the course.

Slide 8: Tell Me / Show Me – Declaring and Initializing Variables: Syntax (continued)
No instructor notes for this slide
Ask students to think of examples of declarations that will cause an error:

```sql
v_location VARCHAR2(10) := 'Washington DC';     (why?)
v_grade    NUMBER(4)    := 'A';                 (why?)
```

A non-initialized variable contains a null value until a non-null value is explicitly assigned to it.

The information in this slide will be covered in more detail in later lessons; the most important point is that you understand that in PL/SQL, a variable can be passed to subprograms.

In the anonymous block, the variable `v_length_of_string` is assigned the value returned by the function `num_characters` when the value, Oracle Corporation is passed to it.

Variables – used for storage of data and manipulation of stored values

Parameters – values passed to a program by a user or by another program to customize the program
SECTION 2 LESSON 2 – Recognizing PL/SQL Lexical Units

Slide 1: Recognizing PL/SQL Lexical Units
No instructor notes for this slide

Slide 2: What Will I Learn?
No instructor notes for this slide

Slide 3: Why Learn It?
No instructor notes for this slide

Slide 4: Tell Me / Show Me – Lexical Units in a PL/SQL Block
No instructor notes for this slide

Slide 5: Tell Me / Show Me – Identifiers
No instructor notes for this slide

Slide 6: Tell Me / Show Me – Identifiers (continued)
Point out that reserved words are NOT identifiers. They have been highlighted here (in red) to contrast them with the identifiers.

Slide 7: Tell Me / Show Me – Properties of an Identifier
No instructor notes for this slide

Slide 8: Tell Me / Show Me – Valid and Invalid Identifiers
Be sure to name your objects carefully. Ideally, the identifier name should describe the object and its purpose. Avoid using identifier names such as A, X, Y1, temp, and so on because they make your code more difficult to read.

Slide 9: Tell Me / Show Me – Reserved Words
No instructor notes for this slide

Slide 10: Tell Me / Show Me – Reserved Words (continued)
No instructor notes for this slide

Slide 11: Tell Me / Show Me – Reserved Words (continued)
No instructor notes for this slide

Slide 12: Tell Me / Show Me – Delimiters
You have already learned that the symbol “;” is used to terminate a SQL or PL/SQL statement.

Slide 13: Tell Me / Show Me – Literals
No instructor notes for this slide

Slide 14: Tell Me / Show Me – Character Literals
Slide 15: Tell Me / Show Me – Numeric Literals
No instructor notes for this slide

Slide 16: Tell Me / Show Me – Boolean Literals
The idea of Boolean variables and literals may be new to students, because an Oracle database table cannot contain columns of datatype Boolean. Students will learn how to define Boolean variables later in this section.

Slide 17: Tell Me / Show Me – Comments
No instructor notes for this slide

Slide 18: Tell Me / Show Me – Syntax for Commenting Code
No instructor notes for this slide

Slide 19: Tell Me / Show Me – Terminology
Lexical Units – Building blocks of any PL/SQL block and are sequences of characters including letters, digits, tabs, returns, and symbols.
Identifiers – A name, up to 30 characters in length, given to a PL/SQL object.
Reserved words – Words that have special meaning to an Oracle database and cannot be used as identifiers.
Delimiters – Symbols that have special meaning to an Oracle database.
Literals – An explicit numeric, character string, date, or Boolean value that is not represented by an identifier.
Comments – Describe the purpose and use of each code segment and are ignored by PL/SQL.

Slide 20: Summary
No instructor notes for this slide

Slide 21: Try It / Solve It
No instructor notes for this slide
SECTION 2 LESSON 3 – Recognizing Data Types

Slide 1: Recognizing Data Types
No instructor notes for this slide

Slide 2: What Will I Learn?
No instructor notes for this slide

Slide 3: Why Learn It?
In programming, a data type is a classification of a particular type of information. People can differentiate between different types of data quite easily: by quickly looking at a number, we know whether it is a decimal, a time, a percentage, an amount of money, or a date. People use the format and symbols of the number (that is, %, :, and $) to recognize the type of the data. Similarly, PL/SQL uses special codes to keep track of the different types of data it processes.

Slide 4: Tell Me / Show Me – PL/SQL Data Types
Reference and Object data types are not covered in this course. For more information, refer to the PL/SQL User’s Guide and Reference manual.

Slide 5: Tell Me / Show Me – Scalar Data Types
Teachers should be aware that character and number data types have subtypes that associate a base type to a constraint. For example, INTEGER and POSITIVE are subtypes of the NUMBER base type: an INTEGER is a base type (NUMBER) constrained to allow only whole numbers (no decimal places). For more information and the complete list of scalar data types, refer to the PL/SQL User’s Guide and Reference.

Slide 6: Tell Me / Show Me – Scalar Data Types: Character (or String)
Students should recognize many of these scalar data types as being identical to table column data types. This is one of the benefits of using PL/SQL.

If anyone asks, a LONG variable can store up to 2 gigabytes (2,000,000,000) bytes.

Slide 7: Tell Me / Show Me – Scalar Data Types: Number
Point out that in PL/SQL (as in table columns) precision includes scale. For example, NUMBER(6,2) can contain a maximum value of 9999.99.

Do not go into detail about the * _INTEGER and BINARY_ * data types. Starting with Oracle version 10.1, PLS_INTEGER and BINARY_INTEGER require the same amount of storage and are equally fast. In Oracle version 9.2 and earlier, PLS_INTEGER required less storage and was faster than BINARY_INTEGER.

Slide 8: Tell Me / Show Me – Scalar Data Types: Date
No instructor notes for this slide
For now, it may be helpful to think of a *scalar* type as being like a single column value in a table, while a *record* data type is like a whole row of a table.

Students will learn about composite data types in Section 11.

LOB data types are covered in detail in Section 11.

- The character large object (CLOB) data type is used to store large blocks of character data in the database.
- The binary large object (BLOB) data type is used to store large unstructured or structured binary objects in the database. When you insert or retrieve such data to and from the database, the database does not interpret the data. External applications that use this data must interpret the data.
- The binary file (BFILE) data type is used to store large binary files. Unlike other LOBS, BFILES are not stored in the database. BFILES are stored outside the database. They could be operating-system files. Only a pointer to the BFILE is stored in the database.
- The national language character large object (NCLOB) data type is used to store large blocks of single-byte or fixed-width multibyte NCHAR Unicode data in the database.

**Scalar** – Hold a single value with no internal components.

**Composite** – Contain internal elements that are either scalar (record) or composite (record and table)

**LOB** – Hold values, called locators, that specify the location of large objects (such as graphic images) that are stored out of line.

**Reference** – Hold values, called pointers, that point to a storage location.

**Object** – A schema object with a name, attributes, and methods.

**CLOB** – Store large blocks of character data in the database.

**BLOB** – Store large unstructured or structured binary objects.

**BFILE** – Store large binary files outside of the database.

**NCLOB** (National Language Character Large Object) – Store large blocks of single-byte or fixed width multi-byte NCHAR Unicode data in the database.
Slide 16: Summary
No instructor notes for this slide

Slide 17: Try It / Solve It
No instructor notes for this slide
SECTION 2 LESSON 4 - Using Scalar Data Types

Slide 1: Using Scalar Data Types
No instructor notes for this slide

Slide 2: What Will I Learn?
No instructor notes for this slide

Slide 3: Why Learn It?
No instructor notes for this slide

Slide 4: Tell Me / Show Me – Declaring Character Variables
The examples of variable declaration shown in the slide are defined as follows:
- **emp_job**: Variable to store an employee job title
- **order_no**: Variable to store an order number. Note that a number can also be used.
- **product_id**: Variable to store a product ID
- **rpt_body_part**: Variable to store a part of a report. Note that a LOB can also be used to store large character-based objects.

