Creating Procedure

Block Types

Subprograms

Anonymous

```
[DECLARE]


BEGIN
  --statements

[EXCEPTION]

END;
```

Procedure

```
PROCEDURE name
IS

BEGIN
  --statements

[EXCEPTION]

END;
```

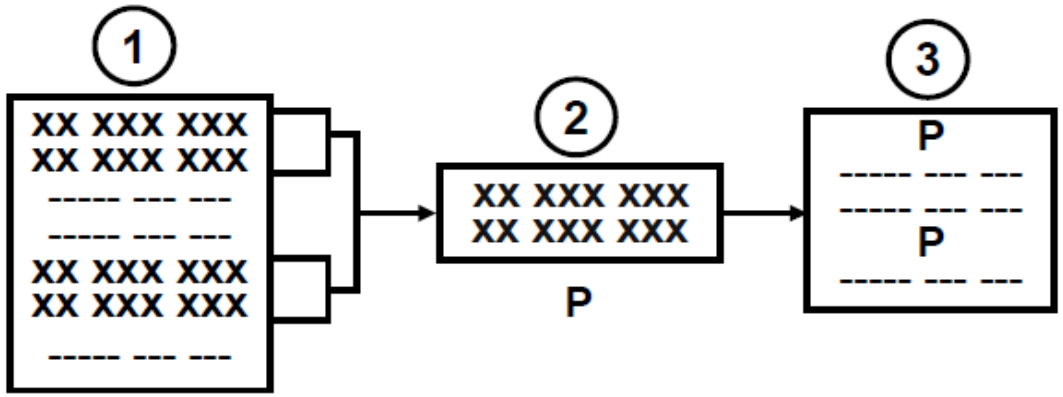
Function

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

END;
```

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

Creating a Modularized Subprogram Design




```
update emp
set sal=sal+ 100
where emp_id =100;

update emp
set sal=sal+ 100
where emp_id =101;

update emp
set sal=sal+ 100
where emp_id =102;
```

```
procedure update_emp
(p_emp_id number, p_am number)
is
begin
 update emp
set sal=sal+ p_am
where emp_id=p_emp_id;
commit;
end;
```

```
procedure update_emp_group
is
...
....
update_emp( x , y);
...
....

....
end;
```

Modularize code into subprograms.

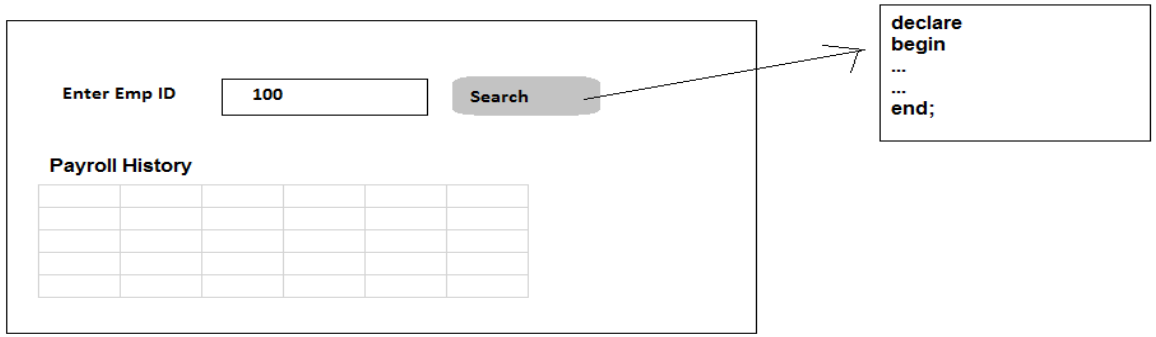
1. Locate code sequences repeated more than once.
2. Create subprogram P containing the repeated code
3. Modify original code to invoke the new subprogram.

Modularizing Development with PL/SQL Blocks

- PL/SQL is a block-structured language. The PL/SQL code block helps modularize code by using:
  - Anonymous blocks
  - Procedures and functions
  - Packages
  - Database triggers
- The benefits of using modular program constructs are:
  - Easy maintenance
  - Improved data security and integrity
  - Improved performance
  - Improved code clarity

So what is the benefits of anonymous block ?

- Writing trigger code for oracle forms components




