OBIECTIVE :

- ce este un bloc PL/SQL şi care sînt secţiunile lui;
- distinctia între variabilele PL/SQL şi variabilele non-PL/SQL
- declararea variabilelor PL/SQL
- executarea unui bloc PL/SQL

Structura unui bloc PL/SQL

PL/SQL este un limbaj structurat pe bloc, programele pot fi împărţite în blocuri logice. Un bloc PL/SQL este compus din pîna la 3 secţiuni: declarativă (opţională), executabilă (obligatorie) şi de tratare a excepţiilor (opţională).

Secţiunea	**Descriere**	**Includere**
Declarativă	conţine toate variabilele, constantele, cursoarele si excepţiile definite de utilizator referite în cele trei secţiuni	opţională
executabilă	conţine instrucţiuni SQL pentru manipularea datelor din baza de date şi instr.PL/SQL pentru manipularea datelor în cadrul blocului	obligatorie
de tratare a excepţiilor	specifică acţiunile de îndeplinit atunci cînd erori sau condiţii anormale apar în cadrul secăiunii executabile	opţională

Executarea blocurilor PL/SQL din SQL*Plus

- Se pune ; dupa fiecare instrucţiune SQL sau instr.de control PL/SQL
- Se pune / pe o linie noua pt.ca blocul anonim din bufferul SQL să fie rulat
- Se pune . pentru a închide bufferul SQL. Un bloc PL/SQL este tratat ca o unică continuă instrucţiune în bufferul SQL iar *delimitatorul ; nu închide şi nu determină execuţia conţinutului bufferului*.

Cuvintele cheie de delimitare a secţiunilor DECLARE, BEGIN si EXCEPTION nu sînt urmate de ;. END şi celelalte instrucţiuni PL/SQL au nevoie de acest delimitator.

Tipuri de blocuri

Orice unitate PL/SQL conţine unul sau mai multe blocuri, complet separate sau imbricate.
Blocurile anonime sînt blocuri nedenumite, declarate în acel punct în aplicaţie unde trebuie executate şi transmise motorului PL/SQL spre execuţie în momentul rulării.

Subprogramele sînt blocuri denumite ce pot primi parametri şi pot fi invocate. Pot fi declarate ca proceduri sau funcţii.

Subprogramele pot fi stocate la nivel de server sau de aplicaţie.

Constructii de program:

Bloc anonim	bloc PL/SQL scufundat într-o aplicaţie sau lansat în mod interactiv
Procedură sau funcţie stocată	bloc PL/SQL denumit, stocat în cadrul serverului ORACLE, ce poate primi parametri şi poate fi invocat în mod repetat prin nume
Procedură sau funcţie de aplicaţie	bloc PL/SQL denumit, definit într-o aplicaţie sau stocat într-o biblioteca partajată, ce poate primi parametri şi poate fi invocat în mod repetat prin nume
Pachet	modul PL/SQL ce grupează funcţii, proceduri şi identificatori
Declanşator al bazei de date	bloc PL/SQL asociat unei tabele a bazei de date, lansat automat în execuţie ca urmare a executării unei operaţii DML asupra tabelei
Declanşator de aplicaţie	bloc PL/SQL asociat unui eveniment în cadrul unei aplicaţii şi lansat în execuţie automat

Variabile bind:

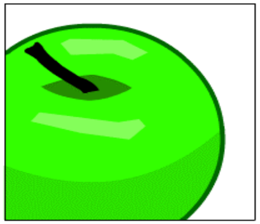
O variabilă bind este o variabilă declarată într-un mediu gazdă şi folosită apoi, în momentul execuţiei, pentru transferul de valori în sau din unul sau mai multe programe PL/SQL care o pot folosi ca pe orice altă variabila. Variabilele declarate în mediul gazadă (mediul apelant) pot fi referite în instrucţiunile PL/SQL, *cu excepţia cazurilor cînd instrucţiunea aparţine unei proceduri, funcţii sau unui pachet*.

In mediul SQL*Plus o variabila bind se declara prin comanda
VARIABLE *variable_name* **{NUMBER | CHAR}**
iar valoarea ei poate fi afisata cu comanda
PRINT *variable_name*

Pentru a fi referite in instructiunile PL/SQL variabilele bind se prefixeaza cu ':' spre a putea fi distinse de variabilele PL/SQL.

Using Variables in PL/SQL

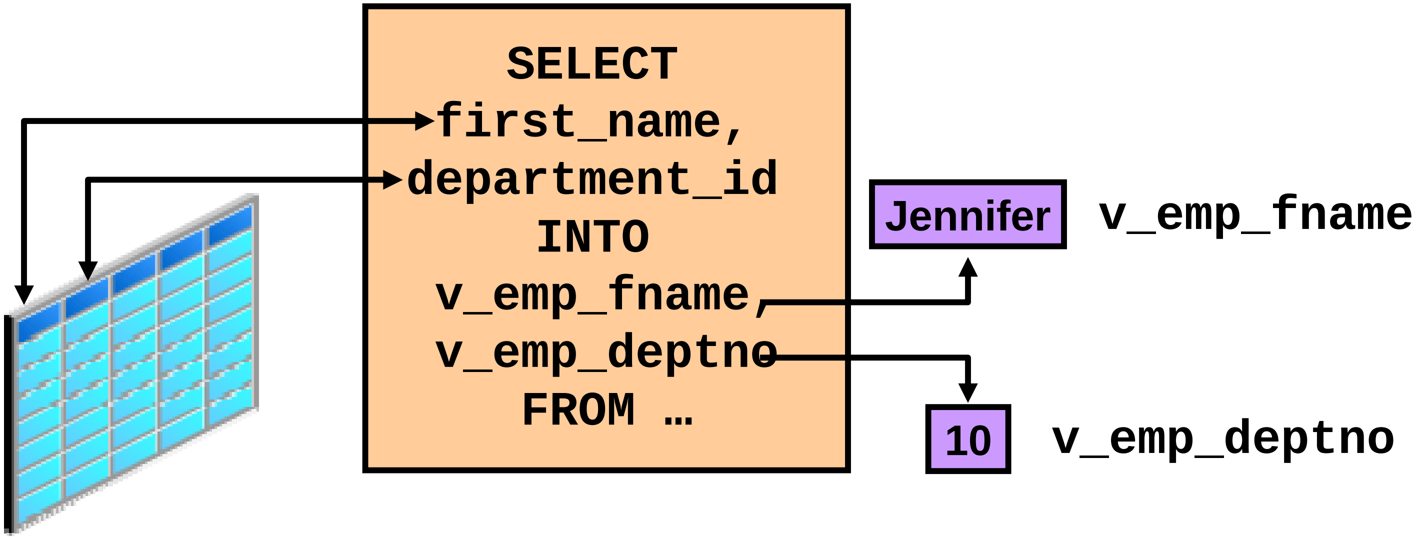
Tell Me/Show Me



Use of Variables

You can use variables for:

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



```
SELECT
first_name,
department_id
INTO
v_emp_fname,
v_emp_deptno
FROM …
```

Jennifer v_emp_fname

10 v_emp_deptno

Tell Me/Show Me

Handling Variables in PL/SQL

Variables are:

- Declared and initialized in the declarative section
- Used and assigned new values in the executable section

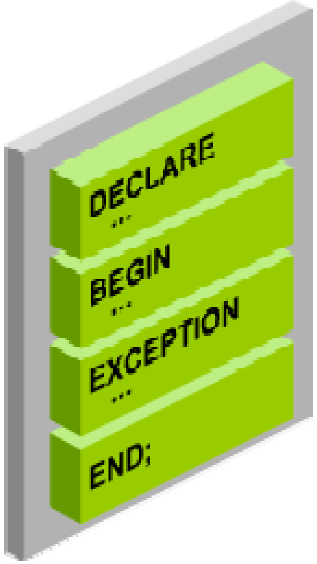
Variables can be:

- Passed as parameters to PL/SQL subprograms
- Assigned to hold the output of a PL/SQL subprogram

Tell Me/Show Me

Declaring and Initializing PL/SQL Variables

- All PL/SQL variables must be declared in the declaration section before referencing them in the PL/SQL block.
- The purpose of a declaration is to allocate storage space for a value, specify its data type, and name the storage location so that you can reference it.
- You can declare variables in the declarative part of any PL/SQL block, subprogram, or package.



Tell Me/Show Me

Declaring and Initializing Variables: Syntax

Syntax:

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

- *identifier* is the name of the variable
- CONSTANT constrains the variable so that its value cannot change; constants must be initialized.
- *datatype* is a scalar, composite, reference, or LOB data type. (This course covers only scalar, composite, and LOB data types.)
- NOT NULL constrains the variable so that it must contain a value. (NOT NULL variables must be initialized.)
- *Expr* is any PL/SQL expression that can be a literal expression, another variable, or an expression involving operators and functions.

Tell Me/Show Me

Declaring and Initializing Variables: Syntax (continued)

Syntax:

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

Conventions:

The *lowercase italic* represents variables or placeholders.

Brackets ([…]) enclose one or more optional items. Do not insert the brackets.

A vertical bar (|) represents a choice of two or more options within brackets. Do not insert the vertical **bar.**

Tell Me/Show Me

Declaring and Initializing Variables: Examples

```
DECLARE
  v_emp_hiredate    DATE;
  v_emp_deptno      NUMBER(2) NOT NULL := 10;
  v_location        VARCHAR2(13) := 'Atlanta';
  c_comm            CONSTANT NUMBER := 1400;
  v_population      INTEGER;
  v_book_type       VARCHAR2(20) DEFAULT 'fiction';
  v_artist_name     VARCHAR2(50);
  v_firstname       VARCHAR2(20):='Rajiv';
  v_lastname        VARCHAR2(20) DEFAULT 'Kumar';
  c_display_no      CONSTANT PLS_INTEGER := 20;
…
```

Tell Me/Show Me

Assigning Values in the Executable Section

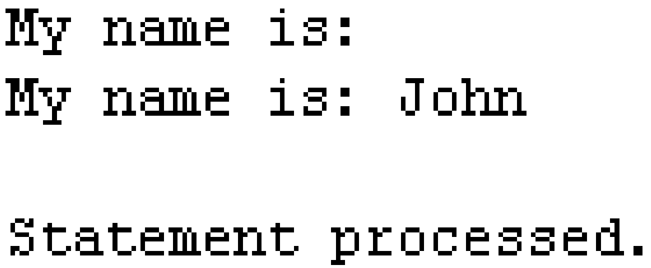
After a variable is declared, you can use it in the executable section of a PL/SQL block. For example, in the following block, the variable `v_myname` is declared in the declarative section of the block. You can access this variable in the executable section of the same block. What do you think the block will print?

```
DECLARE
  v_myname VARCHAR2(20);
BEGIN
  DBMS_OUTPUT.PUT_LINE('My name is: '||v_myname);
  v_myname := 'John';
  DBMS_OUTPUT.PUT_LINE('My name is: '||v_myname);
END;
```

Tell Me/Show Me

Assigning Values in the Executable Section (continued)

In this example, the value `John` is assigned to the variable in the executable section. The value of the variable is concatenated with the string `My name is:` . The output is:



```
My name is:
My name is: John

Statement processed.
```

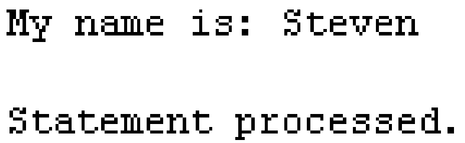
Tell Me/Show Me

Assigning Values in the Executable Section

In this block, the variable v_myname is declared and initialized in the declarative section. v_myname holds the value John after initialization. This value is manipulated in the executable section of the block.

```
DECLARE
  v_myname VARCHAR2(20):= 'John';
BEGIN
  v_myname := 'Steven';
  DBMS_OUTPUT.PUT_LINE('My name is: '||v_myname);
END;
```

The output is:

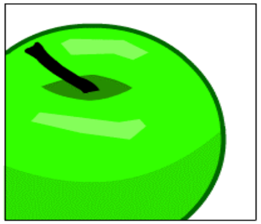


```
My name is: Steven

Statement processed.
```

Recognizing PL/SQL Lexical Units

Tell Me/Show Me



Lexical Units in a PL/SQL Block

Lexical units:

- Are the building blocks of any PL/SQL block
- Are sequences of characters including letters, digits, tabs, returns, and symbols
- Can be classified as:
  - Identifiers
  - Reserved words
  - Delimiters
  - Literals
  - Comments

Tell Me/Show Me

Identifiers

An identifier is the name given to a PL/SQL object, including any of the following:

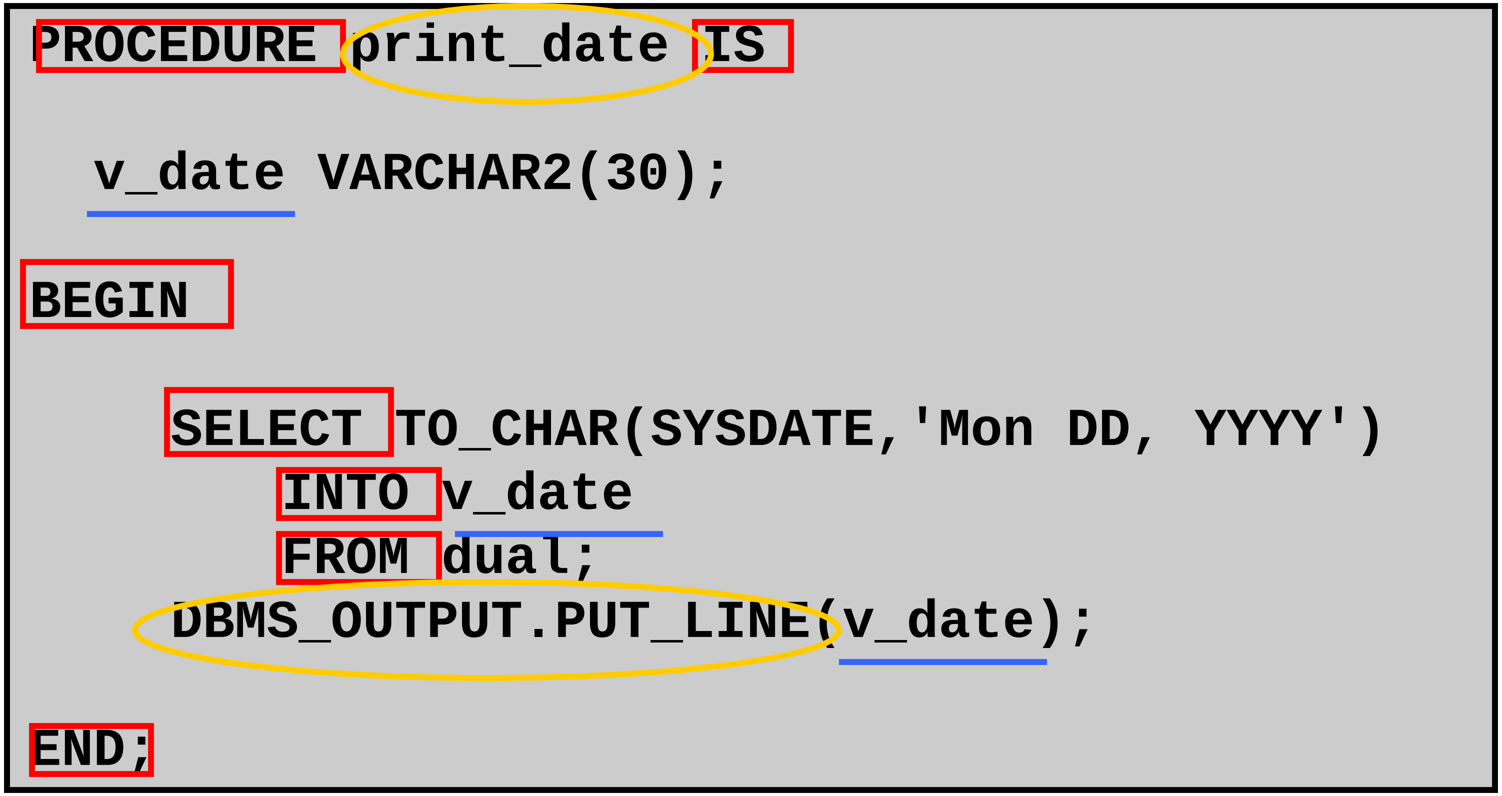
Procedure	Function	Variable
Exception	Constant	Package
Record	PL/SQL table	Cursor

(Do not be concerned if you do not know what all of the above objects are! You will learn about PL/SQL objects throughout this course.)

Tell Me/Show Me

Identifiers (continued)

The identifiers in the following PL/SQL code are highlighted::



```
PROCEDURE print_date IS

  v_date VARCHAR2(30);

BEGIN

      SELECT TO_CHAR(SYSDATE,'Mon DD, YYYY')
         INTO v_date
         FROM dual;
      DBMS_OUTPUT.PUT_LINE(v_date);

END;
```

Key: ○ Procedure ▁▁▁ Variable □ Reserved word

Tell Me/Show Me

Properties of an Identifier

Identifiers:

- Are up to 30 characters in length
- Must start with a letter
- Can include $ (dollar sign), _ (underscore), and # (pound sign/hash sign)
- Cannot contain spaces

Tell Me/Show Me

Valid and Invalid Identifiers

Examples of valid identifiers:

First_Name	LastName	address_1
ID#	Total_$	primary_department_contact

Examples of invalid identifiers:

First Name	Contains a space
Last-Name	Contains invalid "-"
1st_address_line	Begins with a number
Total_%	Contains invalid "%"
primary_building_department_contact	More than 30 characters

Tell Me/Show Me

Reserved Words

Reserved words are words that have special meaning to the Oracle database.

Reserved words cannot be used as identifiers in a PL/SQL program.

Tell Me/Show Me

Reserved Words (continued)

The following is a partial list of reserved words.