Students may ask: how can a constant be variable – surely a constant cannot vary?
Remind them that the word variable in PL/SQL means: a name for a storage location which contains a value.

Slide 5: Tell Me / Show Me – Declaring Number Variables
The examples of variable declaration shown in the slide are defined as follows:
- **dept_total_sal**: Variable to accumulate the total salary for a department and initialized to 0
- **count_loop**: Variable to count the iterations of a loop and initialized to 0
- **c_tax_rate**: A constant variable for the tax rate, which never changes throughout the PL/SQL block and is set to 8.25

Slide 6: Tell Me / Show Me – Declaring Date Variables
The examples of variable declaration shown in the slide are defined as follows:
- **orderdate**: Variable to store the ship date of an order and initialize to one week from today
- **v_natl_holiday**: Variable to store the national holiday date for a country
- **web_sign_on_date**: Variable to store the time a user last logged in to a Web site
Slide 7: Tell Me / Show Me – Declaring Boolean Variables
The examples of variable declaration shown in the slide are defined as follows:

- **valid**: Flag to indicate whether a piece of data is valid or invalid, and initialized to TRUE
- **is_found**: Flag to indicate whether a piece of data has been found and initialized to FALSE
- **v_underage**: Flag to indicate whether a person is underage or not.

Slide 8: Tell Me / Show Me – Declaring Boolean Variables
With PL/SQL, you can compare variables in both SQL and procedural statements. These comparisons, called Boolean expressions, consist of simple or complex expressions separated by relational operators. In a SQL statement, you can use Boolean expressions to specify the rows in a table that are affected by the statement. In a procedural statement, Boolean expressions are the basis for conditional control. NULL stands for a missing, inapplicable, or unknown value.

**Examples**

```plsql
emp_sal1 := 50000;
emp_sal2 := 60000;
```

The following expression yields TRUE:

```plsql
emp_sal1 < emp_sal2
```

Declare and initialize a Boolean variable:

```plsql
DECLARE
  v_flag BOOLEAN := FALSE;
BEGIN
  v_flag := TRUE;
END;
```

Slide 9: Tell Me / Show Me – Guidelines for Declaring and Initializing PL/SQL Variables
Here are some guidelines to follow while declaring PL/SQL variables:

- Use meaningful and appropriate names for variables. For example, consider using salary and sal_with_commission instead of salary1 and salary2.
- Follow naming conventions—for example, v_name to represent a variable and c_name to represent a constant.
- Impose the NOT NULL constraint when the variable must contain a value. You cannot assign nulls to a variable defined as NOT NULL. The NOT NULL constraint must be followed by an initialization clause.
  ```plsql
  v_pincode NUMBER(15) NOT NULL := 17642;
  ```
- Avoid using column names as identifiers. If PL/SQL variables occur in SQL statements and have the same name as a column, the Oracle server assumes that it is the column that is being referenced. Although the example code in the slide works, code that is written using the same name for a database table and variable name is not easy to read or maintain.
Slide 10: Tell Me / Show Me – Anchoring Variables with the %TYPE Attribute
No instructor notes for this slide

Slide 11: Tell Me / Show Me – %TYPE Attribute
Ask students: what if the emp_salary column in the table is later altered to NUMBER(8,2) and a larger value such as 123456.78 is stored in it? What will the PL/SQL block do now?

Slide 12: Tell Me / Show Me – %TYPE Attribute (continued)
No instructor notes for this slide

Slide 13: Tell Me / Show Me – Declaring Variables with the %TYPE Attribute
Declare variables to store the last name of an employee. The variable v_emp_lname is defined to be of the same data type and size as the last_name column in the employees table. The %TYPE attribute provides the data type of a database column.

Declare variables to store the balance of a bank account, as well as the minimum balance, which is 1,000. The variable v_min_balance is defined to be of the same data type as the variable v_balance. The %TYPE attribute provides the data type of a variable.

A NOT NULL database column constraint does not apply to variables that are declared using %TYPE. Therefore, if you declare a variable using the %TYPE attribute that uses a database column defined as NOT NULL, you can assign the NULL value to the variable.

Slide 14: Tell Me / Show Me – Advantages of the %TYPE Attribute
No instructor notes for this slide

Slide 15: Tell Me / Show Me – %TYPE Attribute
If the column’s data type was altered later, the corresponding PL/SQL variable’s data type would automatically be changed to continue to match the column’s data type.

Slide 16: Tell Me / Show Me – Terminology
Boolean – A datatype that stores one of the three possible values used for logical calculations: TRUE, FALSE, or NULL.
%TYPE – Attribute used to declare a variable according to another previously declared variable or database column.

Slide 17: Summary
No instructor notes for this slide

Slide 18: Try It / Solve It
No instructor notes for this slide
SECTION 2 LESSON 5 - Review of SQL Joins

Slide 1: Review of SQL Joins
No instructor notes for this slide

Slide 2: What Will I Learn?
No instructor notes for this slide

Slide 3: Why Learn It?
No instructor notes for this slide

Slide 4: Tell Me / Show Me - Equijoin
No instructor notes for this slide

Slide 5: Tell Me / Show Me – Equijoin
No instructor notes for this slide

Slide 6: Tell Me / Show Me – Nonequijoin
No instructor notes for this slide

Slide 7: Tell Me / Show Me – Nonequijoin (continued)
No instructor notes for this slide

Slide 8: Tell Me / Show Me – Outer Join
No instructor notes for this slide

Slide 9: Tell Me / Show Me – Outer Join (continued)
No instructor notes for this slide

Slide 10: Tell Me / Show Me – Cartesian Product
No instructor notes for this slide

Slide 11: Tell Me / Show Me – Cartesian Product (continued)
No instructor notes for this slide

Slide 12: Tell Me / Show Me – Terminology
Equijoin – Sometimes called a simple join, it combines rows that have equal values for the specified columns.
Nonequijoin – Combines tables that have no exact matching columns.
Outer join – Combines rows that have equivalent values for the specified columns plus those rows in one of the tables that have no matching value in the other table.
Cartesian product – When a join query does not specify a condition in the WHERE clause. Often used with spatial/mapping applications.
Slide 13: Summary
No instructor notes for this slide

Slide 14: Try It / Solve It
No instructor notes for this slide
SECTION 2 LESSON 6 - Review of SQL Group Functions and Subqueries

Slide 1: Review of SQL Group Functions and Subqueries
No instructor notes for this slide

Slide 2: What Will I Learn?
No instructor notes for this slide

Slide 3: Why Learn It?
No instructor notes for this slide

Slide 4: Tell Me / Show Me – Group Functions
No instructor notes for this slide

Slide 5: Tell Me / Show Me – Group Functions (continued)
Point out that these two examples give the same result because COUNTRY_ID cannot be null.

Slide 6: Tell Me / Show Me – Group Functions (continued)
No instructor notes for this slide

Slide 7: Tell Me / Show Me – Group Functions (continued)
No instructor notes for this slide

Slide 8: Tell Me / Show Me – Group Functions (continued)
No instructor notes for this slide

Slide 9: Tell Me / Show Me – GROUP BY
- All individual columns specified along with the group function (AVG, SUM, COUNT, MAX, MIN, STDDEV, and VARIANCE) in the SELECT clause must be included in the GROUP BY clause.
- You cannot use a column alias in the GROUP BY clause.
- A GROUP BY clause can be used in a SQL statement without having a group function in the SELECT clause. For example:

  SELECT  region_id, country_name
  FROM    wf_countries
  GROUP BY region_id, country_name;

Slide 10: Tell Me / Show Me – HAVING
No instructor notes for this slide

Slide 11: Tell Me / Show Me – HAVING (continued)
No instructor notes for this slide
Slide 12: Tell Me / Show Me – Subqueries
Remind students that a subquery must be enclosed in parentheses (brackets).

