+ ENTRENAMIENTO
EXPERIENCIA

GALAXY
TRAINING

ORACLE DATABASE 19c
DEVELOPER: PL/SQL

Ing. Cesar Hijar
Instructor

chijar@galaxy.edu.pe

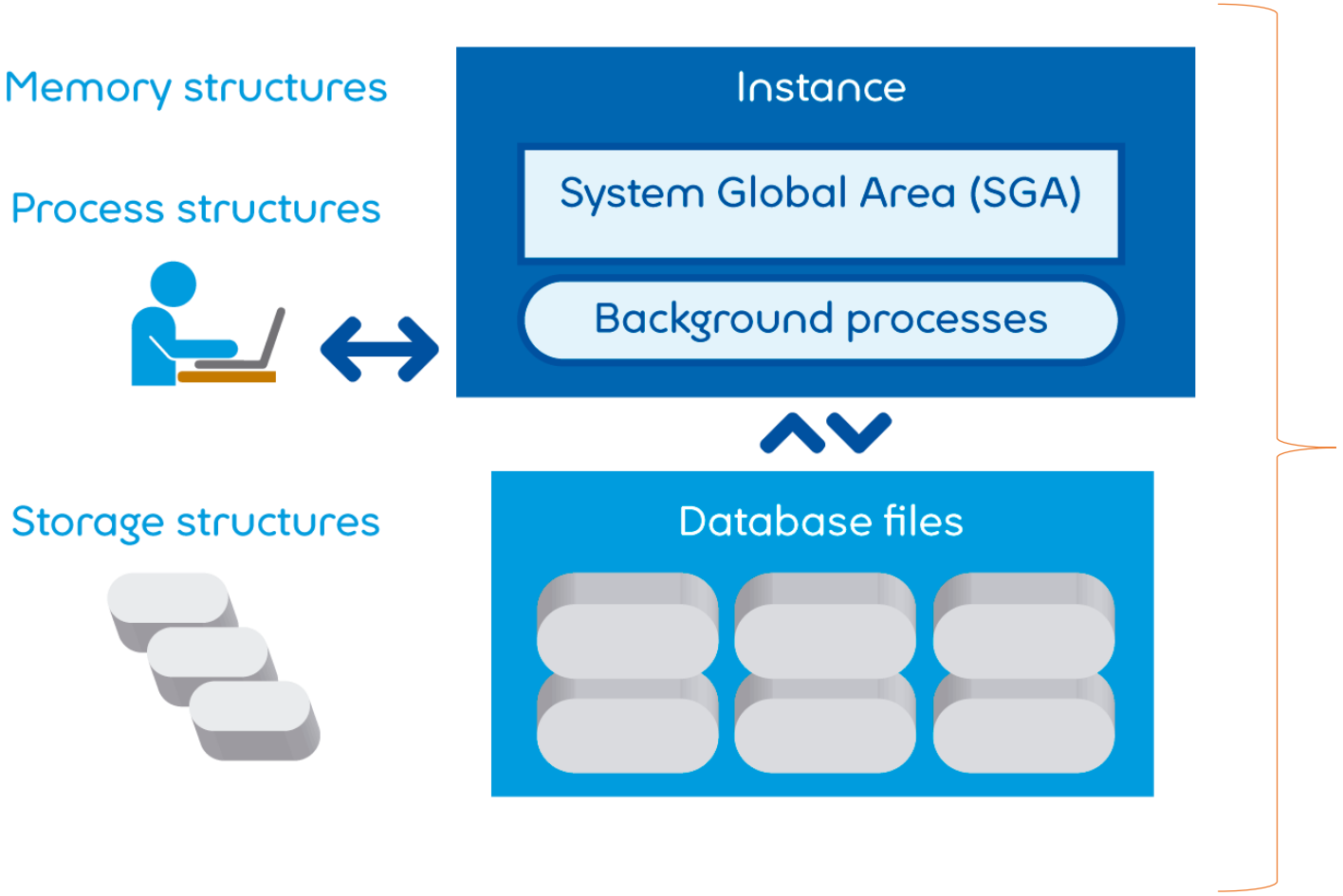
ÍNDICE

Una instancia Oracle:

- Es un medio para acceder a una base de datos Oracle.

- Siempre abre una y solo una base de datos.

- Consiste de estructuras en meoria y estructuras de procesos.



Arquitectura de la base de datos 12c onwards

Why to use PL/SQL ?

