DATABASE SYSTEMS PROGRAMMING

PL/SQL

ORACLE®

Dr. Katrina Sundus
Computer Science Department – Al-Zaytoonah University 2024-2025

Learning Objectives

- PL/SQL
- PL/SQL structure
- Define variables
- Assign values

About PL/SQL

Belong to oracle

- PL/SQL is the procedural extension to SQL with design features of programming languages.
- Data manipulation and query statements of SQL are included within procedural units of code.

```
Declare
....
Begin
....
End;
```

+

```
Insert

Update

Delete
```

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

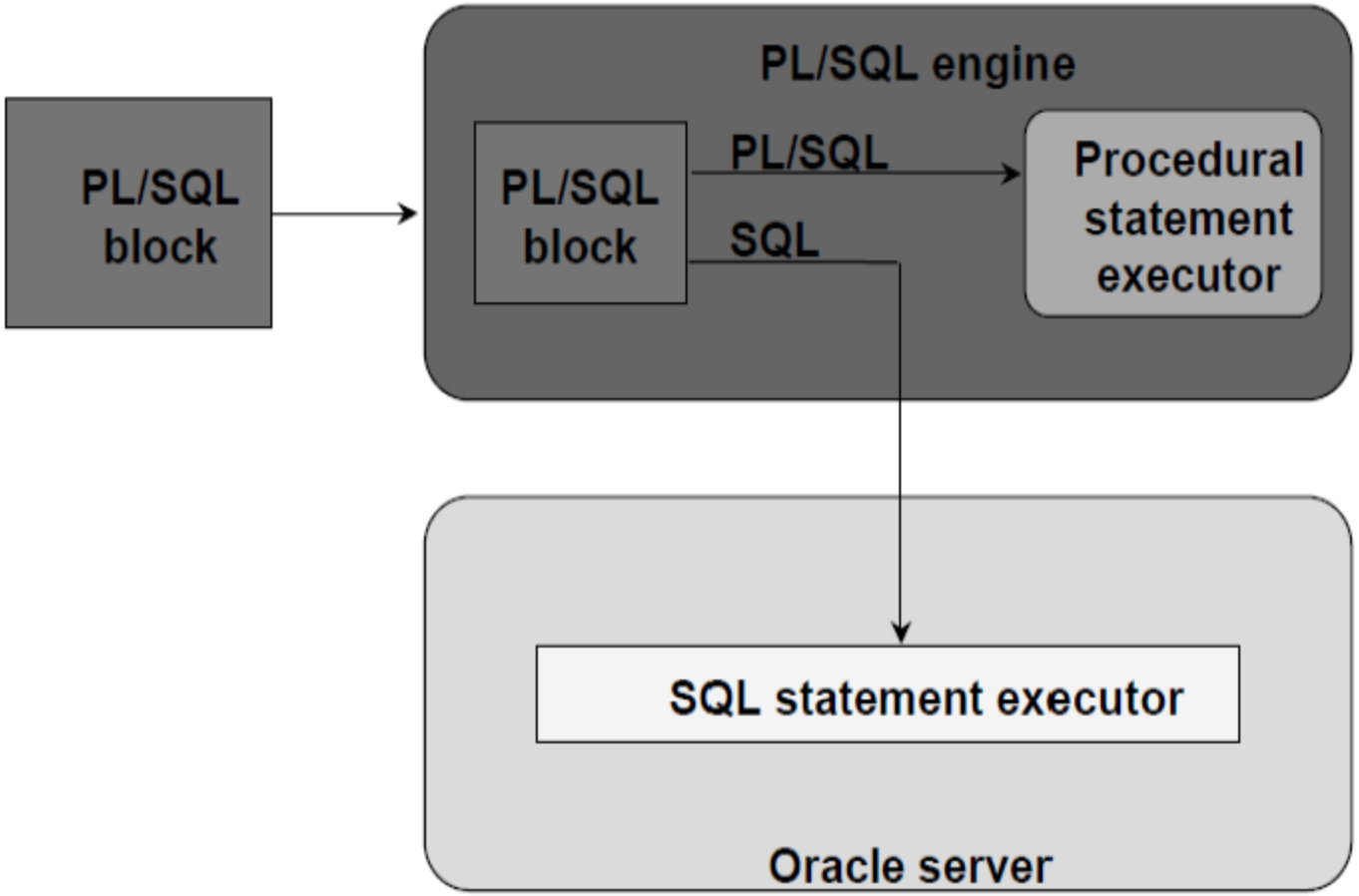
In SQL
You will do multiple SQL statements

```