Slide 13: Tell Me / Show Me – Subqueries (continued)
No instructor notes for this slide

Slide 14: Tell Me / Show Me – Group Functions and Subqueries
No instructor notes for this slide

Slide 15: Tell Me / Show Me – Group Functions
Ask students: what question is answered here?
Answer: Which country in Oceania has the highest population?

Slide 16: Tell Me / Show Me – Multiple-Row Subqueries
No instructor notes for this slide

Slide 17: Tell Me / Show Me – Multiple-Row Subqueries (continued)
Students may point out that we do not need a subquery for this. The following statement
will produce the same results:

```
SELECT country_name, population, airports
FROM   wf_countries
WHERE  airports > 1;
```

Slide 18: Tell Me / Show Me – ANY and ALL Operators
No instructor notes for this slide

Slide 19: Tell Me / Show Me – ANY Operator
Point out that we do not need a subquery for this. It could have been written (more
simply) as:

```
SELECT country_name, population, area
FROM   wf_countries
WHERE  area < 1000;
```

Slide 20: Tell Me / Show Me – ALL Operator
The slide example assumes that “A” is the first letter of the alphabet. It could have been
written better as:

```
SELECT country_name FROM wf_countries c, wf_world_regions w
WHERE c.region_id = w.region_id
 AND region_name NOT IN (SELECT region_name from wf_world_regions
 WHERE UPPER(region_name) LIKE 'A%');
```
Slide 21: Tell Me / Show Me – Terminology

**Group Functions** -- These functions operate on a whole table or on a specific grouping of rows to return one result.

**GROUP BY** – Clause used in a query to divide the rows in a table into smaller groups.

**HAVING** – Clause used in a query to restrict groups.

**Subquery** – A SELECT statement that is embedded in a clause of another SQL statement.

**Multiple Row subqueries** – Subqueries that use multiple row operators and return more than one row from the inner query.

**ANY** -- Operator used when the outer query WHERE clause is designed to restrict rows based on any value returned from the inner query.

**ALL** -- Operator used when the outer query WHERE clause is designed to restrict rows based on all values returned from the inner query.

Slide 22: Summary

No instructor notes for this slide

Slide 23: Try It / Solve It

No instructor notes for this slide
SECTION 2 LESSON 7 - Writing PL/SQL Executable Statements

Slide 1: Writing PL/SQL Executable Statements
No instructor notes for this slide

Slide 2: What Will I Learn?
No instructor notes for this slide

Slide 3: Why Learn It?
No instructor notes for this slide

Slide 4: Tell Me / Show Me – Assigning New Values to Variables
No instructor notes for this slide

Slide 5: Tell Me / Show Me – SQL Functions in PL/SQL
No instructor notes for this slide

Slide 6: Tell Me / Show Me – SQL Functions in PL/SQL (continued)
SQL functions help you to manipulate data; they fall into the following categories:
- Number
- Character
- Conversion
- Date
- Miscellaneous
The following functions are not available in procedural statements:
- DECODE
- Group functions: AVG, MIN, MAX, COUNT, SUM, STDDEV, and VARIANCE. Group functions apply to groups of rows in a table and, therefore, are available only in SQL statements in a PL/SQL block.
The functions mentioned here are only a subset of the complete list.

Slide 7: Tell Me / Show Me – Character Functions
No instructor notes for this slide

Slide 8: Tell Me / Show Me – Examples of Character Functions
No instructor notes for this slide

Slide 9: Tell Me / Show Me – Number Functions
No instructor notes for this slide

Slide 10: Tell Me / Show Me – Examples of Number Functions
No instructor notes for this slide
Slide 11: Tell Me / Show Me – Date Functions
No instructor notes for this slide

Slide 12: Tell Me / Show Me – Examples of Date Functions
No instructor notes for this slide

Slide 13: Tell Me / Show Me – Data Type Conversion
No instructor notes for this slide

Slide 14: Tell Me / Show Me – Implicit Conversions
In the chart, the cells marked ‘X’ show which implicit conversions can be done.

For this course, we will focus on implicit conversions between:
  • Characters and numbers
  • Characters and dates
For more information about the above chart, refer to “Converting PL/SQL Data Types” in the PL/SQL User’s Guide and Reference.

Slide 15: Tell Me / Show Me – Examples of Implicit Conversion
No instructor notes for this slide

Slide 16: Tell Me / Show Me – Drawbacks of Implicit Conversions
No instructor notes for this slide

Slide 17: Tell Me / Show Me – Drawbacks of Implicit Conversions (continued)
No instructor notes for this slide

Slide 18: Tell Me / Show Me – Explicit Conversions
No instructor notes for this slide

Slide 19: Tell Me / Show Me – Examples of Explicit Conversions
No instructor notes for this slide

Slide 20: Tell Me / Show Me – Examples of Explicit Conversions (continued)
Note that the DBMS_OUTPUT.PUT_LINE procedure expects an argument of type character. In the above example, variable v_c is a number, therefore we should explicitly code: DBMS_OUTPUT.PUT_LINE(TO_CHAR(v_c));
Slide 21: Tell Me / Show Me – Data Type Conversion Example
The examples in the slide show implicit and explicit conversions of the DATE data type.
1. Implicit conversion happens in this case and the date is assigned to v_date_of_joining.
2. PL/SQL gives you an error because the date that is being assigned is not in the default format.
3. Use the TO_DATE function to explicitly convert the given date in a particular format and assign it to the DATE data type variable date_of_joining.

Slide 22: Tell Me / Show Me – Operators in PL/SQL
No instructor notes for this slide

Slide 23: Tell Me / Show Me – Operators in PL/SQL (continued)
No instructor notes for this slide

Slide 24: Tell Me / Show Me – Operators in PL/SQL (continued)
The second example could have been written as:

```plsql
IF v_sal BETWEEN 50000 AND 150000 THEN
    v_good_sal := TRUE;
ELSE
    v_good_sal := FALSE;
END IF;
```

But assigning the result of the condition directly to the Boolean variable is much neater.

Slide 25: Tell Me / Show Me – Terminology
**Implicit conversion** – Converts data types dynamically if they are mixed in a statement.
**Explicit conversion** – Converts values from one data type to another by using built-in functions.

Slide 26: Summary
No instructor notes for this slide

Slide 27: Try It / Solve It
No instructor notes for this slide
SECTION 2 LESSON 8 - Nested Blocks and Variable Scope

Slide 1: Nested Blocks and Variable Scope
No instructor notes for this slide

Slide 2: What Will I Learn?
No instructor notes for this slide

Slide 3: Why Learn It?
No instructor notes for this slide

Slide 4: Tell Me / Show Me – Nested Blocks
No instructor notes for this slide

Slide 5: Tell Me / Show Me – Variable Scope
Answer: The scope of v_outer_variable includes both the outer and inner blocks. The scope of v_inner_variable includes only the inner block. It is valid to refer to v_outer_variable within the inner block, but referencing v_inner_variable within the outer block would return an error.

Slide 6: Tell Me / Show Me – Variable Scope
Answer:
The scope of v_father_name and v_date_of_birth is both blocks (inner and outer). The scope of v_child_name is the inner block only. See slide 9.

Slide 7: Tell Me / Show Me – Local and Global Variables
Therefore the scope of a variable consists of all the blocks in which the variable is either local (the declaring block) or global (nested blocks within the declaring block).