Example: Updating salary according to department number

Dept 10 → raise $100
Dept 20 → raise $150
Dept 30 → raise $200
Dept 40 → raise $240

In SQL
You will do multiple SQL statements

```
Update emp
Set sal=sal+100
Where dept_id=10;

Update emp
Set sal=sal+150
Where dept_id=20;
......
.....
```

In PL/SQL
You can write procedure to do this

```
Create procedure update_sal
( p_dept_id number , p_amount number )
Is
Begin
..........
.......
.......
End;
```

Overview PL/SQL

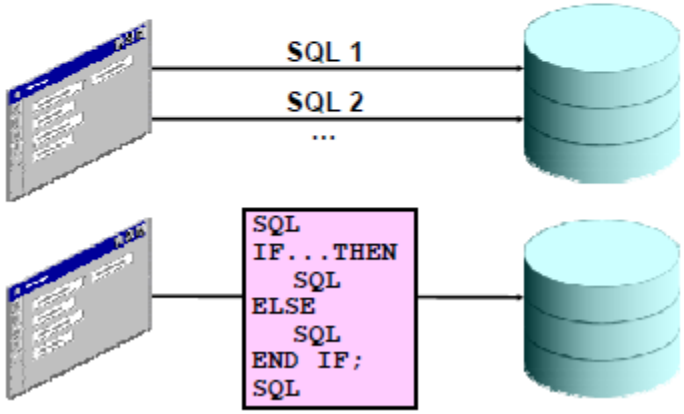
PL/SQL:

- Provides a block structure for executable units of code. Maintenance of code is made easier with such a well-defined structure.
- Provides procedural constructs such as:
  - Variables, constants, and data types
  - Control structures such as conditional statements and loops
  - Reusable program units that are written once and executed many times

Overview PL/SQL

Benefits of PL/SQL

- Integration of procedural constructs with SQL
- Improved performance



Benefits of PL/SQL

- Modularized program development
- Integration with Oracle tools
- Portability
- Exception handling

PL/SQL Block Structure

- DECLARE (optional)
  - Variables, cursors, user-defined exceptions
- BEGIN (mandatory)
  - SQL statements
  - PL/SQL statements
- EXCEPTION (optional)
  - Actions to perform when exceptions occur
- END; (mandatory)

PL/SQL Block Structure (continued)

In a PL/SQL block, the keywords DECLARE, BEGIN, and EXCEPTION are not terminated by a semicolon. However, the keyword END, all SQL statements, and PL/SQL statements must be terminated with a semicolon.

Section	Description	Inclusion
Declarative (DECLARE)	Contains declarations of all variables, constants, cursors, and user-defined exceptions that are referenced in the executable and exception sections	Optional
Executable (BEGIN … END)	Contains SQL statements to retrieve data from the database; contains PL/SQL statements to manipulate data in the block	Mandatory
Exception (EXCEPTION)	Specifies the actions to perform when errors and abnormal conditions arise in the executable section	Optional

Block Types

Procedure

```
PROCEDURE name
IS

BEGIN
  --statements

[EXCEPTION]

END;
```

Function

```
FUNCTION name
RETURN datatype
IS
BEGIN
  --statements
  RETURN value;
[EXCEPTION]

END;
```

Anonymous

```
[DECLARE]


BEGIN
  --statements

[EXCEPTION]

END;
```

SUB PROGRAMAS

Differences Between Anonymous Blocks and Subprograms

Anonymous Blocks	Subprograms
Unnamed PL/SQL blocks	Named PL/SQL blocks
Compiled every time	Compiled only once
Not stored in the database	Stored in the database
Cannot be invoked by other applications	Named and, therefore, can be invoked by other applications
Do not return values	Subprograms called functions must return values.
Cannot take parameters	Can take parameters

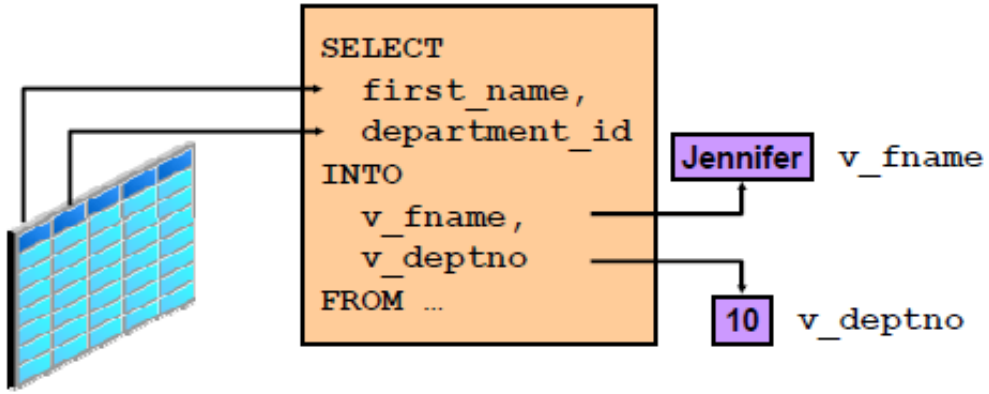
DECLARACIÓN DE VARIABLES PL/SQL

Use of Variables

Variables can be used for:

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

```
Declare
V_fname varchar2(100);
V_deptno number;
begin
```



Declaración de variables PL/SQL

Requirements for Variable Names

A variable name:

- Must start with a letter
- Can include letters or numbers
- Can include special characters (such as $, _, and #)
- Must contain no more than 30 characters
- Must not include reserved words

Handling Variables in PL/SQL

Variables are:

- Declared and initialized in the declarative section (between declare & begin)
- Used and assigned new values in the executable section (between begin & end)
- Passed as parameters to PL/SQL subprograms (like procedure and function)
- Used to hold the output of a PL/SQL subprogram V_sal=get_emp_sal (100)

variable

function

Parameter ex:emp_id

Declaración de variables PL/SQL

Declaring and Initializing PL/SQL Variables

Syntax:

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

Examples:

```
DECLARE
  v_hiredate      DATE;
  v_deptno        NUMBER(2) NOT NULL := 10;
  v_location      VARCHAR2(13) := 'Atlanta';
  c_comm          CONSTANT NUMBER := 1400;
```

In the syntax:

identifier	Is the name of the variable
CONSTANT	Constrains the variable so that its value cannot change (Constants must be initialized.)
data type	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 contains 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

Declaración de variables PL/SQL

Declaring and Initializing PL/SQL Variables

¿Cuál es la diferencia para el caso 1 y para el caso 2?

1

```
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;
/
```

2

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

Declaración de variables PL/SQL

Delimiters in String Literals

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

Resulting output

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

Delimiters in String Literals

If your string contains an apostrophe (identical to a single quotation mark), you must double the quotation mark, as in the following example:

```
v_event  VARCHAR2(15):='Father''s day';
```

The first quotation mark acts as the escape character. This makes your string complicated, especially if you have SQL statements as strings. You can specify any character that is not present in the string as a delimiter. The slide shows how to use the `q'` notation to specify the delimiter. The example uses `!` and `[` as delimiters. Consider the following example:

```
v_event  := q'!Father's day!';
```

You can compare this with the first example on this page. You start the string with `q'` if you want to use a delimiter. The character following the notation is the delimiter used. Enter your string after specifying the delimiter, close the delimiter, and close the notation with a single quotation mark. The following example shows how to use `[` as a delimiter:

```
v_event  := q'[Mother's day]';
```

Guidelines for Declaring and Initializing PL/SQL Variables

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

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

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

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

- Avoid using column names as identifiers.

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

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

Declaración de variables PL/SQL

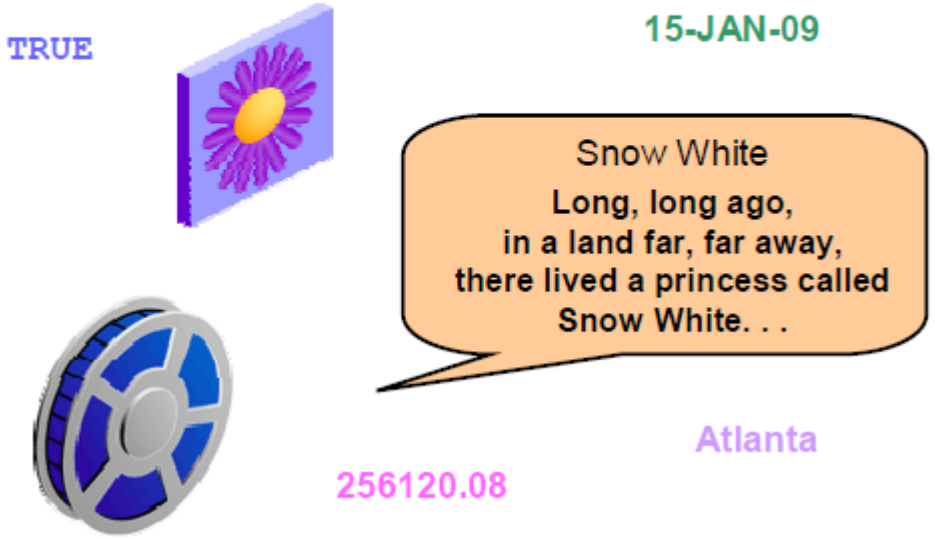
Naming Conventions of PL/SQL

PL/SQL Structure	Convention	Example
Variable	v_*variable_name*	v_rate
Constant	c_*constant_name*	c_rate
Subprogram parameter	p_*parameter_name*	p_id
Bind (host) variable	b_*bind_name*	b_salary
Cursor	cur_*cursor_name*	cur_emp
Record	rec_*record_name*	rec_emp
Type	*type_name_type*	ename_table_type
Exception	e_*exception_name*	e_products_invalid
File handle	f_*file_handle_name*	f_file