ALL	CREATE	FROM	MODIFY	SELECT
ALTER	DATE	GROUP	NOT	SYNONYM
AND	DEFAULT	HAVING	NULL	SYSDATE
ANY	DELETE	IN	NUMBER	TABLE
AS	DESC	INDEX	OR	THEN
ASC	DISTINCT	INSERT	ORDER	UPDATE
BETWEEN	DROP	INTEGER	RENAME	VALUES
CHAR	ELSE	INTO	ROW	VARCHAR2
COLUMN	EXISTS	IS	ROWID	VIEW
COMMENT	FOR	LIKE	ROWNUM	WHERE

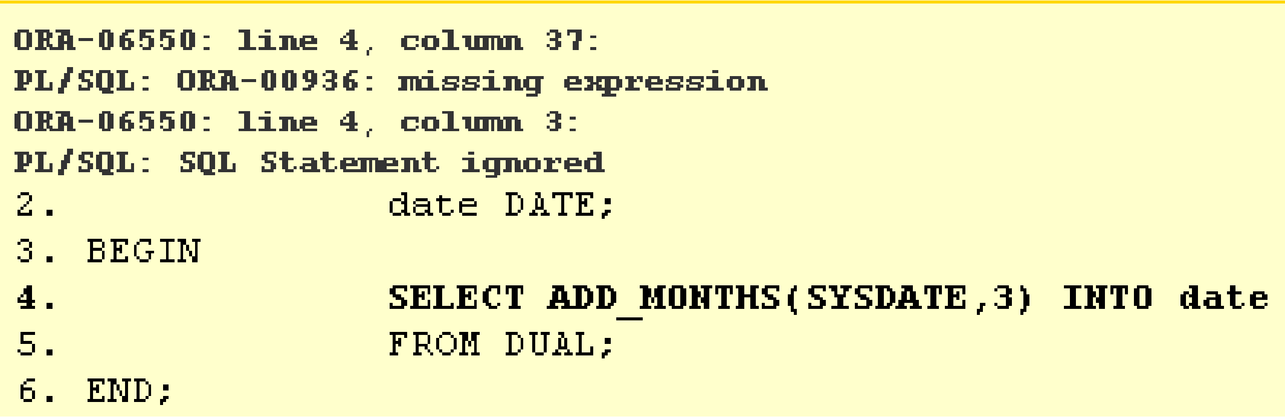
Note: For more information, refer to the “*PL/SQL User’s Guide and Reference*.”

Tell Me/Show Me

Reserved Words (continued)

What happens when you try to use a reserved word as an identifier in a PL/SQL program?

```
DECLARE
     date DATE;
BEGIN
     SELECT ADD_MONTHS(SYSDATE,3) INTO date
     FROM dual;
END;
```



```
ORA-06550: line 4, column 37:
PL/SQL: ORA-00936: missing expression
ORA-06550: line 4, column 3:
PL/SQL: SQL Statement ignored
2.              date DATE;
3.  BEGIN
4.              SELECT ADD_MONTHS(SYSDATE,3) INTO date
5.              FROM DUAL;
6.  END;
```

Tell Me/Show Me

Delimiters

Delimiters are symbols that have special meaning to the Oracle database.

Simple delimiters

Symbol	Meaning
+	Addition operator
–	Subtraction/negation operator
*	Multiplication operator
/	Division operator
=	Equality operator
'	Character string delimiter
;	Statement terminator

Compound delimiters

Symbol	Meaning
<>	Inequality operator
!=	Inequality operator
||	Concatenation operator
--	Single-line comment indicator
/*	Beginning comment delimiter
*/	Ending comment delimiter
:=	Assignment operator

Tell Me/Show Me

Literals

A literal is an explicit numeric, character string, date, or Boolean value that is not represented by an identifier.

Literals are classified as:

- Character (also known as string literals)
- Numeric
- Boolean

Tell Me/Show Me

Character Literals

- Character literals include all the printable characters in the PL/SQL character set: letters, numerals, spaces, and special symbols.
- Character literals have the data type `CHAR` and must be enclosed in single quotation marks.
- Character literals can be composed of zero or more characters from the PL/SQL character set.
- Character literals are case sensitive and, therefore, `PL/SQL` is not equivalent to `pl/sql`.

```
v_first_name := 'John';
v_classroom  := '12C';
v_date_today := '20-MAY-2005';
```

Tell Me/Show Me

Numeric Literals

- Values that represent an integer or real value are numeric literals
- You can represent numeric literals either by a simple value (for example, –32.5) or by a scientific notation (for example, 2E5, meaning 2* (10 to the power of 5) = 200000).
- Examples: 428, 1.276, 2.09E14

```
v_elevation        := 428;
v_order_subtotal := 1025.69;
v_growth_rate     := .56;
v_distance_sun_to_centauri := 4.3E13;
```

Tell Me/Show Me

Boolean Literals

- Values that are assigned to Boolean variables are Boolean literals. They are not surrounded by quotes.
- TRUE, FALSE, and NULL are Boolean literals or keywords.

```
v_new_customer             := FALSE;
v_paid_in_full             := TRUE;
v_authorization_approved := FALSE;
v_high_school_diploma     := NULL;
v_island                   := FALSE;
```

Tell Me/Show Me

Comments

Comments explain what a piece of code is trying to achieve. Well-placed comments are extremely valuable for code readability and future code maintenance. It is good programming practice to comment code.

Comments are ignored by PL/SQL. They make no difference to how a PL/SQL block executes or the results it displays.

Tell Me/Show Me

Syntax for Commenting Code

Commenting a single line:

- Two dashes **--** are used for commenting a single line.

Commenting multiple lines:

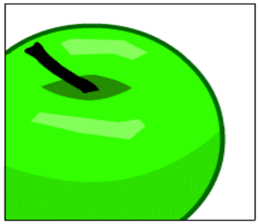
- /* */ is used for commenting multiple lines.

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

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

Recognizing Data Types

Tell Me/Show Me



PL/SQL Data Types

- A data type specifies a storage format, constraints, and a valid range of values.
- PL/SQL supports five categories of data type:
  - Scalar: Holds a single value.
  - Composite: Contains internal elements that are either scalar (record) or composite (record and table).
  - Large object (LOB): Holds values, called locators that specify the location of large objects (such as graphic images) that are stored out of line.
  - Reference: Holds values, called pointers that point to a storage location.
  - Object: Is a schema object with a name, attributes, and methods. An object data type is similar to the class mechanism supported by C++ and Java.

