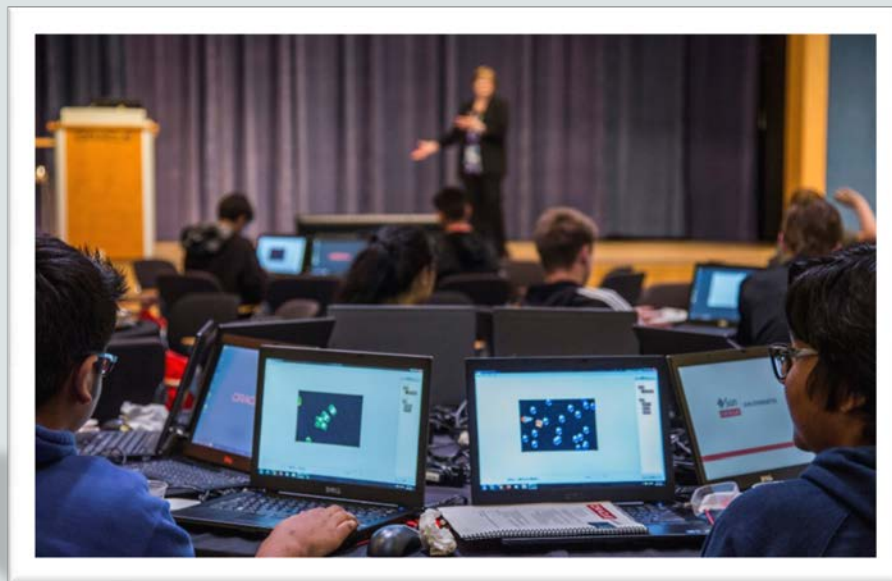




Database Programming with PL/SQL

8-1 Creating Procedures



Objectives

This lesson covers the following objectives:

- Differentiate between anonymous blocks and subprograms
- Identify the benefits of subprograms
- Define a stored procedure
- Create a procedure
- Describe how a stored procedure is invoked
- List the development steps for creating a procedure
- Create a nested subprogram in the declarative section of a procedure

Purpose

- There are times that you want to give a set of steps a name.
- For example, if you're told to take notes, you know that this means you need to get out a piece of paper and a pencil and prepare to write.
- So far you have learned to write and execute anonymous PL/SQL blocks (blocks that do not have a name associated with them).

Purpose

- Next you will learn how to create, execute, and manage two types of PL/SQL subprograms that are named and stored in the database, resulting in several benefits such as shareability, better security, and faster performance.
- Two types of subprograms:
 - Functions
 - Procedures

Differences Between Anonymous Blocks and Subprograms

- As the word “anonymous” indicates, anonymous blocks are unnamed executable PL/SQL blocks.
- Because they are unnamed, they can neither be reused nor stored in the database for later use.
- While you can store anonymous blocks on your PC, the database is not aware of them, so no one else can share them.
- Procedures and functions are PL/SQL blocks that are named, and they are also known as subprograms.

Differences Between Anonymous Blocks and Subprograms

- These subprograms are compiled and stored in the database.
- The block structure of the subprograms is similar to the structure of anonymous blocks.
- While subprograms can be explicitly shared, the default is to make them private to the owner's schema.
- Later subprograms become the building blocks of packages and triggers.

Differences Between Anonymous Blocks and Subprograms

- Anonymous blocks

```
DECLARE (Optional)
    Variables, cursors, etc.;
BEGIN (Mandatory)
    SQL and PL/SQL statements;
EXCEPTION (Optional)
    WHEN exception-handling actions;
END; (Mandatory)
```

- Subprograms (procedures)

```
CREATE [OR REPLACE] PROCEDURE name [parameters] IS|AS (Mandatory)
    Variables, cursors, etc.; (Optional)
BEGIN (Mandatory)
    SQL and PL/SQL statements;
EXCEPTION (Optional)
    WHEN exception-handling actions;
END [name]; (Mandatory)
```


Differences Between Anonymous Blocks and Subprograms

| Anonymous Blocks | Subprograms |
|--|--|
| Unnamed PL/SQL blocks | Named PL/SQL blocks |
| Compiled on every execution | Compiled only once, when created |
| Not stored in the database | Stored in the database |
| Cannot be invoked by other applications | They are 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 |

Benefits of Subprograms

- Procedures and functions have many benefits due to the modularizing of the code:
 - Easy maintenance: Modifications need only be done once to improve multiple applications and minimize testing.
 - Code reuse: Subprograms are located in one place.
- When compiled and validated, they can be used and reused in any number of applications.