```
Begin
P1;
P2;
End;
```

- Initiate calls for procedures , functions and packages

- Isolating exception handling within a block of code

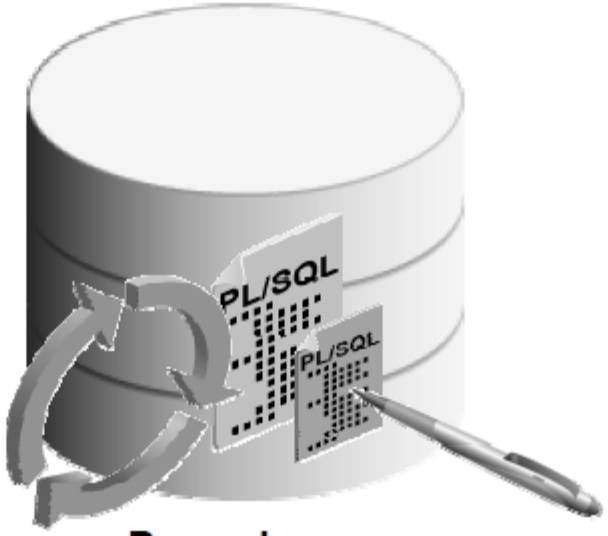
```
Procedure p1
Is
Begin
    begin
     ........
    Exception
    End;

End;
```

What Are PL/SQL Subprograms?

- A PL/SQL subprogram is a named PL/SQL block that can be called with a set of parameters.
- You can declare and define a subprogram within either a PL/SQL block or another subprogram.
- A subprogram consists of a specification and a body.
- A subprogram can be a procedure or a function.
- Typically, you use a procedure to perform an action and a function to compute and return a value.

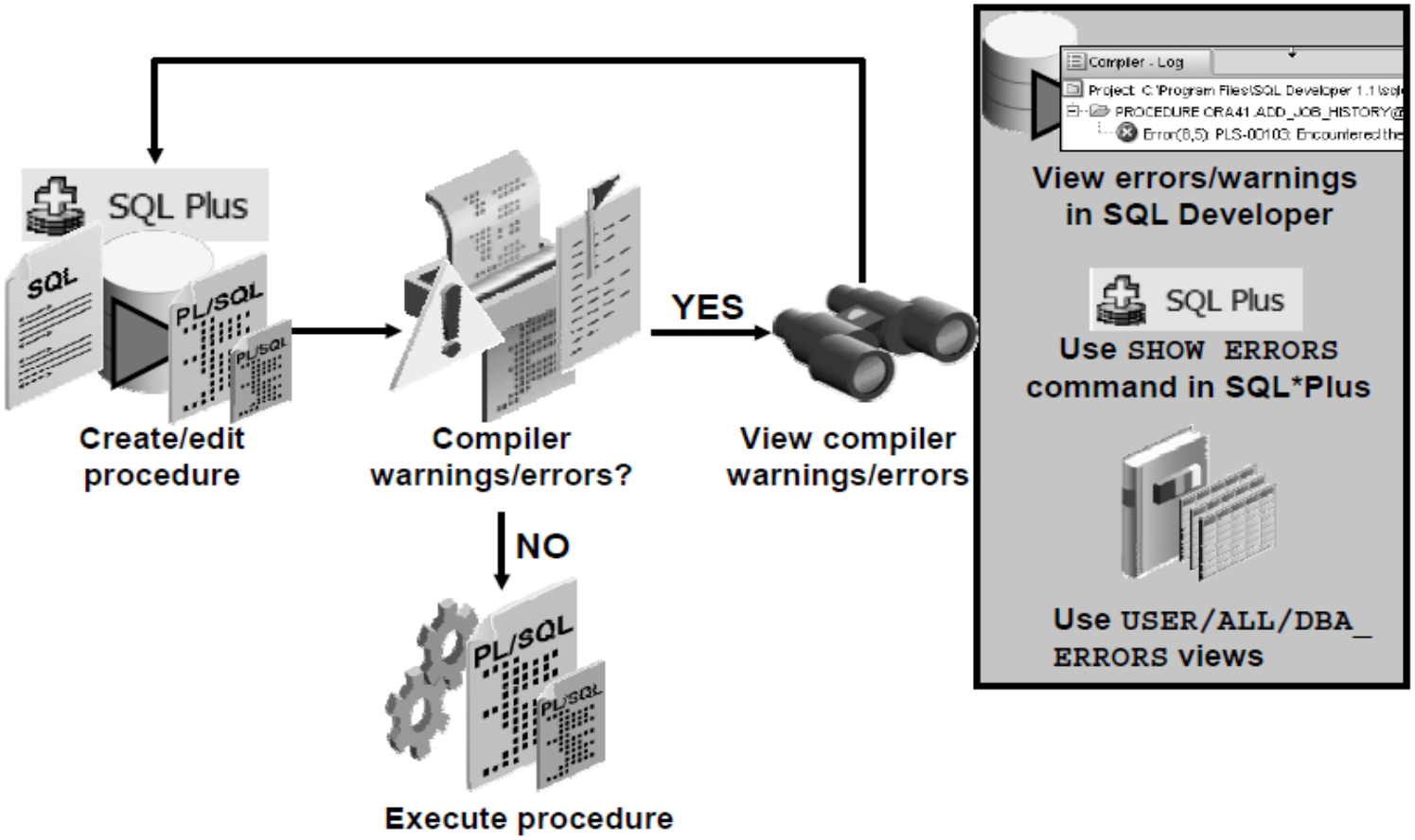
What Are Procedures?

- Are a type of subprogram that perform an action
- Can be stored in the database as a schema object
- Promote reusability and maintainability



Procedures

Creating Procedures: Overview



Create procedure statement

hr for pdborcl

0.123 seconds

Worksheet | Query Builder