Types of Variables

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

Types of Variables



Declaración de variables PL/SQL

Base Scalar Data Types

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

Base Scalar Data Types

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

Cuatro categorías: CHAR, NUMBER, DATE & BOOLEAN

Data type	Category	Default	Range	Notes
Char	Characters	1	Up to 32,767 bytes	Fixed length characters
Varchar2	Characters		Up to 32,767 bytes	Variable character
Number [(precision, scale)]	Number		P from 1 through 38 S from -84 through 127.	
BINARY_INTEGER	Number		integers between -2,147,483,647 and 2,147,483,647	They are same and faster than number
PLS_INTEGER	Number		integers between -2,147,483,647 and 2,147,483,647	
BOOLEAN	BOOLEAN		TRUE, FALSE, NULL	
BINARY_FLOAT	Number		Represents floating-point number in IEEE 754 format. It requires 5 Bytes to store the value.	
BINARY_DOUBLE	Number		Represents floating-point number in IEEE 754 format. It requires 9 Bytes to store the value.	

SCALAR DATA TYPE

Declaración de variables PL/SQL

Data type	range	Notes
DATE	Between 4712 B.C. and A.D. 9999.	It also include hours/ minutes/seconds
TIMESTAMP [(precision)]	Between 4712 B.C. and A.D. 9999.	The TIMESTAMP data type, which extends the DATE data type, stores the year, Month, day, hour, minute, second, and fraction of second. Precision from 1-9 default 6
TIMESTAMP WITH TIME ZONE	Between 4712 B.C. and A.D. 9999.	includes a time-zone
TIMESTAMP WITH LOCAL TIME ZONE	Between 4712 B.C. and A.D. 9999	includes a local time-zone
INTERVAL YEAR TO MONTH		store and Manipulate intervals of years and months. Example 1-2
INTERVAL DAY TO SECOND		store and Manipulate intervals of days, hours, minutes, and seconds. Example: 4 08:12:33

