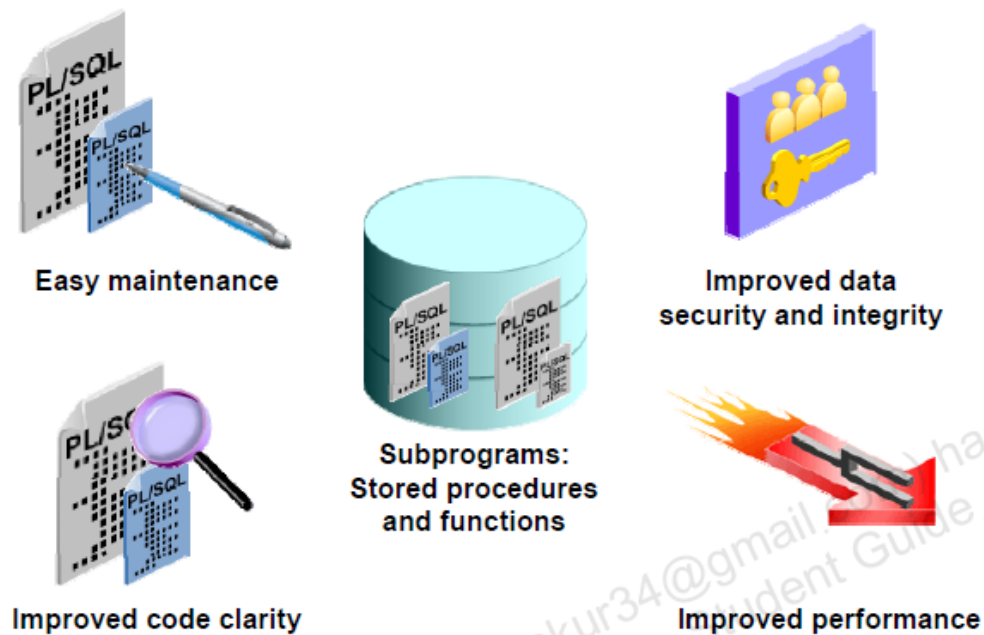


## Creating Procedures:

How it helps us?

1. Easier Maintenance of Code:
  - a. Refactoring → Two types requirement one is static and Second one is dynamic.  
For the Dynamic requirement we need to do refactoring. Changes are reflected to all the files.
2. Security and Integrity of Data: Data should be remained unaltered (It should not be changed at three form of data)
  - a. Three forms of Data:
    - i. IN USE
    - ii. AT REST → Just dumped data → not using
    - iii. IN TRANSIT → Most critically form → Data is sent from Client to Developer → SQL Injection is done in this part of form (Putting FALSE values in columns)

## **The Benefits of Using PL/SQL Subprograms**



- Two Type of Rights in SQL Objects:
  1. Definer's Right → Schema maker → Will have all the rights
  2. Invoker's Right → Have only right to invoke query

- Two Type of Privileges:
  1. System: EXECUTE ANY PRIVILAGE → Critical and need to manage it
  2. Object: something that doesn't have ANY keyword →  
For example,  
CREATE ANY TABLE  
VS CREATE TABLE

NEVER ASSIGN **SYS DBA** PRIVILAGE TO ANYONE.

### PL/SQL Subprogrammes:

Named PL/SQL block that can be called with a set of Parameters

- Two types of PARAMETERS:
  1. FORMAL → Part of Declaration Section
  2. ACTUAL → Used in Executable Section

It can be Procedure or a function.