Benefits of Subprograms

- Improved data security: Indirect access to database objects is permitted by the granting of security privileges on the subprograms.
- By default, subprograms run with the privileges of the subprogram owner, not the privileges of the user.
- Data integrity: Related actions can be grouped into a block and are performed together (“Statement Processed”) or not at all.



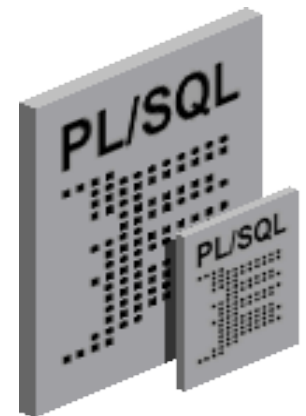
Benefits of Subprograms

- Improved performance: You can reuse compiled PL/SQL code that is stored in the shared SQL area cache of the server.
- Subsequent calls to the subprogram avoid compiling the code again.
- Also, many users can share a single copy of the subprogram code in memory.
- Improved code clarity: By using appropriate names and conventions to describe the action of the routines, you can reduce the need for comments, and enhance the clarity of the code.

Procedures and Functions

Procedures and functions:

- Are named PL/SQL blocks
- Are called PL/SQL subprograms
- Have block structures similar to anonymous blocks:
 - Optional parameters
 - Optional declarative section (but the `DECLARE` keyword changes to `IS` or `AS`)
 - Mandatory executable section
 - Optional section to handle exceptions
- This section focuses on procedures.



What Is a Procedure?

- A procedure is a named PL/SQL block that can accept parameters.
- Generally, you use a procedure to perform an action (sometimes called a “side-effect”).
- A procedure is compiled and stored in the database as a schema object.
 - Shows up in `USER_OBJECTS` as an object type of `PROCEDURE`
 - More details in `USER_PROCEDURES`
 - Detailed PL/SQL code in `USER_SOURCE`

Syntax for Creating Procedures

- Parameters are optional
- Mode defaults to IN
- Datatype can be either explicit (for example, VARCHAR2) or implicit with %TYPE
- Body is the same as an anonymous block

```
CREATE [OR REPLACE] PROCEDURE procedure_name
  [(parameter1 [mode1] datatype1,
    parameter2 [mode2] datatype2,
    . . .)]
IS|AS
procedure_body;
```

Syntax for Creating Procedures

- Use `CREATE PROCEDURE` followed by the name, optional parameters, and keyword `IS` or `AS`.
- Add the `OR REPLACE` option to overwrite an existing procedure.
- Write a PL/SQL block containing local variables, a `BEGIN`, and an `END` (or `END procedure_name`).

```
CREATE [OR REPLACE] PROCEDURE procedure_name
  [(parameter1 [mode] datatype1,
    parameter2 [mode] datatype2, ...)]
IS|AS
  [local_variable_declarations; ...]
BEGIN
  -- actions;
END [procedure_name];
```



PL/SQL Block

Procedure: Example

- In the following example, the `add_dept` procedure inserts a new department with the `department_id` 280 and `department_name` `ST-Curriculum`.
- The procedure declares two variables, `v_dept_id` and `v_dept_name`, in the declarative section.

```
CREATE OR REPLACE PROCEDURE add_dept IS
  v_dept_id      dept.department_id%TYPE;
  v_dept_name    dept.department_name%TYPE;
BEGIN
  v_dept_id      := 280;
  v_dept_name    := 'ST-Curriculum';
  INSERT INTO dept(department_id, department_name)
    VALUES(v_dept_id, v_dept_name);
  DBMS_OUTPUT.PUT_LINE('Inserted ' || SQL%ROWCOUNT || ' row.');
```

```
END;
```