```
CREATE OR REPLACE PROCEDURE UPDATE_SAL
(P_EMP_ID IN NUMBER, P_AMOUNT IN NUMBER)
IS

BEGIN

  UPDATE employees
  set salary=salary+P_AMOUNT
  where employee_id=P_EMP_ID;
  Commit;
exception

WHEN OTHERS THEN

DBMS_OUTPUT.PUT_LINE (SQLCODE);
DBMS_OUTPUT.PUT_LINE (SQLERRM);

END;
```

Optional to overwide existing procedure

Optional parameters list

Here you declare variables

Parameter mode

Or
AS

- Parameters data types without size
- You should have create procedure privilege
- Substitution and host variables not allowed in procedures

1- you can call the procedure alone by this command

```
execute UPDATE_SAL (100,50);
```

2- you can call the procedure inside any PL/SQL block

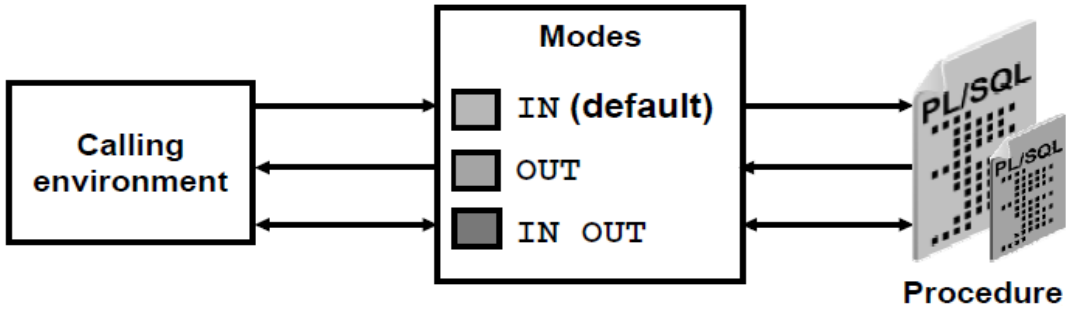
```
Begin
…
UPDATE_SAL (100,50);
……
End;
```

parameter-passing mode:

- An IN parameter mode (the default) provides values for a subprogram to process
- An OUT parameter mode returns a value to the caller
- An IN OUT parameter mode supplies an input value, which may be returned (output) as a modified value

- Parameter modes are specified in the formal parameter declaration, after the parameter name and before its data type.
- The IN mode is the default if no mode is specified.

