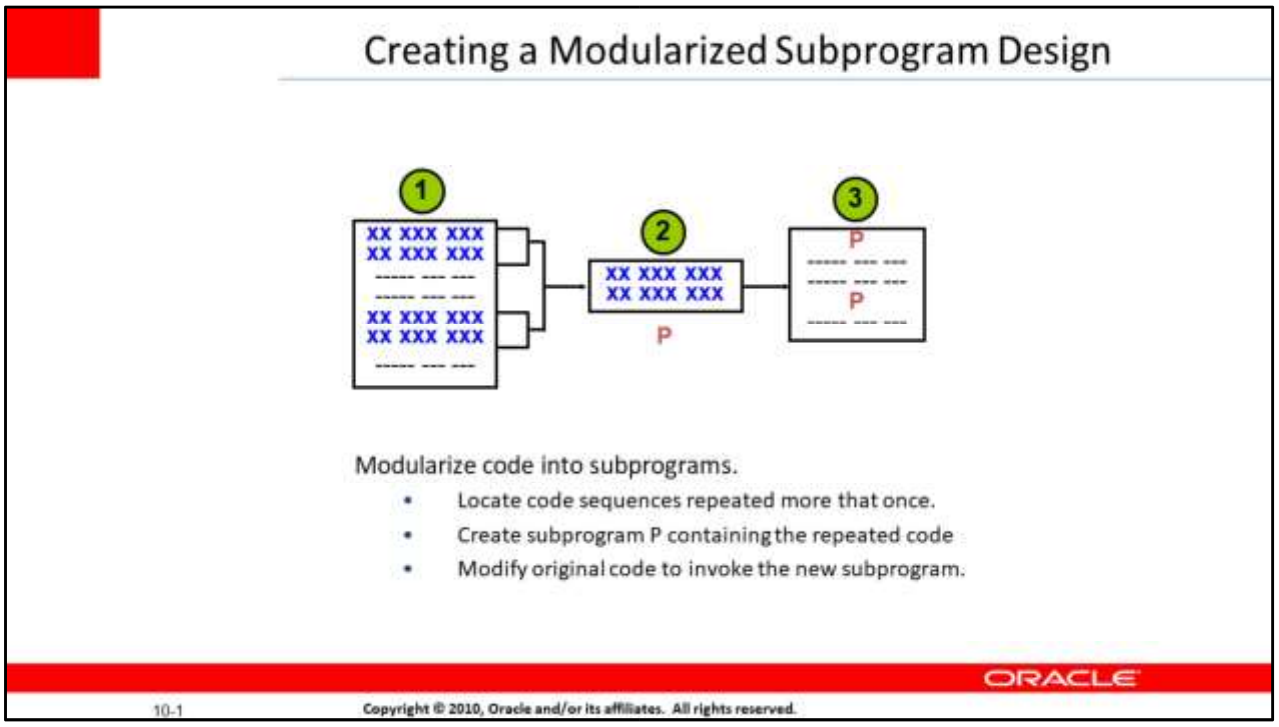


Creating a Modularized and Layered Subprogram Design

The diagram illustrates the principle of modularization with subprograms: the creation of smaller manageable pieces of flexible and reusable code. Flexibility is achieved by using subprograms with parameters, which in turn makes the same code reusable for different input values. To modularize existing code, perform the following steps:

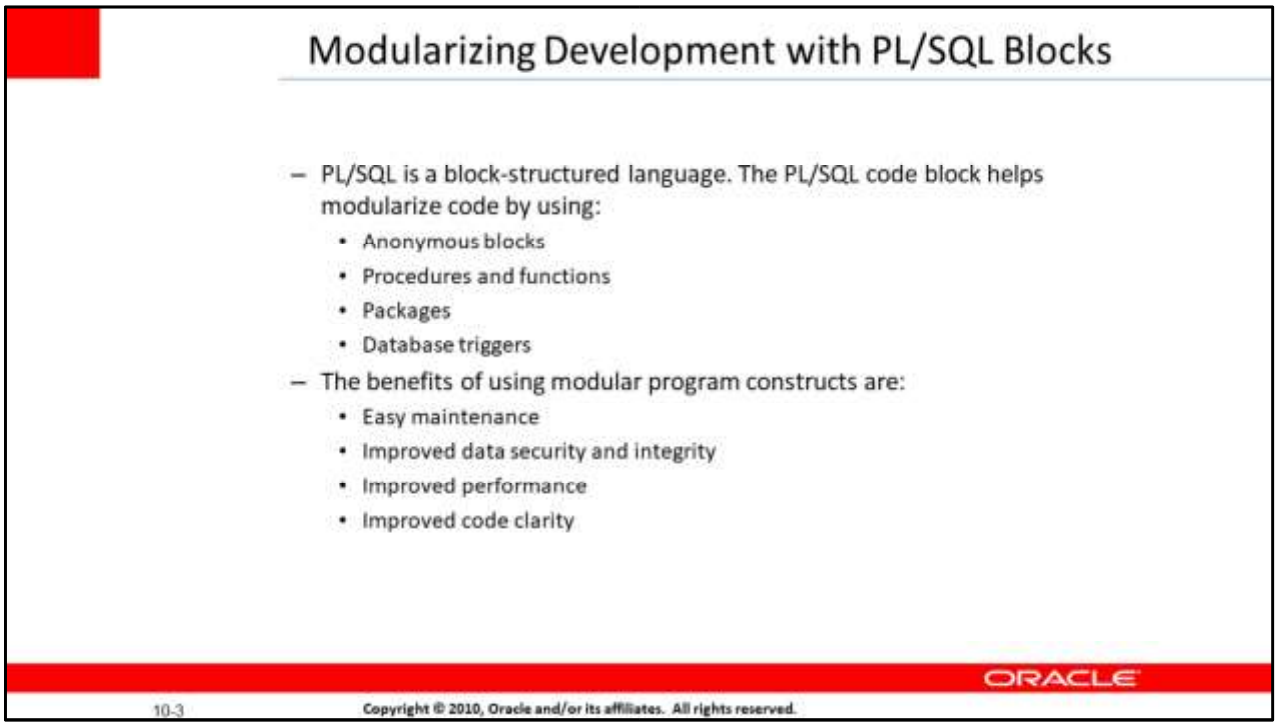
1. Locate and identify repetitive sequences of code.
2. Move the repetitive code into a PL/SQL subprogram. 3. Replace the original repetitive code with calls to the new PL/SQL subprogram.