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

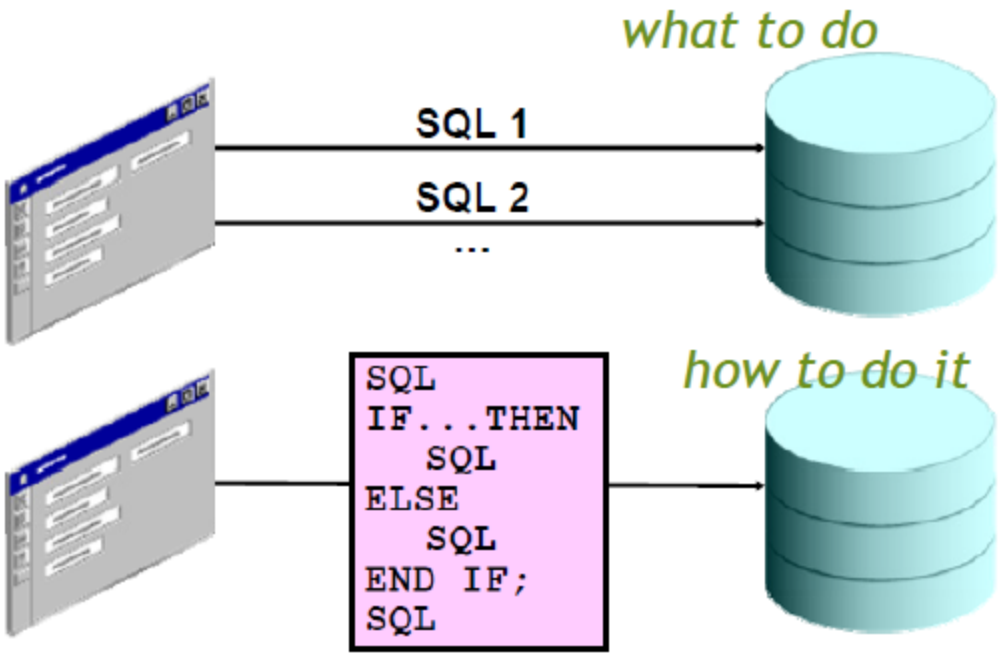
```
Create procedure update_sal
( p_dept_id number , p_amount number )
Is
Begin
……..
…….
…….
End;
```

PL/SQL Environment



Benefits of PL/SQL

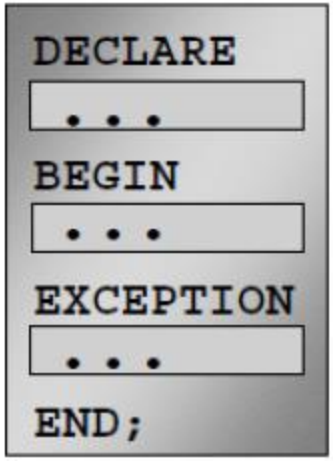
- Integration of procedural constructs with SQL
- Improved performance



- Modularized program development (You can create procedure
- Integration with Oracle tools (oracle forms, oracle reports ...)
- Portability PL/SQL programs can run anywhere an Oracle server runs
- 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 errors occur
END;             – Mandatory
```