```
CREATE PROCEDURE proc_name(param_name [mode] datatype)
...
```



Comparing the Parameter Modes

IN	OUT	IN OUT
Default mode	Must be specified	Must be specified
Value is passed into subprogram	Returned to calling environment	Passed into subprogram; 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

Available Notations for Passing Actual Parameters

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

Using the DEFAULT Option for the Parameters

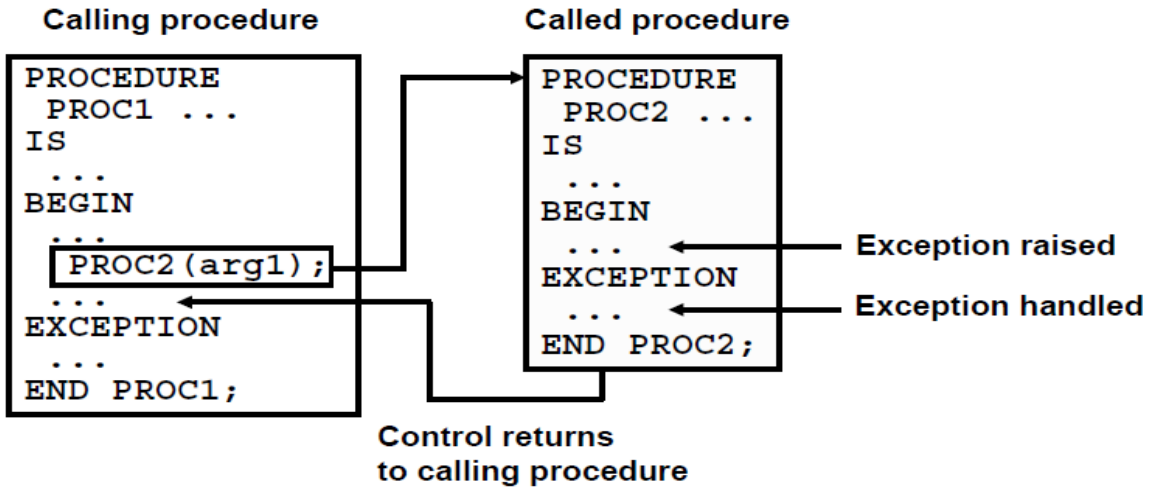
- Defines default values for parameters.
- Provides flexibility by combining the positional and named parameter-passing syntax.

```
CREATE OR REPLACE PROCEDURE add_dept(
 p_name departments.department_name%TYPE:='Unknown',
 p_loc  departments.location_id%TYPE DEFAULT 1700)
IS
BEGIN
  INSERT INTO departments (department_id,
     department_name, location_id)
  VALUES (departments_seq.NEXTVAL, p_name, p_loc);
END add_dept;
```

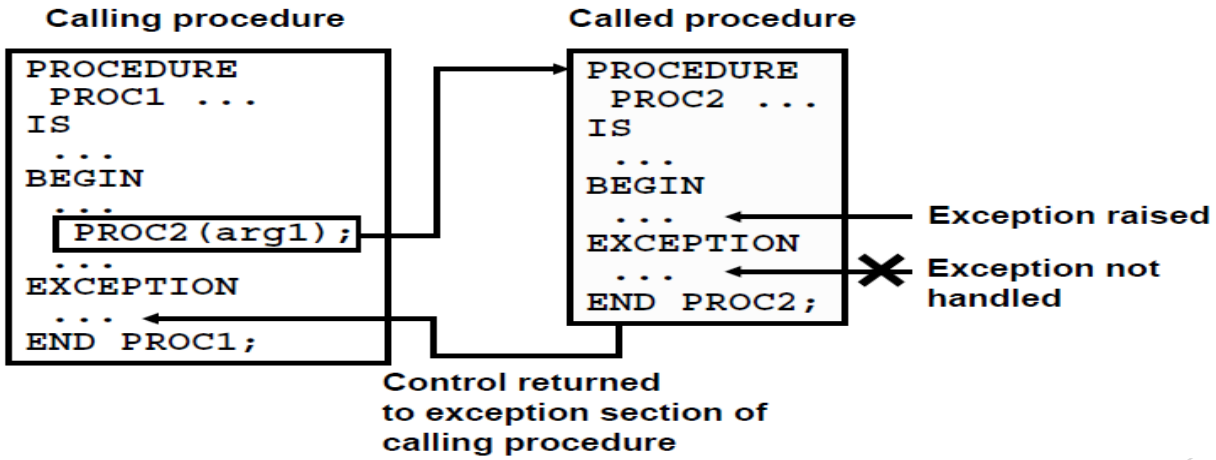
```
EXECUTE add_dept
EXECUTE add_dept ('ADVERTISING', p_loc => 1200)
EXECUTE add_dept (p_loc => 1200)
```

Note: Default value used only for IN parameters

Handled Exceptions



Exceptions Not Handled



Thank You