Tell Me/Show Me

Scalar Data Types

- Hold a single value
- Have no internal components
- Can be classified into four categories:
  - Character
  - Number
  - Date
  - Boolean

TRUE

25-JAN-01

Atlanta

256120.08

Tell Me/Show Me

Scalar Data Types: Character (or String)

CHAR [(*maximum_length*)]	Base type for fixed-length character data up to 32,767 bytes. If you do not specify a *maximum_length*, the default length is set to 1.
VARCHAR2 (*maximum_length*)	Base type for variable-length character data up to 32,767 bytes. There is no default size for VARCHAR2 variables and constants.
LONG	Character data of variable length (a bigger version of the VARCHAR2 data type).
LONG RAW	Raw binary data of variable length (not interpreted by PL/SQL).

Tell Me/Show Me

Scalar Data Types: Number

NUMBER [(*precision, scale*)]	Number having precision *p* and scale *s*. The precision *p* can range from 1 to 38. The scale *s* can range from –84 to 127.
BINARY_INTEGER	Base type for signed integers between -2,147,483,647 and 2,147,483,647.
PLS_INTEGER	Base type for signed integers between -2,147,483,647 and 2,147,483,647. PLS_INTEGER and BINARY_INTEGER values require less storage and are faster than NUMBER values.
BINARY_FLOAT BINARY_DOUBLE	New data types introduced in Oracle Database 10*g*. They represent a floating-point number in the IEEE 754 format. BINARY_FLOAT requires 5 bytes to store the value and BINARY_DOUBLE requires 9 bytes.

Tell Me/Show Me

Scalar Data Types: Date

DATE	Base type for dates and times. DATE values include the time of day in seconds since midnight. The range for dates is between 4712 B.C. and A.D. 9999.
TIMESTAMP	The TIMESTAMP data type, which extends the DATE data type, stores the year, month, day, hour, minute, second, and fraction of seconds.
TIMESTAMP WITH TIME ZONE	The TIMESTAMP WITH TIME ZONE data type, which extends the TIMESTAMP data type, includes a time-zone displacement—that is, the difference (in hours and minutes) between local time and Coordinated Universal Time (UTC), formerly known as Greenwich Mean Time.

Tell Me/Show Me

Scalar Data Types: Date (continued)

TIMESTAMP WITH LOCAL TIME ZONE	This data type differs from TIMESTAMP WITH TIME ZONE in that when you insert a value into a database column, the value is normalized to the database time zone, and the time-zone displacement is not stored in the column. When you retrieve the value, the Oracle server returns the value in your local session time zone.
INTERVAL YEAR TO MONTH	You use the INTERVAL YEAR TO MONTH data type to store and manipulate intervals of years and months.
INTERVAL DAY TO SECOND	You use the INTERVAL DAY TO SECOND data type to store and manipulate intervals of days, hours, minutes, and seconds.

Tell Me/Show Me

Scalar Data Types: Boolean

BOOLEAN	Base type that stores one of the three possible values used for logical calculations: TRUE, FALSE, or NULL.

Tell Me/Show Me

Composite Data Types

A scalar type has no internal components. A composite type has internal components that can be manipulated individually.

Composite data types include the following:

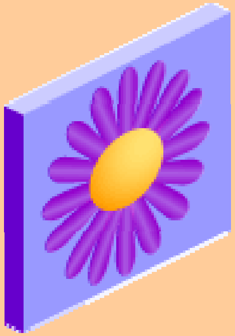
- TABLE
- RECORD
- NESTED TABLE
- VARRAY

TABLE and RECORD data types are covered later in this course.

Tell Me/Show Me

Composite Data Types (continued)