Slide 8: Tell Me / Show Me – Local and Global Variables (continued)
No instructor notes for this slide

Slide 9: Tell Me / Show Me – Variable Scope
No instructor notes for this slide

Slide 10: Tell Me / Show Me – Variable Naming
No instructor notes for this slide

Slide 11: Tell Me / Show Me – Variable Visibility
The statement will reference the v_date_of_birth declared in the inner block.
Slide 12: Tell Me / Show Me – Variable Visibility (continued)
   1. Observe the code in the executable section of the PL/SQL block. You can print the father’s name, the child’s name, and the date of birth. Only the child’s date of birth can be printed here because the father’s date of birth is not visible here.
   2. The father’s date of birth is visible here and therefore can be printed.

Slide 13: Tell Me / Show Me – Variable Visibility (continued)
No instructor notes for this slide

Slide 14: Tell Me / Show Me – Qualifying an Identifier
No instructor notes for this slide

Slide 15: Tell Me / Show Me – Qualifying an Identifier (continued)
We could also label the inner block but this is not needed here.

Slide 16: Tell Me / Show Me – Scope of Exceptions in Nested Blocks
This lesson briefly introduces the ideas of exception handling and propagation to the calling environment. Tell students they will learn much more about exception handling in Section 6.

Slide 17: Tell Me / Show Me – Trapping Exceptions with a Handler
No instructor notes for this slide

Slide 18: Tell Me / Show Me – Handling Exceptions in an Inner Block
No instructor notes for this slide

Slide 19: Tell Me / Show Me – Propagating Exceptions to an Outer Block
No instructor notes for this slide

Slide 20: Tell Me / Show Me – Propagating Exceptions to an Outer Block (continued)
No instructor notes for this slide

Slide 21: Tell Me / Show Me – Propagating Exceptions to a Subblock
No instructor notes for this slide
Slide 22: Tell Me / Show Me – Terminology

**Variable scope** – Consists of all the blocks in which the variable is either local (the declaring block) or global (nested blocks within the declaring block).

**Variable visibility** – The portion of the program where the variable can be accessed without using a qualifier.

**Qualifier** – A label given to a block.

**Exception handling** – Allows clean separation of the error processing code from the executable code so that a program can continue operating in the presence of errors.

**Exception propagating** – The exception reproduces itself in successive enclosing blocks until a handler is found or there are no more blocks to search in.

---

Slide 23: Summary

No instructor notes for this slide

---

Slide 24: Try It / Solve It

No instructor notes for this slide
SECTION 2 LESSON 9 - Good Programming Practices

Slide 1: Good Programming Practices
No instructor notes for this slide

Slide 2: What Will I Learn?
No instructor notes for this slide

Slide 3: Why Learn It?
There are several books and Web sites written about PL/SQL best practices. Ask your students to search the Internet to find these books.

Slide 4: Tell Me / Show Me – Programming Practices
No instructor notes for this slide.

Slide 5: Tell Me / Show Me – Programming Guidelines
Follow programming guidelines shown in the slide to produce clear code and reduce maintenance when developing a PL/SQL block.

Slide 6: Tell Me / Show Me – Commenting Code
Comment code to document each phase and to assist debugging. Comment the PL/SQL code with two dashes (--) if the comment is on a single line, or enclose the comment between the symbols “/*” and “*/” if the comment spans several lines. Comments are strictly informational and do not enforce any conditions or behavior on logic or data. Well-placed comments are extremely valuable for code readability and future code maintenance. In the example in the slide, the lines enclosed within “/*” and “*/” is a comment that explains the code that follows it.

Slide 7: Tell Me / Show Me – Case Conventions
In a sense, it doesn’t matter which convention we use as long as (a) a meaningful convention exists, and (b) we use it consistently. The case convention described here is the one most commonly used in SQL and PL/SQL, and is also the one used in the Oracle product documentation.

Slide 8: Tell Me / Show Me – Naming Conventions
The course examples follow the convention described in this slide.
Slide 9: Tell Me / Show Me – Indenting Code
For clarity, and to enhance readability, indent each level of code. To show structure, you can divide lines by using carriage returns and indent lines by using spaces or tabs. Compare the following IF statements for readability:

```sql
IF x>y THEN v_max:=x;ELSE
  v_max:=y;END IF;

IF x > y THEN
  v_max := x;
ELSE
  v_max := y;
END IF;
```

Slide 10: Summary
No instructor notes for this slide

Slide 11: Try It / Solve It
No instructor notes for this slide
PRACTICE SOLUTIONS

SECTION 2 LESSON 1 - Using Variables in PL/SQL

Terminology
1. __Variables______________ Used for storage of data and manipulation of stored values.
2. __Parameters______________ values passed to a program by a user or by another program to customize the program.

Try It/Solve It
1. Fill in the blanks.
   A. Variables can be assigned to the output of a ___ sub program ___.
   B. Variables can be assigned values in the __ executable (or declarative) _ section of a PL/SQL block.
   C. Variables can be passed as ___ parameters ____ to subprograms.

2. Identify valid and invalid variable declaration and initialization:
   
   number_of_copies   PLS_INTEGER;
   Valid
   printer_name   CONSTANT VARCHAR2(10);
   Invalid
   deliver_to   VARCHAR2(10):=Johnson;
   Invalid
   by_when     DATE:= SYSDATE+1;
   Valid

3. Examine the following anonymous block and choose the appropriate statement.

   DECLARE
   fname VARCHAR2(20);
   lname VARCHAR2(15) DEFAULT 'fernandez';
   BEGIN
   DBMS_OUTPUT.PUT_LINE( FNAME ||' ' ||lname);
   END;

   A. The block will execute successfully and print ' fernandez'.
   B. The block will give an error because the fname variable is used without initializing.
   C. The block will execute successfully and print 'null fernandez'.
   D. The block will give an error because you cannot use the DEFAULT keyword to initialize a variable of the VARCHAR2 type.
   E. The block will give an error because the FNAME variable is not declared.
4. In Application Express:

A. Create the following function:

```sql
CREATE FUNCTION num_characters (p_string IN VARCHAR2) RETURN INTEGER AS
   v_num_characters INTEGER;
BEGIN
   SELECT length(p_string) into v_num_characters
   FROM dual;
   RETURN v_num_characters;
END;
```

B. Create and execute the following anonymous block:

```sql
DECLARE
   v_length_of_string INTEGER;
BEGIN
   v_length_of_string := num_characters('Oracle Corporation');
   DBMS_OUTPUT.PUT_LINE(v_length_of_string);
END;
```

5. Write an anonymous block that uses a country name as input and prints the highest and lowest elevations for that country. Use the `wf_countries` table. Execute your block three times using United States of America, French Republic, and Japan.

```sql
DECLARE
   v_country_name      varchar2(50):= 'United States of America';
   v_lowest_elevation  number(6);
   v_highest_elevation number(6);
BEGIN
   SELECT lowest_elevation, highest_elevation
   INTO v_lowest_elevation, v_highest_elevation
   FROM  wf_countries
   WHERE country_name = v_country_name;
   DBMS_OUTPUT.PUT_LINE('The lowest elevation for '||v_country_name||' is: '||v_lowest_elevation);
   DBMS_OUTPUT.PUT_LINE('The highest elevation for '||v_country_name||' is: '||v_highest_elevation);
END;
```
SECTION 2 LESSON 2 - Recognizing PL/SQL Lexical Units

Terminology
1. **Literals** ____________ An explicit numeric, character string, date, or Boolean value that is not represented by an identifier.
2. **Delimiters** _____________ Symbols that have special meaning to an Oracle database.
3. **Reserved words** ____________ Words that have special meaning to an Oracle database and cannot be used as identifiers.
4. **Comments** _____________ Describe the purpose and use of each code segment and are ignored by PL/SQL.
5. **Lexical Units** ____________ Building blocks of any PL/SQL block and are sequences of characters including letters, digits, tabs, returns, and symbols.
6. **Identifiers** _____________ A name, up to 30 characters in length, given to a PL/SQL object.