Procedure: Example

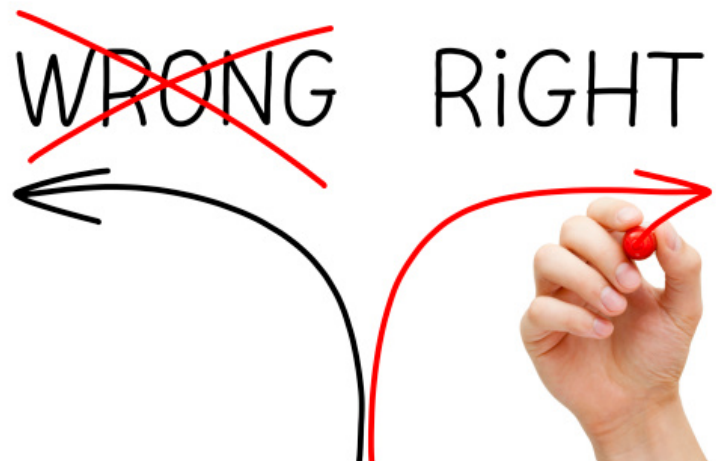
- The declarative section of a procedure starts immediately after the procedure declaration and does not begin with the keyword `DECLARE`.
- This procedure uses the `SQL%ROWCOUNT` cursor attribute to check if the row was successfully inserted. `SQL%ROWCOUNT` should return 1 in this case.

```
CREATE OR REPLACE PROCEDURE add_dept IS
  v_dept_id      dept.department_id%TYPE;
  v_dept_name    dept.department_name%TYPE;
BEGIN
  v_dept_id      := 280;
  v_dept_name    := 'ST-Curriculum';
  INSERT INTO dept(department_id, department_name)
    VALUES(v_dept_id, v_dept_name);
  DBMS_OUTPUT.PUT_LINE('Inserted ' || SQL%ROWCOUNT || ' row.');
```

```
END;
```

Invoking Procedures

- You can invoke (execute) a procedure from:
 - An anonymous block
 - Another procedure
 - A calling application
- Note: You cannot invoke a procedure from inside a SQL statement such as `SELECT`.



Invoking the Procedure from Application Express

- To invoke (execute) a procedure in Oracle Application Express, write and run a small anonymous block that invokes the procedure.
- For example:

```
BEGIN
  add_dept;
END;

SELECT department_id, department_name FROM dept WHERE department_id=280;
```

- The select statement at the end confirms that the row was successfully inserted.

Correcting Errors in CREATE PROCEDURE Statements

- If compilation errors exist, Application Express displays them in the output portion of the SQL Commands window.
- You must edit the source code to make corrections.
- The procedure is still created even though it contains errors.



Correcting Errors in CREATE PROCEDURE Statements

- After you have corrected the error in the code, you need to recreate the procedure.
- There are two ways to do this:
 - Use a CREATE OR REPLACE PROCEDURE statement to overwrite the existing code (most common).
 - DROP the procedure first and then execute the CREATE PROCEDURE statement (less common).



Saving Your Work

Once a procedure has been created successfully, you should save its definition in case you need to modify the code later.

```
CREATE OR REPLACE PROCEDURE add_dept IS
  v_dept_id    dept.department_id%TYPE;
  v_dept_name  dept.department_name%TYPE;
BEGIN
  v_dept_id :=280;
  v_dept_name := 'ST-Curriculum';
  INSERT INTO dept(department_id, department_name)
    VALUES(v_dept_id, v_dept_name);
  DBMS_OUTPUT.PUT_LINE('Inserted ' || SQL%ROWCOUNT || ' row');
END;
```