PL/SQL Record Structure

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

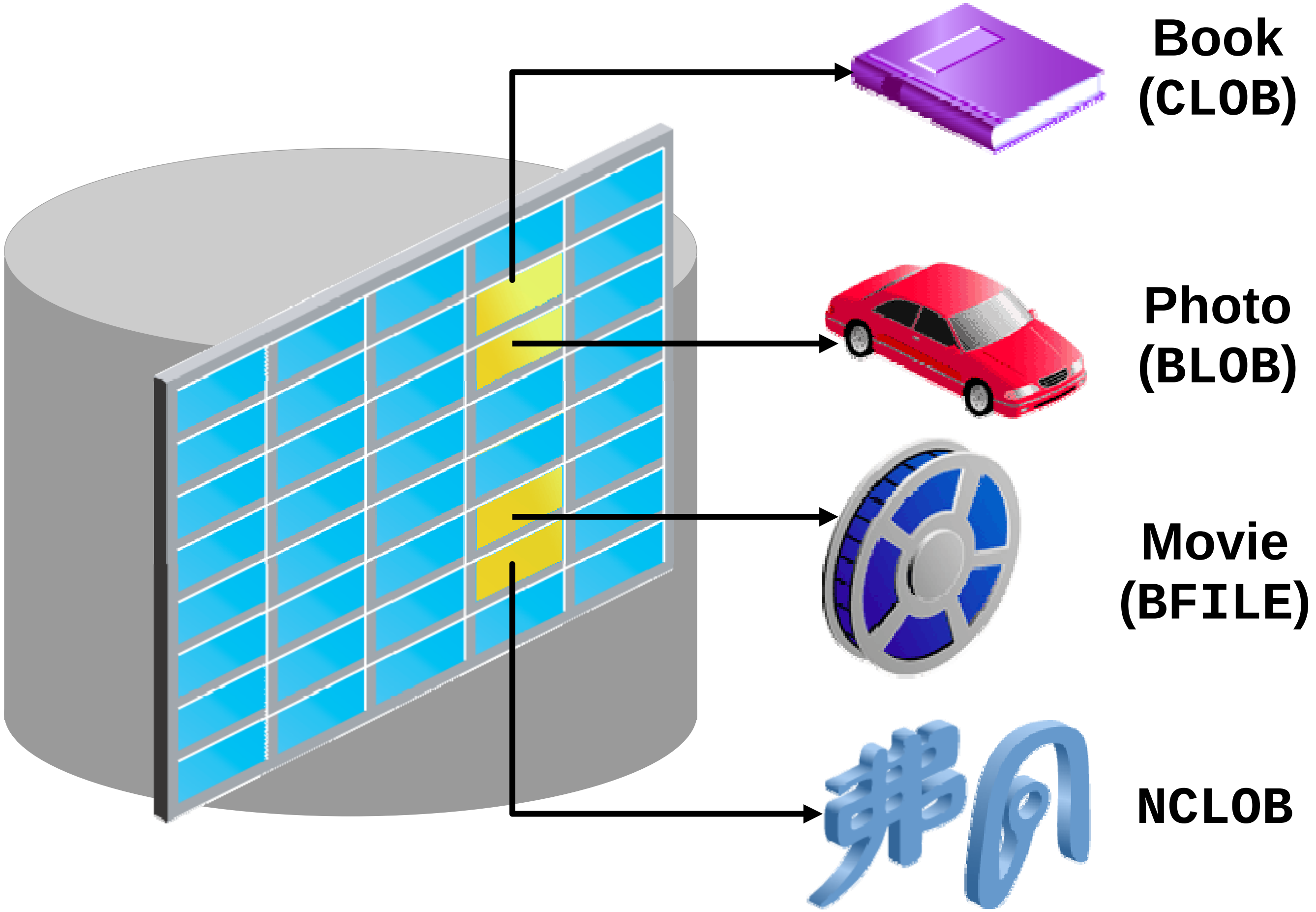
Tell Me/Show Me

LOB Data Type

- Large objects (LOBs) are meant to store a large amount of data.
- A database column can be of the LOB category.
- There are several categories of LOB data types:
  - Character large object (CLOB)
  - Binary large object (BLOB)
  - Binary file (BFILE)
  - National language character large object (NCLOB)
- LOB data types enable you to store blocks of unstructured data up to 4 gigabytes in size.
- LOB data types enable efficient, random, piece-wise access to the data and can be attributes of an object type.

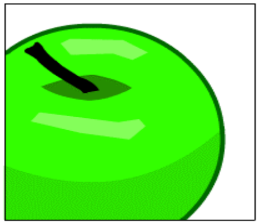
Tell Me/Show Me

LOB Data Type (continued)



Using Scalar Data Types

Tell Me/Show Me



Declaring Character Variables

Character data types include CHAR, VARCHAR2, and LONG.

```
DECLARE
  v_emp_job          VARCHAR2(9);
  v_order_no         VARCHAR2(6);
  v_product_id       VARCHAR2(10);
  v_rpt_body_part    LONG;
…
```

Tell Me/Show Me

Declaring Number Variables

Number data types include NUMBER, PLS_INTEGER, BINARY_INTEGER, and BINARY_FLOAT. In the syntax, CONSTANT constrains the variable so that its value cannot change. Constants must be initialized.

INTEGER is an alias for NUMBER(38,0).

```
DECLARE
   v_dept_total_sal   NUMBER(9,2) := 0;
   v_count_loop       INTEGER := 0;
   c_tax_rate         CONSTANT NUMBER(3,2) := 8.25;
   …
```

Tell Me/Show Me

Declaring Date Variables

Date data types include `DATE`, `TIMESTAMP`, and `TIMESTAMP WITH TIMEZONE`.

```
DECLARE
  v_orderdate           DATE := SYSDATE + 7;
  v_natl_holiday        DATE;
  v_web_sign_on_date    TIMESTAMP;
…
```

Tell Me/Show Me

Declaring Boolean Variables

Boolean is a data type that stores one of the three possible values used for logical calculations: TRUE, FALSE, or NULL.

```
DECLARE
  v_valid            BOOLEAN NOT NULL := TRUE;
  v_is_found         BOOLEAN := FALSE;
  v_underage         BOOLEAN;
…
```

Tell Me/Show Me

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 operator `NOT` to check the variable values.
- The variables always yield `TRUE`, `FALSE`, or `NULL`.
- You can use arithmetic, character, and date expressions to return a Boolean value.

Tell Me/Show Me

Guidelines for Declaring and Initializing PL/SQL Variables

- Use meaningful names and follow naming conventions.
- Declare one identifier per line for better readability, code maintenance, and easier commenting.
- Use the `NOT NULL` constraint when the variable must hold a value.
- Avoid using column names as identifiers.

```
DECLARE
  country_id   CHAR(2);
BEGIN
  SELECT country_id
   INTO country_id
   FROM countries
   WHERE country_name = 'Canada';
END;
```

Tell Me/Show Me

Anchoring Variables with the %TYPE Attribute

Rather than hard-coding the data type and precision of a variable, you can use the %TYPE attribute to declare a variable according to another previously declared variable or database column.

The %TYPE attribute is most often used when the value stored in the variable is derived from a table in the database.

When you use the %TYPE attribute to declare a variable, you should prefix it with the database table and column name.

Tell Me/Show Me

%TYPE Attribute

Look at this database table and the PL/SQL block that uses it:

```
CREATE TABLE myemps (
   emp_name           VARCHAR2(6),
   emp_salary         NUMBER(6,2));

DECLARE
  v_emp_salary   NUMBER(6,2);
BEGIN
  SELECT emp_salary INTO v_emp_salary
    FROM myemps WHERE emp_name = 'Smith';
END;
```

This PL/SQL block stores the correct salary in the `v_emp_salary` variable. But what if the table column is altered later?

Tell Me/Show Me

%TYPE Attribute (continued)

The %TYPE attribute:

- Is used to automatically give a variable the same data type and size as:
  - A database column definition
  - Another declared variable
- Is prefixed with either of the following:
  - The database table and column
  - The name of the other declared variable

Tell Me/Show Me

Declaring Variables with the %TYPE Attribute

Syntax:

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

Examples:

```
...
  v_emp_lname      employees.last_name%TYPE;
  v_balance        NUMBER(7,2);
  v_min_balance    v_balance%TYPE := 1000;
...
```

Tell Me/Show Me

Advantages of the %TYPE Attribute

- You can avoid errors caused by data type mismatch or wrong precision.

- You need not change the variable declaration if the column definition changes. That is, if you have already declared some variables for a particular table without using the %TYPE attribute, then the PL/SQL block can return errors if the column for which the variable declared is altered.

- When you use the %TYPE attribute, PL/SQL determines the data type and size of the variable when the block is compiled. This ensures that such a variable is always compatible with the column that is used to populate it.

Tell Me/Show Me

%TYPE Attribute

Look again at the database table and the PL/SQL block:

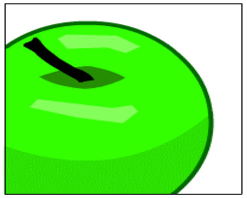
```
CREATE TABLE myemps (
  emp_name          VARCHAR2(6),
  emp_salary        NUMBER(6,2));

DECLARE
  v_emp_salary   myemps.emp_salary%TYPE;
BEGIN
  SELECT emp_salary INTO v_emp_salary
  FROM myemps WHERE emp_name = 'Smith';
END;
```

Now the PL/SQL block continues to work correctly even if the column data type is altered later.