Try It/Solve It Questions
1. Fill in the blanks.
   A. An __**identifier**_____ is the name given to a PL/SQL object.
   B. A __**reserved word**_____ is a word that has special meaning to the Oracle database.
   C. A _____**delimiter**___ is a symbol that has special meaning to the Oracle database.
   D. A _____**literal**________ is an explicit numeric, character string, date, or Boolean value that is not represented by an identifier.
   E. A_____**comment**_______ explains what a piece of code is trying to achieve.
2. Identify each of the following identifiers as valid or invalid. If invalid, specify why.

<table>
<thead>
<tr>
<th>Identifier</th>
<th>Valid (X)</th>
<th>Invalid (X)</th>
<th>Why Invalid?</th>
</tr>
</thead>
<tbody>
<tr>
<td>Today</td>
<td></td>
<td>X</td>
<td>Contains a space</td>
</tr>
<tr>
<td>Last name</td>
<td></td>
<td>X</td>
<td>Contains a quote delimiter</td>
</tr>
<tr>
<td>today’s_date</td>
<td></td>
<td>X</td>
<td>Contains more than 30 characters</td>
</tr>
<tr>
<td>number_of_days_in_february_this_year</td>
<td></td>
<td>X</td>
<td></td>
</tr>
<tr>
<td>Isleap$year</td>
<td></td>
<td>X</td>
<td></td>
</tr>
<tr>
<td>#number</td>
<td></td>
<td>X</td>
<td>Must start with a letter</td>
</tr>
<tr>
<td>NUMBER#</td>
<td></td>
<td>X</td>
<td></td>
</tr>
<tr>
<td>Number1to7</td>
<td></td>
<td>X</td>
<td></td>
</tr>
</tbody>
</table>

3. Identify the reserved words in the following list.

<table>
<thead>
<tr>
<th>Word</th>
<th>Reserved? Y/N</th>
</tr>
</thead>
<tbody>
<tr>
<td>create</td>
<td>Y</td>
</tr>
<tr>
<td>make</td>
<td>N</td>
</tr>
<tr>
<td>table</td>
<td>Y</td>
</tr>
<tr>
<td>seat</td>
<td>N</td>
</tr>
<tr>
<td>alter</td>
<td>Y</td>
</tr>
<tr>
<td>rename</td>
<td>Y</td>
</tr>
<tr>
<td>row</td>
<td>Y</td>
</tr>
<tr>
<td>number</td>
<td>Y</td>
</tr>
<tr>
<td>web</td>
<td>N</td>
</tr>
</tbody>
</table>

4. What kind of lexical unit (for example Reserved word, Delimiter, Literal, Comment) is each of the following?

<table>
<thead>
<tr>
<th>Value</th>
<th>Lexical Unit</th>
</tr>
</thead>
<tbody>
<tr>
<td>SELECT</td>
<td>Reserved word</td>
</tr>
<tr>
<td>:=</td>
<td>Delimiter</td>
</tr>
<tr>
<td>‘TEST’</td>
<td>Literal</td>
</tr>
<tr>
<td>FALSE</td>
<td>Literal</td>
</tr>
<tr>
<td>-- new process</td>
<td>Comment</td>
</tr>
<tr>
<td>FROM</td>
<td>Reserved word</td>
</tr>
<tr>
<td>/*select the country with the highest elevation */</td>
<td>Comment</td>
</tr>
<tr>
<td>V_test</td>
<td>Identifier</td>
</tr>
<tr>
<td>4.09</td>
<td>Literal</td>
</tr>
</tbody>
</table>
SECTION 2 LESSON 3 - Recognizing Data Types

Terminology
1. **NCLOB** Store large blocks of single-byte or fixed width multi-byte NCHAR data in the database.
2. **LOB** Hold values, called locators, that specify the location of large objects (such as graphic images) that are stored out of line.
3. **Scalar** Hold a single value with no internal components.
4. **BLOB** Store large unstructured or structured binary objects.
5. **Composite** Contain internal elements that are either scalar (record) or composite (record and table)
6. **BFILE** Store large binary files outside of the database.
7. **Reference** Hold values, called pointers, that point to a storage location.
8. **Object** A schema object with a name, attributes, and methods.
9. **CLOB** Store large blocks of character data in the database.

Try It/Solve It
1. In your own words, describe what a data type is and explain why it is important.

*PL/SQL uses special data types to keep track of the different types of data it processes. These data types define how the data is physically stored, what the constraints for the data are, and finally, what the valid range of values for the data is.*
2. Match the data type category (LOB, Scalar, Composite, Reference, and Object) with the appropriate definition. Each data type may be used more than once.

<table>
<thead>
<tr>
<th>Description</th>
<th>Data Type</th>
</tr>
</thead>
<tbody>
<tr>
<td>Stores a large amount of data</td>
<td>LOB</td>
</tr>
<tr>
<td>Has internal components that can be manipulated individually</td>
<td>Composite</td>
</tr>
<tr>
<td>Has a name, attributes, and methods</td>
<td>Object</td>
</tr>
<tr>
<td>Includes CLOBs, BLOBs, BFILEs, and NCLOBs</td>
<td>LOB</td>
</tr>
<tr>
<td>Has no internal components</td>
<td>Scalar</td>
</tr>
<tr>
<td>Includes TABLEs, RECORDs, NESTED TABLEs, and VARRAYs</td>
<td>Composite</td>
</tr>
<tr>
<td>Includes TIMESTAMP, DATE, BINARY_INTEGER, LONG, LONG RAW, and BOOLEAN</td>
<td>Scalar</td>
</tr>
<tr>
<td>Holds values, called pointers, that point to a storage location</td>
<td>Reference</td>
</tr>
</tbody>
</table>

3. Enter the data type category for each value into the Data Type Category column. In the Data Type column, enter a specific data type that can be used for the value. The first one has been done for you.

<table>
<thead>
<tr>
<th>Value</th>
<th>Data Type Category</th>
<th>Data Type</th>
</tr>
</thead>
<tbody>
<tr>
<td>Switzerland</td>
<td>Scalar</td>
<td>Character</td>
</tr>
<tr>
<td>100.20</td>
<td>Scalar</td>
<td>Number</td>
</tr>
<tr>
<td>1053</td>
<td>Scalar</td>
<td>Number (or pls_integer)</td>
</tr>
<tr>
<td>12-DEC-2005</td>
<td>Scalar</td>
<td>Date</td>
</tr>
<tr>
<td>False</td>
<td>Scalar</td>
<td>Boolean</td>
</tr>
<tr>
<td>Index 1</td>
<td>'Newman'</td>
<td>Composite</td>
</tr>
<tr>
<td>Index 2</td>
<td>'Raman'</td>
<td>Table</td>
</tr>
<tr>
<td>Index 3</td>
<td>'Han'</td>
<td></td>
</tr>
<tr>
<td>A movie</td>
<td>LOB</td>
<td>BFILE</td>
</tr>
<tr>
<td>A soundbyte</td>
<td>LOB</td>
<td>BFILE or BLOB</td>
</tr>
<tr>
<td>A picture</td>
<td>LOB</td>
<td>BLOB</td>
</tr>
</tbody>
</table>
SECTION 2 LESSON 4 - Using Scalar Data Types

Terminology
1. ___ Boolean _____________ A datatype that stores one of the three possible values used for logical calculations: TRUE, FALSE, or NULL.
2. ___ %TYPE _______________ Attribute used to declare a variable according to another previously declared variable or database column.

Try It/Solve It
1. Declarations:
   A. Which of the following variable declarations are valid?