SCALAR DATA TYPE

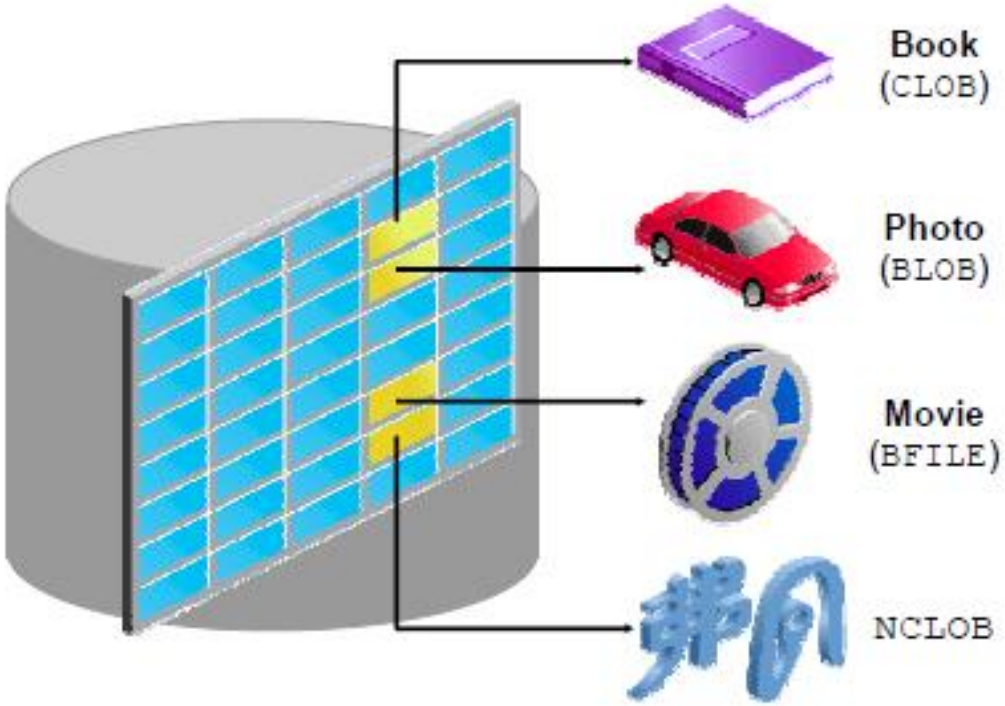
Declaración de variables PL/SQL

Declaring Scalar Variables

Examples:

```
DECLARE
  v_emp_job          VARCHAR2(9);
  v_count_loop       BINARY_INTEGER := 0;
  v_dept_total_sal   NUMBER(9,2) := 0;
  v_orderdate        DATE := SYSDATE + 7;
  c_tax_rate         CONSTANT NUMBER(3,2) := 8.25;
  v_valid            BOOLEAN NOT NULL := TRUE;
  ...
```

LOB Data Type Variables



Composite Data Types: Records and Collections

In a PL/SQL record, the internal components can be of different data types, and are called fields. ES UNA FILA

PL/SQL Record:

TRUE	23-DEC-98	ATLANTA	

PL/SQL Collections:

1	SMITH
2	JONES
3	NANCY
4	TIM

VARCHAR2

PLS_INTEGER

1	5000
2	2345
3	12
4	3456

NUMBER

PLS_INTEGER

In a PL/SQL collection, the internal components are always of the same data type, and are called elements

Lists and arrays are classic examples of collections

Declaración de variables PL/SQL

%TYPE Attribute

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

Advantages of the %TYPE Attribute

- You can avoid errors caused by data type mismatch or wrong precision.
- You can avoid hard coding the data type of a variable.
- You need not change the variable declaration if the column definition changes. If you have already declared some variables for a particular table without using the %TYPE attribute, the PL/SQL block may throw errors if the column for which the variable is 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.

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;
...
```

Declaración de variables PL/SQL

Declaring Boolean Variables

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

Bind Variables

Bind variables are:

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

Values can be output using the `PRINT` command.

* Required when using SQL*Plus and SQL Developer

Script - demo bind variable

Creating Bind Variables

To create a bind variable in SQL Developer, use the `VARIABLE` command. For example, you declare a variable of type `NUMBER` and `VARCHAR2` as follows:

```
VARIABLE return_code NUMBER
VARIABLE return_msg  VARCHAR2(30)
```

Viewing Values in Bind Variables

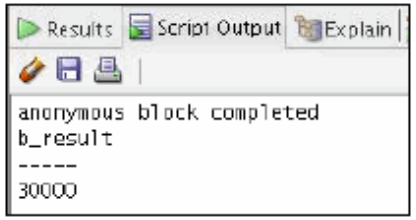
You can reference the bind variable using SQL Developer and view its value using the `PRINT` command.

Example

You can reference a bind variable in a PL/SQL program by preceding the variable with a colon.

For example, the following PL/SQL block creates and uses the bind variable `b_result`. The output resulting from the `PRINT` command is shown below the code.

```
VARIABLE b_result NUMBER
BEGIN
  SELECT (SALARY*12) + NVL(COMMISSION_PCT,0) INTO :b_result
  FROM employees WHERE employee_id = 144;
END;
/
PRINT b_result
```



Declaración de variables PL/SQL

Referencing Bind Variables

Example:

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

Output

Results | Script Output | Explain | Autotrace

```
anonymous block completed
b_emp_salary
----
7000

FIRST_NAME           LAST_NAME
-------------------- -------------------------
Oliver               Tuvault
Sarath               Sewall
Kimberely            Grant

3 rows selected
```

Declaración de variables PL/SQL

USING AUTOPRINT WITH BIND VARIABLES

SQL_Worksheet History

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

Enter Substitution Variable

EMPNO:

178

OK Cancel

Results Script Output Exp

```
anonymous block completed
b_emp_salary
----
7000
```

Declaración de variables PL/SQL

Question 1:

Which of the following is the best answer to describe PL/SQL?

- Pl/sql is Extension to SQL
- You can define variables inside pl/sql block
- We have 2 types of blocks in pl/sql which is subprogram and anonyms block
- All the above

Quiz

Question 2:

The Declare section in PL/SQL is

- Mandatory
- Optional

Quiz

Question 3:

You can only write pl/sql statements inside pl/sql block

- True
- False

Quiz

Question 4:

Which is the correct answer for anonyms block?

- Can be invoked
- It is a named pl/sql block
- Stored in database
- Compiled every time

Quiz

GRACIAS

POR SU PREFERENCIA

GALAXY
TRAINING