Writing PL/SQL Executable Statements

Tell Me/Show Me



Assigning New Values to Variables

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

```
v_name       := 'Henderson';
v_start_date := '12-DEC-2005';
```

- Statements can continue over several lines.

```
v_quote := 'The only thing that we can
know is that we know nothing and that
is the highest flight of human
reason.';
```

- Numbers can be simple values or scientific notation.

```
v_my_integer := 100;
v_my_sci_not := 2E5;
```

(2E5 meaning 2x10 to the power of 5 = 200,000)

Tell Me/Show Me

SQL Functions in PL/SQL

You are already familiar with functions in SQL statements. For example:

```
SELECT country_name, LAST_DAY(date_of_independence)
  FROM wf_countries
  WHERE date_of_independence IS NOT NULL;
```

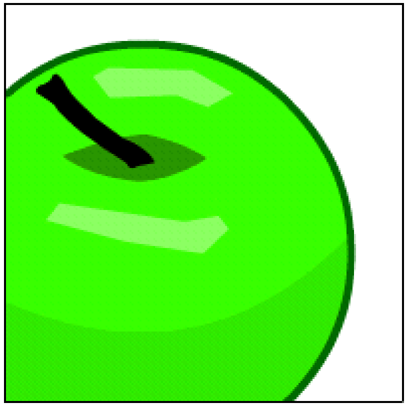
You can also use these functions in PL/SQL procedural statements. For example:

```
DECLARE
  v_last_day DATE;
BEGIN
  v_last_day := LAST_DAY(SYSDATE);
  DBMS_OUTPUT.PUT_LINE(v_last_day);
END;
```

Tell Me/Show Me

SQL Functions in PL/SQL

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



Tell Me/Show Me

Character Functions

Valid character functions in PL/SQL include:

ASCII	LENGTH	RPAD
CHR	LOWER	RTRIM
CONCAT	LPAD	SUBSTR
INITCAP	LTRIM	TRIM
INSTR	REPLACE	UPPER

This is not an exhaustive list. Refer to the Oracle documentation for the complete list.

Tell Me/Show Me

Examples of Character Functions

- Get the length of a string:

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

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

- Convert the name of the country capitol to upper case:

```
v_capitol_name:= UPPER(v_capitol_name);
```

- Concatenate the first and last names:

```
v_emp_name:= v_first_name||' '||v_last_name;
```

Tell Me/Show Me

Number Functions

Valid number functions in PL/SQL include:

ABS	EXP	ROUND
ACOS	LN	SIGN
ASIN	LOG	SIN
ATAN	MOD	TAN
COS	POWER	TRUNC

This is not an exhaustive list. Refer to the Oracle documentation for the complete list.

Tell Me/Show Me

Examples of Number Functions

- Get the sign of a number:

```
DECLARE
  v_my_num BINARY_INTEGER :=-56664;
BEGIN
  DBMS_OUTPUT.PUT_LINE(SIGN(v_my_num));
END;
```

- Round a number to 0 decimal places:

```
DECLARE
  v_median_age NUMBER(6,2);
BEGIN
  SELECT median_age INTO v_median_age
    FROM wf_countries WHERE country_id=27;
  DBMS_OUTPUT.PUT_LINE(ROUND(v_median_age,0));
END;
```

Tell Me/Show Me

Date Functions

Valid date functions in PL/SQL include:

`ADD_MONTHS`	`MONTHS_BETWEEN`
`CURRENT_DATE`	`ROUND`
`CURRENT_TIMESTAMP`	`SYSDATE`
`LAST_DAY`	`TRUNC`

This is not an exhaustive list. Refer to the Oracle documentation for the complete list.

Tell Me/Show Me

Examples of Date Functions

- Add months to a date:

```
DECLARE
  v_new_date   DATE;
  v_num_months NUMBER := 6;
BEGIN
  v_new_date := ADD_MONTHS(SYSDATE,v_num_months);
  DBMS_OUTPUT.PUT_LINE(v_new_date);
END;
```

- Calculate the number of months between two dates:

```
DECLARE
  v_no_months  PLS_INTEGER:=0;
BEGIN
  v_no_months := MONTHS_BETWEEN('31-JAN-06','31-MAY-05');
  DBMS_OUTPUT.PUT_LINE(v_no_months);
END;
```

Tell Me/Show Me

Data-Type Conversion

In any programming language, converting one data type to another is a common requirement. PL/SQL can handle such conversions with scalar data types. Data-type conversions can be of two types:

- Implicit conversions
- Explicit conversions

Tell Me/Show Me

Implicit Conversions

In implicit conversions, PL/SQL attempts to convert data types dynamically if they are mixed in a statement. Implicit conversions can happen between many types in PL/SQL, as illustrated by the following chart.

	DATE	LONG	NUMBER	PLS_INTEGER	VARCHAR2
DATE	N/A	X			X
LONG		N/A			X
NUMBER		X	N/A	X	X
PLS_INTEGER		X	X	N/A	X
VARCHAR2	X	X	X	X	N/A

Tell Me/Show Me

Example of Implicit Conversion

Consider the following example:

```
DECLARE
  v_salary            NUMBER(6):=6000;
  v_sal_increase      VARCHAR2(5):='1000';
  v_total_salary      v_salary%TYPE;
BEGIN
  v_total_salary:= v_salary + v_sal_increase;
  DBMS_OUTPUT.PUT_LINE(v_total_salary);
END;
```

In this example, the variable `v_sal_increase` is of type VARCHAR2. While calculating the total salary, PL/SQL first converts `v_sal_increase` to NUMBER and then performs the operation. The result of the operation is the `NUMBER` type.

Tell Me/Show Me

Drawbacks of Implicit Conversions

At first glance, implicit conversions might seem useful; however, there are several drawbacks:

- Implicit conversions can be slower.
- When you use implicit conversions, you lose control over your program because you are making an assumption about how Oracle handles the data. If Oracle changes the conversion rules, then your code can be affected.
- Implicit conversion rules depend upon the environment in which you are running. For example, the date format varies depending on the language setting and installation type. Code that uses implicit conversion might not run on a different server or in a different language.
- Code that uses implicit conversion is harder to read and understand.