Block Types

Subprograms

Anonymous

```
[DECLARE]

BEGIN
   --statements

[EXCEPTION]

END;
```

Procedure

```
PROCEDURE name
IS

BEGIN
   --statements

[EXCEPTION]

END;
```

Function

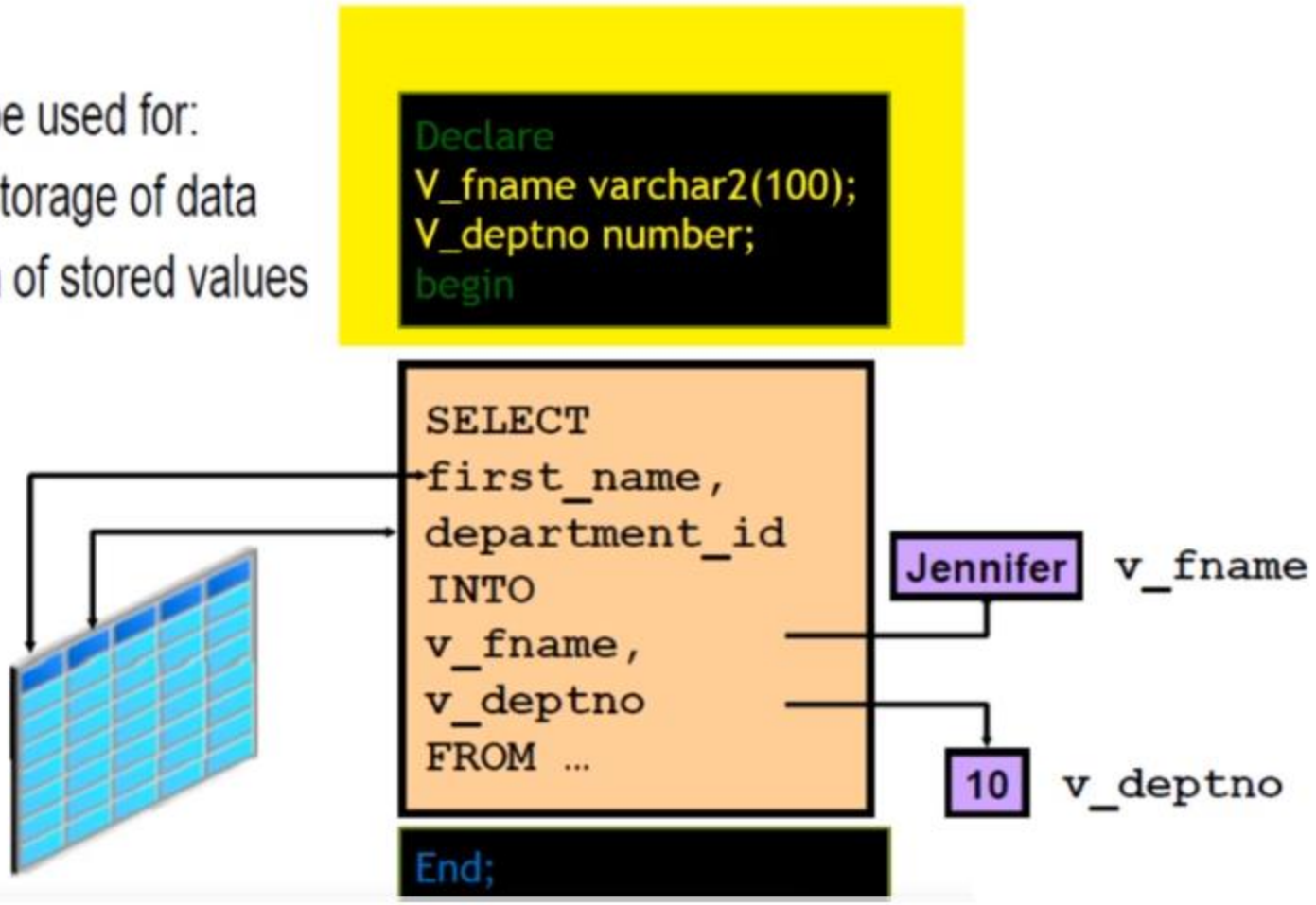
```
FUNCTION name
RETURN datatype
IS
BEGIN
   --statements
   RETURN value;
[EXCEPTION]

END;
```


Functions
calc_area_Square
calc_area_Triangle
calc_area_Rectangle
Package
AREA
AREA.calc_area_Square
AREA.calc_area_Triangle
AREA.calc_area_Rectangle

Use of Variables

Variables can be used for:

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



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

Starting with Oracle Database 12*c* Release 2 (12.2),
the maximum length of identifier names for most types of database objects has been increased to 128 bytes.

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)

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;
```

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.

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 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 notes 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]';
```

Types of Variables

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

Scalar Data Types

- Hold a single value
- Have no internal components

We can arrange it in 4 categories:

- Character
- Date
- Number
- Boolean

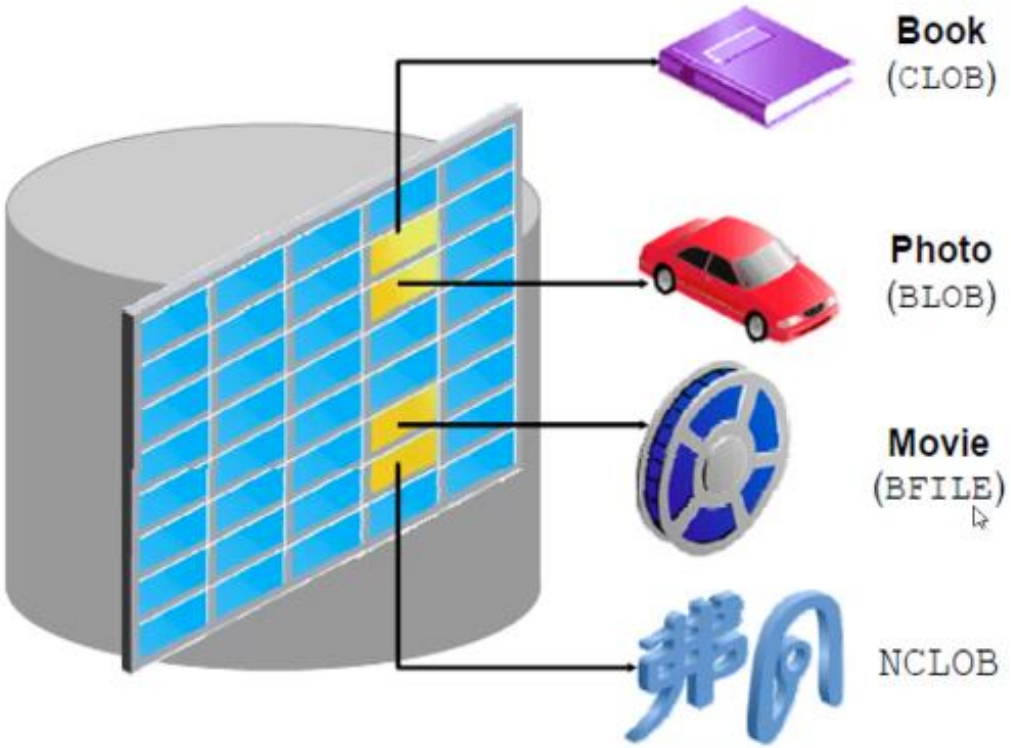
PL/SQL Data types

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.	

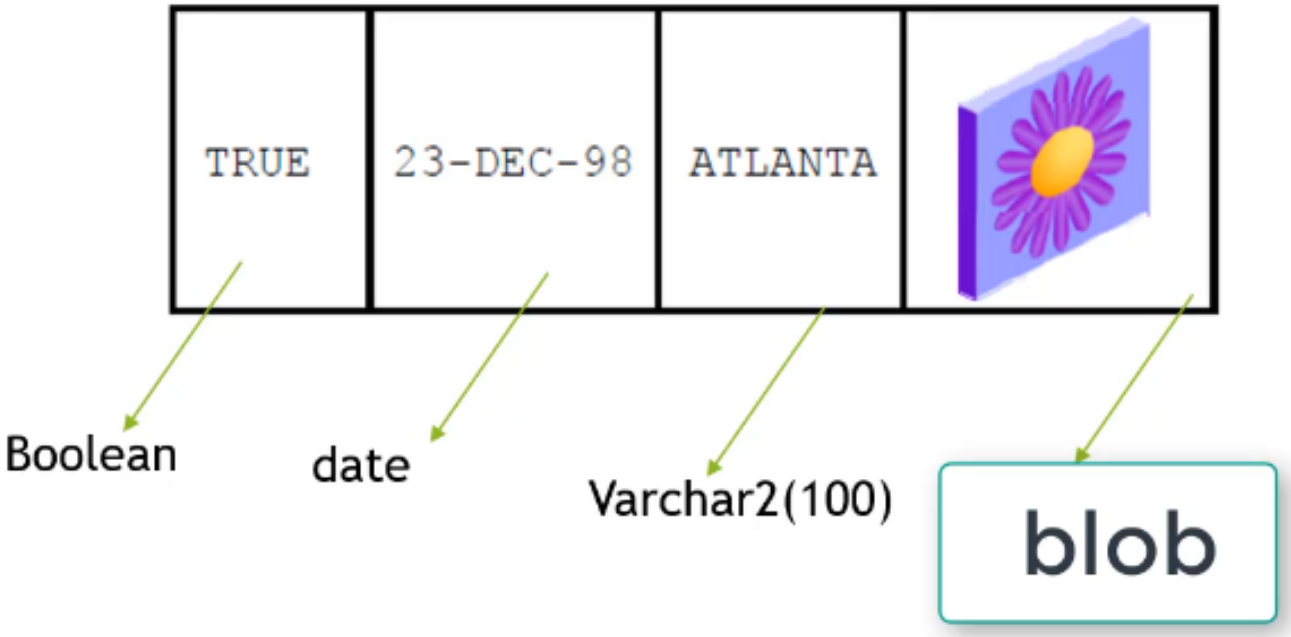
Scalar Date types

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

LOB Data Type Variables



Composite Data Types



%TYPE Attribute

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

```
...
  emp_lname      employees.last_name%TYPE;
...
```

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

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.

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

```
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

b_result
----------
30000
```