<table>
<thead>
<tr>
<th>Declaration</th>
<th>Valid or Invalid</th>
</tr>
</thead>
<tbody>
<tr>
<td>a number_of_students PLS_INTEGER;</td>
<td>Valid</td>
</tr>
<tr>
<td>b STUDENT_NAME VARCHAR2(10)=Johnson;</td>
<td>Invalid</td>
</tr>
<tr>
<td>c stu_per_class CONSTANT NUMBER;</td>
<td>Invalid</td>
</tr>
<tr>
<td>d today DATE := SYSDATE+1;</td>
<td>Valid</td>
</tr>
</tbody>
</table>

   B. For those declarations in 1.A. that are invalid, describe why they are invalid.

   b is invalid because string literals should be enclosed within single quotation marks and because := is used to assign values.
   c is invalid because constant variables must be initialized during declaration.

   C. Write an anonymous block in which you declare and print each of the variables in 1A, correcting the invalid declarations.

   DECLARE
   number_of_students PLS_INTEGER := 30;
   student_name VARCHAR2(10) := 'Johnson';
   stu_per_class CONSTANT NUMBER := 1;
   today DATE := SYSDATE + 1;
   BEGIN
   DBMS_OUTPUT.PUT_LINE ('The number of students is: ''||number_of_students||''.');
   DBMS_OUTPUT.PUT_LINE ('The name of the students is: ''||student_name||''.');
   DBMS_OUTPUT.PUT_LINE ('The number of students per class is: ''||stu_per_class||''.');
   DBMS_OUTPUT.PUT_LINE ('Todays date is: ''||today||''.');
   END;
2. Evaluate the variables in the following code. Answer the following questions about each variable. Is it named well? Why or why not? If it is not named well, what would be a better name and why?

```sql
DECLARE
    country_name  VARCHAR2 (50);
    median_age      NUMBER(6,2);
BEGIN
    SELECT country_name, median_age INTO country_name, median_age
    FROM wf_countries
    WHERE country_name = 'United States of America');
    DBMS_OUTPUT.PUT_LINE(' The median age in '||country_name||' is '||median_age||'.');
END;
```

Both variables have the same name as database table columns. There are many possible better names, for example `v_country_name` and `v_median_age`.

3. Examine the declarations in question 2. Change the declarations so that they use the `%TYPE` attribute.

```sql
country_name  wf_countries.country_name%TYPE;
median_age      wf_countries.median_age%TYPE;
```

4. In your own words, describe why using the `%TYPE` attribute is preferable to hard-coding data types. Can you explain how you could run into problems in the future by hard-coding the data types of the `country_name` and `median_age` variables in question 2?

It is preferable to use the `%TYPE` attribute rather than hard-coding data types because it is possible that the table definition for the underlying data will change. For example, a country may change its name to a value longer than 50 characters. If this were to happen, the `COUNTRY_NAME` column definition in the `WF_COUNTRIES` table would need to be altered.
5. Create the following anonymous block:

```
BEGIN
    DBMS_OUTPUT.PUT_LINE('Hello World');
END;
```

A. Add a declarative section to this PL/SQL block. In the declarative section, declare the following variables:

- The today variable of the DATE type. Initialize today with sysdate.
- The tomorrow variable of the today type. Use the %TYPE attribute to declare this variable.

```
DECLARE
    today          DATE:=SYSDATE;
    tomorrow   today%TYPE;
BEGIN
    DBMS_OUTPUT.PUT_LINE('Hello World');
END;
```

B. In the executable section, initialize the tomorrow variable with an expression that calculates tomorrow’s date (add 1 to the value in today). Print the value of today and tomorrow after printing ‘Hello World’.

```
DECLARE
    today          DATE:=SYSDATE;
    tomorrow   today%TYPE;
BEGIN
    tomorrow := today + 1;
    DBMS_OUTPUT.PUT_LINE('Hello World');
    DBMS_OUTPUT.PUT_LINE(today);
    DBMS_OUTPUT.PUT_LINE(tomorrow);
END;
```
SECTION 2 LESSON 5 - Review of SQL Joins

Terminology
1. **Outer join** ________________ Combines rows that have equivalent values for the specified columns plus those rows in one of the tables that have no matching value in the other table.
2. **Cartesian product** __________ When a join query does not specify a condition in the WHERE clause.
3. **Equijoin** ________________ Sometimes called a simple join, it combines rows that have equal values for the specified columns.
4. **Nonequijoin** ________________ Combines tables that have no exact matching columns.

Try It/Solve It
1. Write and test an equijoin statement that lists each country’s name, currency code, and currency name. Order the list by country name.

```
SELECT c.country_name, cu.currency_code, cu.currency_name
    FROM wf_countries c, wf_currencies cu
    WHERE c.currency_code = cu.currency_code
    ORDER BY c.country_name;
```

2. Write and test an equijoin statement that lists each language and the country or countries where it is official.

```
SELECT l.language_name, c.country_name
    FROM wf_countries c, wf_spoken_languages sl, wf_languages l
    WHERE c.country_id = sl.country_id
    AND     sl.language_id = l.language_id
    AND     sl.official = ‘Y’;
```

3. List the name of each country, its population, and the name of its region. Order the list by region name.

```
SELECT c.country_name, c.population, r.region_name
    FROM wf_world_regions r, wf_countries c
    WHERE r.region_id = c.region_id
    ORDER by r.region_name;
```
4. Display a list of currencies whose name begins with “R” and the country or countries in which they are used. Include currencies which are not used in any country.

```
SELECT c.country_name, cu.currency_name
FROM wf_countries c, wf_currencies cu
WHERE c.currency_code(+) = cu.currency_code
AND cu.currency_name like 'R%'
```

(Outer join output should include the Renminbi, which is not used in any country.)

5. There are 100 rows in table A and 250 rows in table B. The Cartesian product of A and B would yield this number of rows:

A. 250  
B. 25000  
C. 100  
D. none of the above

6. Which statement is definitely wrong?

A. `SELECT e.employee_id, d.dept_id
   FROM employees e, department d
   WHERE e.dept_id = d.dept_id(+);`

B. `SELECT e.employee_id, d.dept_id
   FROM employees e, department d
   WHERE e.dept_id (+)= d.dept_id(+);`

C. `SELECT e.employee_id, d.dept_id
   FROM employees e, department d
   WHERE e.dept_id (+)= d.dept_id;`

D. none of the above
7. Which statement’s results will include departments with no employees?

A. SELECT e.employee_id, d.department_id
   FROM employees e, departments d
   WHERE e.department_id = d.department_id(+);

B. SELECT e.employee_id, d.department_id
   FROM employees e, departments d
   WHERE e.department_id (+)= d.department_id;

C. both A and B

D. none of the above

8. Which statement’s results will include employees with no department?

A. SELECT e.employee_id, d.department_id
   FROM employees e, departments d
   WHERE e.department_id = d.department_id(+);

B. SELECT e.employee_id, d.department_id
   FROM employees e, departments d
   WHERE e.department_id (+)= d.department_id;

C. both A and B

D. none of the above
9. Given the tables BEVERAGES and TEMPERATURE_RANGES, write a SQL statement that will display the beverage, temperature, and range as defined by the low and high values.

<table>
<thead>
<tr>
<th>BEVERAGES</th>
</tr>
</thead>
<tbody>
<tr>
<td>Beverage</td>
</tr>
<tr>
<td>Coffee</td>
</tr>
<tr>
<td>Wine</td>
</tr>
<tr>
<td>Soda</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>TEMPERATURE_RANGES</th>
</tr>
</thead>
<tbody>
<tr>
<td>Range</td>
</tr>
<tr>
<td>Low Value</td>
</tr>
<tr>
<td>Hot</td>
</tr>
<tr>
<td>Room</td>
</tr>
<tr>
<td>Cool</td>
</tr>
<tr>
<td>Very cool</td>
</tr>
</tbody>
</table>

SELECT b.beverage, b.temp, t.range
FROM beverages b, temperature_ranges t
WHERE b.temp BETWEEN t.low_value AND t.high_value;
Extension Exercise

1. Write SQL scripts to create the beverage and temperature range tables from question 9. Create the tables in Application Express. Create anonymous PL/SQL blocks to populate the tables with the data illustrated in exercise 10. Execute the blocks in Application Express. Using the two tables, write one of each of the types of joins you learned in this lesson (equijoin, nonequijoin, outer join, and Cartesian product).