Results

Explain

Describe

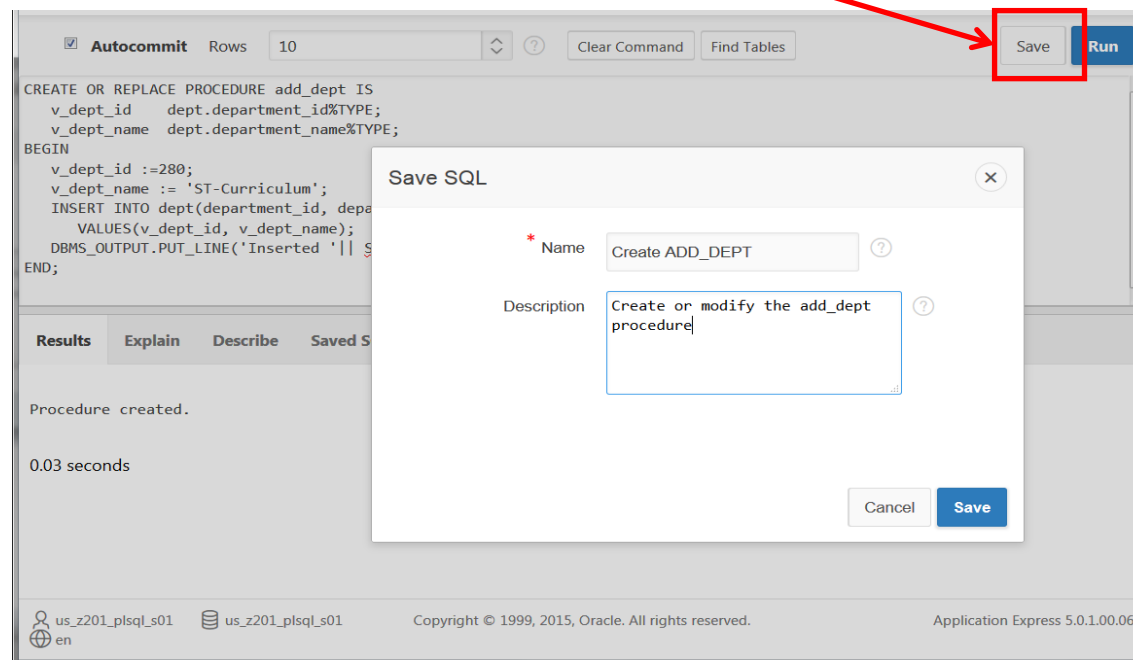
Saved SQL

History

Procedure created.