Tell Me/Show Me

Drawbacks of Implicit Conversions

It is the programmer's responsibility to ensure that values can be converted. For instance, PL/SQL can convert the CHAR value '02-JUN-92' to a DATE value, but cannot convert the CHAR value ‘Yesterday' to a DATE value. Similarly, PL/SQL cannot convert a VARCHAR2 value containing alphabetic characters to a NUMBER value.

Valid?	Statement
Yes	`v_new_date  DATE := '02-JUN-1992';`
No	`v_new_date  DATE := 'Yesterday';`
Yes	`v_my_number NUMBER := '123';`
No	`v_my_number NUMBER := 'abc';`

Tell Me/Show Me

Explicit Conversions

Explicit conversions convert values from one data type to another by using built-in functions. Examples of conversion functions include:

TO_NUMBER()	ROWIDTONCHAR()
TO_CHAR()	HEXTORAW()
TO_CLOB()	RAWTOHEX()
CHARTOROWID()	RAWTONHEX()
ROWIDTOCHAR()	TO_DATE()

Tell Me/Show Me

Examples of Explicit Conversions

TO_CHAR

```
BEGIN
  DBMS_OUTPUT.PUT_LINE(TO_CHAR(SYSDATE,'Month YYYY'));
END;
```

TO_DATE

```
BEGIN
  DBMS_OUTPUT.PUT_LINE(TO_DATE('April-1999','Month-YYYY'));
END;
```

Tell Me/Show Me

Examples of Explicit Conversions (continued)

TO_NUMBER

```
DECLARE
  v_a VARCHAR2(10) := '-123456';
  v_b VARCHAR2(10) := '+987654';
  v_c PLS_INTEGER;
BEGIN
  v_c := TO_NUMBER(v_a) + TO_NUMBER(v_b);
  DBMS_OUTPUT.PUT_LINE(v_c);
END;
```

Tell Me/Show Me

Data Type Conversion Example

1

```
v_date_of_joining DATE:= '02-Feb-2000';
```

2

```
v_date_of_joining DATE:= 'February 02,2000';
```

3

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

Tell Me/Show Me

Operators in PL/SQL

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

Same as in SQL

- Exponential operator (**)

The operations within an expression are performed in a particular order depending on their precedence (priority).

Tell Me/Show Me

Operators in PL/SQL

The following table shows the default order of operations from high priority to low priority:

Operator	Operation
`**`	Exponentiation
`+, -`	Identity, negation
`*, /`	Multiplication, division
`+, -, ||`	Addition, subtraction, concatenation

`=, <, >, <=, >=, <>, !=, ~=, ^=, IS NULL,`
`LIKE, BETWEEN, IN`
	Comparison

`NOT`	Logical negation
`AND`	Conjunction
`OR`	Inclusion

Tell Me/Show Me

Operators in PL/SQL

Examples:

- Increment the counter for a loop.

```
v_loop_count := v_loop_count + 1;
```

- Set the value of a Boolean flag.

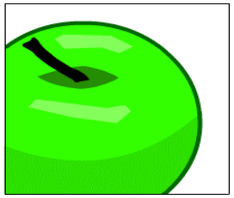
```
v_good_sal := v_sal BETWEEN 50000 AND 150000;
```

- Validate whether an employee number contains a value.

```
v_valid      := (v_empno IS NOT NULL);
```

Nested Blocks and Variable Scope

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;
```

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';
  …
```

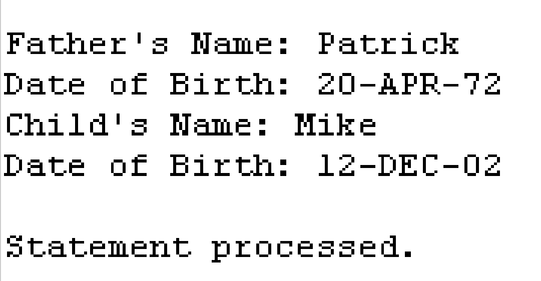
Labeling is not limited to the outer block; you can label any block.

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;
```

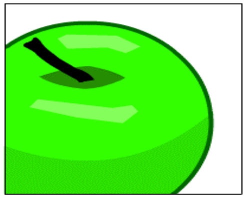


```
Father's Name: Patrick
Date of Birth: 20-APR-72
Child's Name: Mike
Date of Birth: 12-DEC-02

Statement processed.
```

Good Programming Practices

Tell Me/Show Me



Programming Practices

You've already learned several good programming practices in this course:

- Conversions:
  - Do not rely on implicit data type conversions because they can be slower and the rules can change in later software releases.
- Declaring and initializing PL/SQL variables:
  - Use meaningful names
  - Declare one identifier per line for better readability and code maintenance.
  - Use the `NOT NULL` constraint when the variable must hold a value.
  - Avoid using column names as identifiers.
  - Use the `%TYPE` attribute to declare a variable according to another previously declared variable or database column.

Tell Me/Show Me

Programming Guidelines

Other programming guidelines include:

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

Tell Me/Show Me

Commenting Code

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

Example:

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

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

Tell Me/Show Me

Case Conventions

The following table provides guidelines for writing code in uppercase or lowercase to help you distinguish keywords from named objects.

Category	Case Convention	Examples
SQL keywords	Uppercase	`SELECT`, `INSERT`
PL/SQL keywords	Uppercase	`DECLARE`, `BEGIN`, `IF`
Data types	Uppercase	`VARCHAR2`, `BOOLEAN`
Identifiers and parameters	Lowercase	`v_sal`, `emp_cursor`, `g_sal`, `p_empno`
Database tables and columns	Lowercase	`employees`, `employee_id`, `department_id`

Tell Me/Show Me

Naming Conventions

The naming of identifiers should be clear, consistent, and unambiguous. One commonly-used convention is to name:

- Variables starting with v_
- Constants starting with c_
- Parameters (passed to procedures and functions) starting with p_

Examples: `v_date_of_birth`; `c_tax_rate`; `p_empno`;

Tell Me/Show Me

Indenting Code

For clarity, indent each level of code.

Examples:

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

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