--execute each statement separately

CREATE TABLE beverages (beverage VARCHAR2(15) NOT NULL UNIQUE, temperature NUMBER);

CREATE TABLE temperature_ranges (range VARCHAR2(15) NOT NULL UNIQUE, low_value NUMBER NOT NULL, high_value NUMBER NOT NULL);

BEGIN
  INSERT into temperature_ranges VALUES ('Hot',120, 212);
  INSERT into temperature_ranges VALUES('Room',60, 119);
  INSERT into temperature_ranges VALUES('Cool',32, 59);
  INSERT into temperature_ranges VALUES('Very cool',0, 31);
END;

BEGIN
  INSERT into beverages VALUES ('Coffee', 180);
  INSERT into beverages VALUES('Wine', 68);
  INSERT into beverages VALUES('Soda' ,45);
END;

<write one of each of the join types; there are many possibilities>
SECTION 2 LESSON 6 - Review of SQL Functions and Subqueries

Terminology
1. **Group Functions** These functions operate on a whole table or on a specific grouping of rows to return one result.
2. **GROUP BY** Clause used in a query to divide the rows in a table into smaller groups.
3. **ALL** Operator used when the outer query WHERE clause is designed to restrict rows based on all values returned from the inner query.
4. **Multiple Row subqueries** Use multiple row operators and return more than one row from the inner query.
5. **ANY** Operator used when the outer query WHERE clause is designed to restrict rows based on any value returned from the inner query.
6. **HAVING** Clause used in a query to restrict groups.
7. **Subquery** A SELECT statement that is embedded in a clause of another SQL statement.

Try It/Solve It
1. Write a SQL statement that will return the earliest independence date from `wf_countries`.
   ```sql
   SELECT MIN(date_of_independence)
   FROM wf_countries;
   ```
2. Without referring to the answer in question 1, write a SQL statement that lists the name of the country with the earliest independence date.
   ```sql
   SELECT country_name
   FROM wf_countries
   WHERE date_of_independence =
     (SELECT MIN(date_of_independence) FROM wf_countries);
   ```
3. Which country has the smallest area?
   **Vatican City**
   ```sql
   SELECT country_name
   FROM wf_countries
   WHERE area =
     (SELECT MIN(area) FROM wf_countries);
   ```
4. Write a SQL statement that lists the countries with the maximum highest elevation, along with the highest elevation value.

```
SELECT country_name, highest_elevation
FROM wf_countries
WHERE highest_elevation =
    (SELECT MAX (highest_elevation) FROM wf_countries);
```

5. List the name of each country and the number of languages spoken in it. Order the results by the number of languages, from the most to the least.

```
SELECT country_name, COUNT(language_id)
FROM wf_spoken_languages sl, wf_countries c
WHERE c.country_id = sl.country_id
GROUP BY country_name
ORDER BY count(language_id) DESC;
```

6. List the name of each language and the number of countries it is spoken in. Order the results by the number of countries, from the most to the least.

```
SELECT language_name, count(country_id)
FROM wf_spoken_languages sl, wf_languages l
WHERE l.language_id = sl.language_id
GROUP BY language_name
ORDER BY count(country_id) DESC;
```

7. List the name of each currency and the number of countries it is used in. Restrict the list to those currencies which are used in more than one country.

```
SELECT currency_name, COUNT(country_id)
FROM wf_currencies cc, wf_countries c
WHERE c.currency_code = cc.currency_code
GROUP BY currency_name
HAVING COUNT(country_id) >1;
```

8. Write a SQL statement that displays the name of all official languages.

```
SELECT language_name
FROM wf_languages
WHERE language_id IN
    (SELECT language_id FROM wf_spoken_languages
     WHERE upper(official) = 'Y');
```
9. List the names of countries in the Oceania region.

```
SELECT country_name  
    FROM wf_countries  
    WHERE region_id =  
        (SELECT region_id FROM wf_world_regions  
          WHERE region_name = 'Oceania');
```

10. List the name of each country whose name is alphabetically greater than the names of all countries in Western Europe (region_id 155). Use the ANY operator.

```
SELECT country_name  
    FROM wf_countries  
    WHERE country_name > ALL  
        (SELECT country_name FROM wf_countries  
          WHERE region_id = 155);
```
SECTION 2 LESSON 7 - Writing PL/SQL Executable Statements

Terminology
1. **Explicit conversion** Converts values from one data type to another by using built-in functions.
2. **Implicit conversion** Converts data types dynamically if they are mixed in a statement.

Try It/Solve It
1. Examine the following code and then answer the questions.

```sql
DECLARE
    x VARCHAR2(20);
BEGIN
    x := '123' + '456' ;
    DBMS_OUTPUT.PUT_LINE(x);
END;
```

A. What do you think the output will be when you run the above code?

**Students may think that the answer might be: 579 or 123456 or Error**

B. Now, run the code. What is the output?

579

C. In your own words, describe what happened when you ran the code. Did any implicit conversions take place?

**PL/SQL implicitly converted the VARCHAR2 values to the NUMBER format and added them.**

2. Write an anonymous PL/SQL block that uses the programmer’s full name and then returns the number of characters in the name.

```sql
DECLARE
    v_name VARCHAR2(50) := '<Enter your full name> ;
    v_length_name PLS_INTEGER;
BEGIN
    v_length_name := LENGTH(v_name);
    DBMS_OUTPUT.PUT_LINE(v_length_name);
END;
```
3. Write an anonymous PL/SQL block that uses today's date and outputs it in the format of ‘Month dd, yyyy’. Store the date in a DATE variable called my_date. Create another variable of the date type called v_last_day. Assign v_last_day to the last day of this month. Display the output.

```
DECLARE
    my_date     DATE := SYSDATE;
    v_last_day DATE;
BEGIN
    DBMS_OUTPUT.PUT_LINE(TO_CHAR(my_date, 'Month dd, yyyy'));
    v_last_day := LAST_DAY(my_date);
    DBMS_OUTPUT.PUT_LINE(v_last_day);
END;
```

4. Modify the program created in question 3 to add 45 days to today’s date and then calculate the number of months between the two dates.

```
DECLARE
    my_date DATE := SYSDATE;
    new_date DATE;
    v_months_between NUMBER;
BEGIN
    new_date := my_date + 45;
    v_months_between := MONTHS_BETWEEN(my_date, new_date);
    DBMS_OUTPUT.PUT_LINE(v_months_between);
END;
```
5. Examine the following code and then answer the questions.

```
DECLARE
  x NUMBER(6);
BEGIN
  x := 5 + 3 * 2 ;
  DBMS_OUTPUT.PUT_LINE(x);
END;
```

A. What do you think the output will be when you run the above code?

Students may think that the answer might be: 16 or 11

B. Now run the code. What is the output?

11

C. In your own words, explain the results.

The order of operations tells you that multiplication takes precedence over (that is, comes before) addition. Therefore, 3 * 2 is executed before 5 is added.

6. Examine the following code and then answer the question.

```
DECLARE
  v_number   NUMBER;
  v_boolean  BOOLEAN;
BEGIN
  v_number := 25;
  v_boolean := NOT(v_number > 30);
END;
```

What value is assigned to v_boolean?