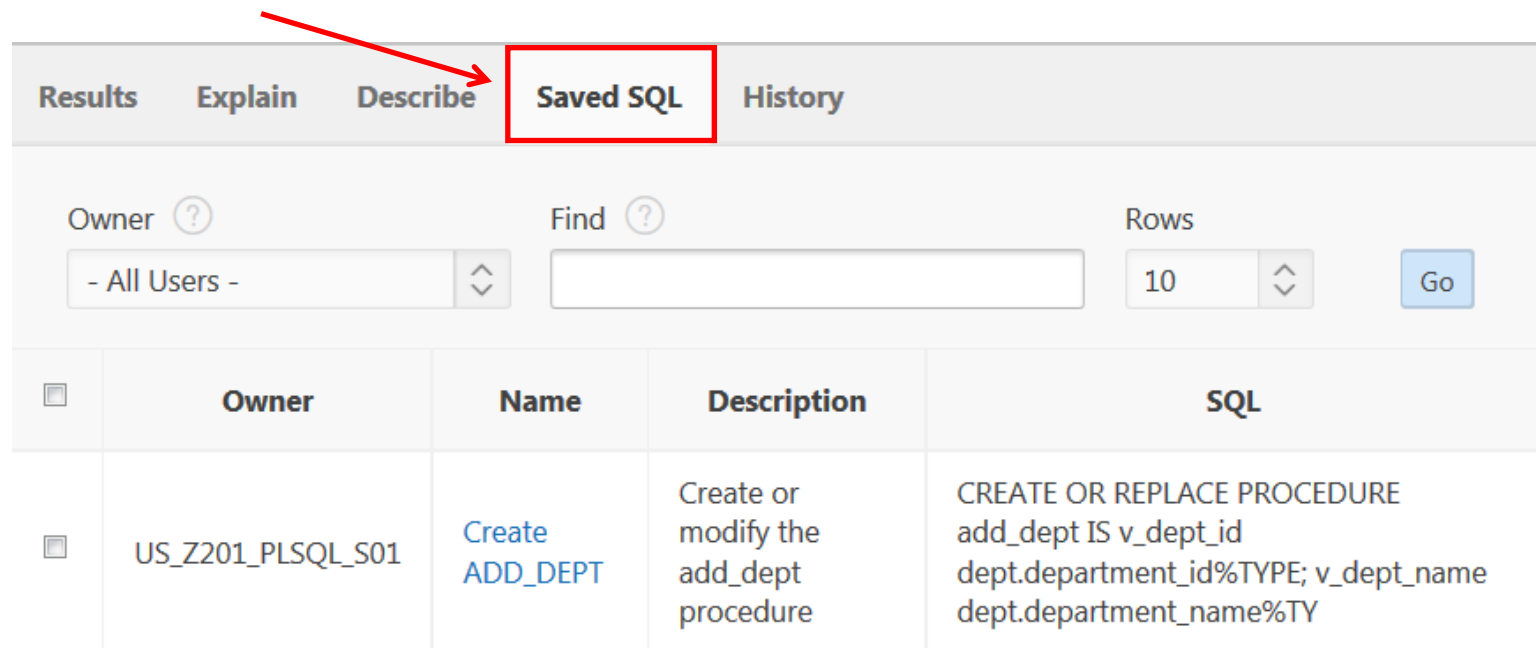
Saving Your Work

In the Application Express SQL Commands window, click the **SAVE** button, then enter a name and optional description for your code.



Saving Your Work

You can view and reload your code later by clicking on the Saved SQL button in the SQL Commands window.



The screenshot shows the SQL Commands window interface. At the top, there are five tabs: Results, Explain, Describe, Saved SQL, and History. The 'Saved SQL' tab is highlighted with a red box, and a red arrow points to it from the text above. Below the tabs, there are search filters: 'Owner' (set to '- All Users -'), 'Find' (empty), and 'Rows' (set to 10). A 'Go' button is located to the right of the 'Rows' filter. Below the filters is a table with the following columns: Owner, Name, Description, and SQL.

| <input type="checkbox"/> | Owner | Name | Description | SQL |
|--------------------------|-------------------|------------------------------------|---|--|
| <input type="checkbox"/> | US_Z201_PLSQL_S01 | Create ADD_DEPT | Create or modify the add_dept procedure | CREATE OR REPLACE PROCEDURE add_dept IS v_dept_id dept.department_id%TYPE; v_dept_name dept.department_name%TY |

Local Subprograms

When one procedure invokes another procedure, we would normally create them separately, but we can create them together as a single procedure if we like.

```
CREATE OR REPLACE PROCEDURE subproc
...
END subproc;
```

```
CREATE OR REPLACE PROCEDURE mainproc
...
IS BEGIN
...
    subproc(...);
...
END mainproc;
```

Local Subprograms

- All the code is now in one place, and is easier to read and maintain.
- The nested subprogram's scope is limited to the procedure within which it is defined; SUBPROC can be invoked from MAINPROC, but from nowhere else.

```
CREATE OR REPLACE PROCEDURE mainproc
  ...
IS
  PROCEDURE subproc (...) IS BEGIN
    ...
  END subproc;
BEGIN
  ...
  subproc(...);
  ...
END mainproc;
```

Local Subprograms

- Every time an employee is deleted, we need to insert a row into a logging table.
- The nested procedure LOG_EMP is called a Local Subprogram.

```
CREATE OR REPLACE PROCEDURE delete_emp
  (p_emp_id IN employees.employee_id%TYPE)
IS
  PROCEDURE log_emp (p_emp IN employees.employee_id%TYPE)
  IS BEGIN
    INSERT INTO logging_table VALUES(p_emp, ...);
  END log_emp;
BEGIN
  DELETE FROM employees
    WHERE employee_id = p_emp_id;
  log_emp(p_emp_id);
END delete_emp;
```

Alternative Tools for Developing Procedures

- If you end up writing PL/SQL procedures for a living, there are other free tools that can make this process easier.
- For instance, Oracle tools, such as SQL Developer and JDeveloper assist you by:
 - Color-coding **commands** vs **variables** vs **constants**
 - Highlighting matched and mismatched (parentheses)
 - Displaying errors more graphically

Alternative Tools for Developing Procedures

- Enhancing code with standard indentations and capitalization
- Completing commands when typing
- Completing column names from tables

Terminology

Key terms used in this lesson included:

- Anonymous blocks
- IS or AS
- Procedures
- Subprograms

Summary

In this lesson, you should have learned how to:

- Differentiate between anonymous blocks and subprograms
- Identify the benefits of subprograms
- Define a stored procedure
- Create a procedure
- Describe how a stored procedure is invoked
- List the development steps for creating a procedure
- Create a nested subprogram in the declarative section of a procedure