## 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

Three types of Mode:

1. IN → By default its declared → It provides values for a subprogramme. Getting Values from Calling to Procedure → Actual to Formal Parameters
2. OUT → Sending Parameters from Procedures to Caller → Formal To Actual
3. IN OUT → Takes value in IN and manipulates it and send it OUT

```

7 CREATE OR REPLACE PROCEDURE format_phone (p_phone_no IN OUT VARCHAR2) IS
8 BEGIN
9     p_phone_no := '(' || SUBSTR(p_phone_no,1,3) ||
10                  ' ' || SUBSTR(p_phone_no,4,3) ||
11                  '-' || SUBSTR(p_phone_no,7);
12 END format_phone;
13 /
14

```

Available Notations for Passing Actual Parameters: (After 12c)

1. POSITIONAL
  - a. Ordinal Position should be matched
  - b. List the actual parameters in same order as formal parameters
2. NAMED
  - a. List the actual parameters in arbitrary order and use association operator (=>) to associate a named formal parameter with its actual parameters.
3. MIXED

### Creating Functions:

## Creating Functions

The PL/SQL block must have at least one RETURN statement.

```

CREATE [OR REPLACE] FUNCTION function_name
[(parameter1 [mode1] datatype1, . . .)]
RETURN datatype IS|AS
[local_variable_declarations;
. . .]
BEGIN
-- actions;
RETURN expression;
END [function_name];

```

PL/SQL Block

## The Difference Between Procedures and Functions

Procedures	Functions
Execute as a PL/SQL statement	Invoke as part of an expression
Do not contain RETURN clause in the header	Must contain a RETURN clause in the header
Can pass values (if any) using output parameters	Must return a single value
Can contain a RETURN statement without a value	Must contain at least one RETURN statement

## Naming Conventions of PL/SQL Structures Used in This Course

PL/SQL Structure	Convention	Example
Variable	<code>v_variable_name</code>	<code>v_rate</code>
Constant	<code>c_constant_name</code>	<code>c_rate</code>
Subprogram parameter	<code>p_parameter_name</code>	<code>p_id</code>
Bind (host) variable	<code>b_bind_name</code>	<code>b_salary</code>
Cursor	<code>cur_cursor_name</code>	<code>cur_emp</code>
Record	<code>rec_record_name</code>	<code>rec_emp</code>
Type	<code>type_name_type</code>	<code>ename_table_type</code>
Exception	<code>e_exception_name</code>	<code>e_products_invalid</code>
File handle	<code>f_file_handle_name</code>	<code>f_file</code>

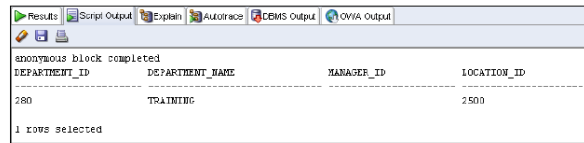
---

## Comparing the Parameter Modes

IN	OUT	IN OUT
Default mode	Must be specified	Must be specified
Value is passed into subprogram	Value is returned to the calling environment	Value passed into subprogram; value returned to calling environment
Formal parameter acts as a constant	Uninitialized variable	Initialized variable
Actual parameter can be a literal, expression, constant, or initialized variable	Must be a variable	Must be a variable
Can be assigned a default value	Cannot be assigned a default value	Cannot be assigned a default value

- When calling a subprogram, you can write the actual parameters using the following notations:
  - **Positional:** Lists the actual parameters in the same order as the formal parameters
  - **Named:** Lists the actual parameters in arbitrary order and uses the association operator (`=>`) to associate a named formal parameter with its actual parameter
  - **Mixed:** Lists some of the actual parameters as positional and some as named

```
-- Passing parameters using the positional notation.  
EXECUTE add_dept ('TRAINING', 2500)
```



The screenshot shows a database interface with a tabbed menu at the top containing 'Results', 'Script Output', 'Explain', 'Autotrace', 'DBMS Output', and 'OVIA Output'. The 'Results' tab is active, displaying the message 'anonymous block completed'. Below this is a table with four columns: DEPARTMENT\_ID, DEPARTMENT\_NAME, MANAGER\_ID, and LOCATION\_ID. The table contains one row with the values 280, TRAINING, and 2500. At the bottom, it states '1 rows selected'.

DEPARTMENT_ID	DEPARTMENT_NAME	MANAGER_ID	LOCATION_ID
280	TRAINING		2500

1 rows selected

```
-- Passing parameters using the named notation.  
EXECUTE add_dept (p_loc=>2400, p_name=>'EDUCATION')
```