TRUE. The condition (v_number > 30) is FALSE, and NOT FALSE = TRUE.
SECTION 2 LESSON 8 - Nested Blocks and Variable Scope

Terminology
1. ___Exception handling________ Allows clean separation of the error processing code from the executable code so that a program can continue operating in the presence of errors.
2. ___Qualifier________________ A label given to a block.
3. ___Variable scope___________ Consists of all the blocks in which the variable is either local (the declaring block) or global (nested blocks within the declaring block).
4. ___Exception propagating______ The exception reproduces itself in successive enclosing blocks until a handler is found or there are no more blocks to search in.
5. ___Variable visibility_________ The portion of the program where the variable can be accessed without using a qualifier.

Try It / Solve It
1. Evaluate the PL/SQL block below and determine the value of each of the following variables according to the rules of scoping.

   DECLARE
   weight  NUMBER(3) := 600;
   message VARCHAR2(255) := 'Product 10012';
   BEGIN

   DECLARE
   weight  NUMBER(3) := 1;
   message  VARCHAR2(255) := 'Product 11001';
   new_locn  VARCHAR2(50) := 'Europe';
   BEGIN
   weight := weight + 1;
   new_locn := 'Western ' || new_locn;
   -- Position 1 --
   END;

   weight := weight + 1;
   message := message || ' is in stock';
   -- Position 2 --
   END;

   A. The value of weight at position 1 is:
   2

   B. The value of new_locn at position 1 is:
   Western Europe
C. The value of weight at position 2 is:  
601

D. The value of message at position 2 is:  
Product 10012 is in stock

E. The value of new_locn at position 2 is:  
Out of range – new_locn is undefined in the outer block.

Students can test the accuracy of their answers, if desired, by entering and running the following code:

```
DECLARE
  weight      NUMBER(3) := 600;
  message   VARCHAR2(255) := 'Product 10012';
BEGIN
  DECLARE
    weight    NUMBER(3) := 1;
    message   VARCHAR2(255) := 'Product 11001';
    new_locn  VARCHAR2(50) := 'Europe';
  BEGIN
    weight := weight + 1;
    new_locn := 'Western ' || new_locn;
    -- Position 1 --
    DBMS_OUTPUT.PUT_LINE('At Position 1, weight = '||weight);
    DBMS_OUTPUT.PUT_LINE('At Position 1, new_locn= '||new_locn);
  END;
  weight := weight + 1;
  message := message || ' is in stock';
  -- Position 2 --
  DBMS_OUTPUT.PUT_LINE('At Position 2, weight = '||weight);
  DBMS_OUTPUT.PUT_LINE('At Position 2, message= '||message);
END;
```
2. Enter and run the following PL/SQL block, which contains a nested block. Look at the output and answer the questions.

DECLARE
    v_employee_id employees.employee_id%TYPE;
    v_job          employees.job_id%TYPE;
BEGIN
    SELECT employee_id, job_id INTO v_employee_id,  v_job
    FROM employees
    WHERE employee_id = 100;

DECLARE
    v_employee_id employees.employee_id%TYPE;
    v_job          employees.job_id%TYPE;
BEGIN
    SELECT employee_id, job_id INTO v_employee_id, v_job
    FROM employees
    WHERE employee_id = 103;
    DBMS_OUTPUT.PUT_LINE(v_employee_id||' is a '||v_job);
END;

    DBMS_OUTPUT.PUT_LINE(v_employee_id||' is a '||v_job);
END;

A. Why does the inner block display the job_id of employee 103, not employee 100?

Because although both declarations of v_job are in scope and in the inner block, the outer block’s declaration is not visible.

B. Why does the outer block display the job_id of employee 100, not employee 103?

Because the inner block’s declaration is out of scope in the outer block.
C. Modify the code to display the details of employee 100 in the inner block. Use block labels.

```plsql
<<outer_block>>
DECLARE
  v_employee_id employees.employee_id%TYPE;
  v_job employees.job_id%TYPE;
BEGIN
  SELECT employee_id, job_id INTO v_employee_id, v_job
  FROM employees
  WHERE employee_id = 100;
END;

<<inner_block>>
DECLARE
  v_employee_id employees.employee_id%TYPE;
  v_job employees.job_id%TYPE;
BEGIN
  SELECT employee_id, job_id INTO v_employee_id, v_job
  FROM employees
  WHERE employee_id = 103;
  DBMS_OUTPUT.PUT_LINE(outer_block.v_employee_id || ' is a ' || outer_block.v_job);
END;

DBMS_OUTPUT.PUT_LINE(v_employee_id || ' is a ' || v_job);
END;
```

3. Enter and run the following PL/SQL block. Explain the output. Note: the WHEN OTHERS handler successfully handles any type of exception which occurs.

```plsql
DECLARE
  v_number number(2);
BEGIN
  v_number := 9999;
EXCEPTION
  WHEN OTHERS THEN
    DBMS_OUTPUT.PUT_LINE('An exception has occurred');
END;
```

An exception has occurred because the 4-digit value 9999 is too large to be assigned to a NUMBER(2) variable. The block’s exception section has handled the exception successfully and displayed ‘An exception has occurred’. The exception has NOT been propagated back to the calling environment (Application Express) which therefore reports ‘Statement Processed’, meaning: success.
4. Modify the block in question 3 to omit the exception handler, then re-run the block.
   Explain the output.

```
DECLARE
  v_number     NUMBER(2);
BEGIN
  v_number := 9999;
END;
```

The block does not handle the exception, which therefore propagates back to Application Express. Application Express displays an ‘ORA-06502: PL/SQL: numeric or value error’ message.

5. Enter and run the following code and explain the output.

```
DECLARE
  v_number    NUMBER(4);
BEGIN
  v_number := 1234;

  DECLARE
    v_number     NUMBER(4);
  BEGIN
    v_number := 5678;
    v_number := 'A character string';
  END;
  EXCEPTION
    WHEN OTHERS THEN
      DBMS_OUTPUT.PUT_LINE('An exception has occurred');
      DBMS_OUTPUT.PUT_LINE('The number is: '||v_number);
  END;
```

The inner block’s attempt to assign a character string to a NUMBER variable causes an exception. The exception is not handled in the inner block, which therefore propagates the exception to the outer block. The outer block successfully handles the exception.

The number 1234 (not 5678) is displayed because the inner block’s v_number is out of scope in the outer block.
SECTION 2 LESSON 9 - Good Programming Practices

Terminology
No new vocabulary for this lesson.

Try It/Solve It
1. Enter and run the following PL/SQL block. It will execute correctly if you have entered it correctly, but it contains some examples of bad programming practice.

   A. Modify the block to use good programming practice, and re-run the block.

   B. Your modified block should contain examples of the following good programming practices: explicit data type conversions, meaningful and consistent variable names, use of %TYPE, upper and lowercase conventions, single and multi-line comments, and clear indentation.

   DECLARE
       myvar1    VARCHAR2(20);
       myvar2    number(4);
   BEGIN
       SELECT country_name INTO  myvar1
       FROM wf_countries WHERE country_id = 1246;
       myvar2 := '1234';
       MYVAR2 := myvar2 * 2;
       DBMS_OUTPUT.PUT_LINE(myvar1);
   End;

Students answers will vary, especially when inserting comments (there are many possibilities). A sample answer could be:

   DECLARE
       v_country_name  wf_countries.country_name%TYPE;
       v_number        NUMBER(4);
   BEGIN
       /* Read the country name of Barbados from the database
          and assign it to the first variable */
       SELECT country_name INTO v_country_name
       FROM wf_countries
       WHERE country_id = 1246;
       v_number := TO_NUMBER('1234');    -- or v_number := 1234;
       v_number := v_number * 2;
       DBMS_OUTPUT.PUT_LINE(v_country_name);
   END;