Following this modular and layered approach can help you create code that is easier to maintain, particularly when the business rules change. In addition, keeping the SQL logic simple and free of complex business logic can benefit from the work of Oracle Database Optimizer, which can reuse parsed SQL statements for better use of server-side resources.

Creating a Layered Subprogram Design

Because PL/SQL allows SQL statements to be seamlessly embedded into the logic, it is too easy to have SQL statement spread all over the code. However, it is recommended that you keep the SQL logic separate from the business logic—that is, create a layered application design with a minimum of two layers:

Data access layer: For subroutines to access the data by using SQL statements



Business logic layer: For subprograms to implement the business processing rules, which may or may not call on the data access layer routines

Oracle Database: Develop PL/SQL

Modularizing Development with PL/SQL Blocks

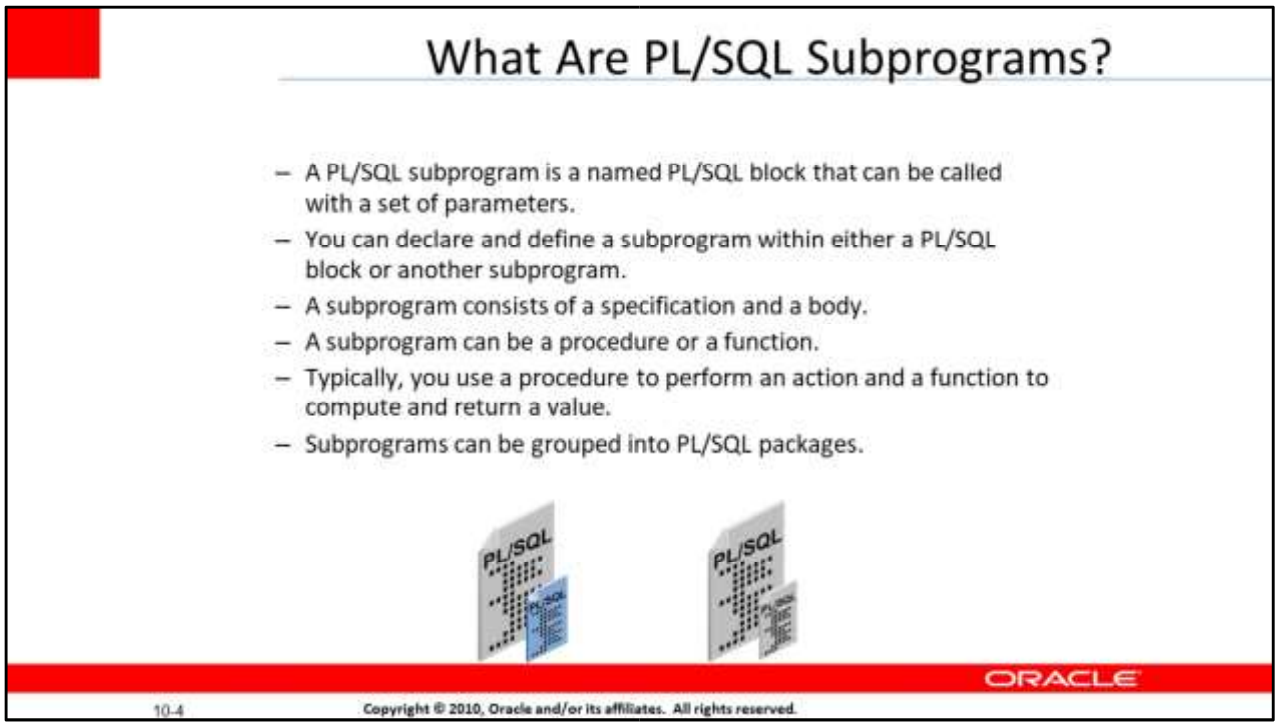
A subprogram is based on standard PL/SQL structures. It contains a declarative section, an executable section, and an optional exception-handling section (for example, anonymous blocks, procedures, functions, packages, and triggers). Subprograms can be compiled and stored in the database, providing modularity, extensibility, reusability, and maintainability.

Modularization converts large blocks of code into smaller groups of code called modules. After modularization, the modules can be reused by the same program

or shared with other programs. It is easier to maintain and debug code that comprises smaller modules than it is to maintain code in a single large program. Modules can be easily extended for customization by incorporating more functionality, if required, without affecting the remaining modules of the program.

Subprograms provide easy maintenance because the code is located in one place and any modifications required to the subprogram can, therefore, be performed in this single location. Subprograms provide improved data integrity and security. The data objects are accessed through the subprogram, and a user can invoke the subprogram only if the appropriate access privilege is granted to the user.

Note: Knowing how to develop anonymous blocks is a prerequisite for this course. For detailed information about anonymous blocks, see the course titled Oracle: PL/SQL Fundamentals.



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.

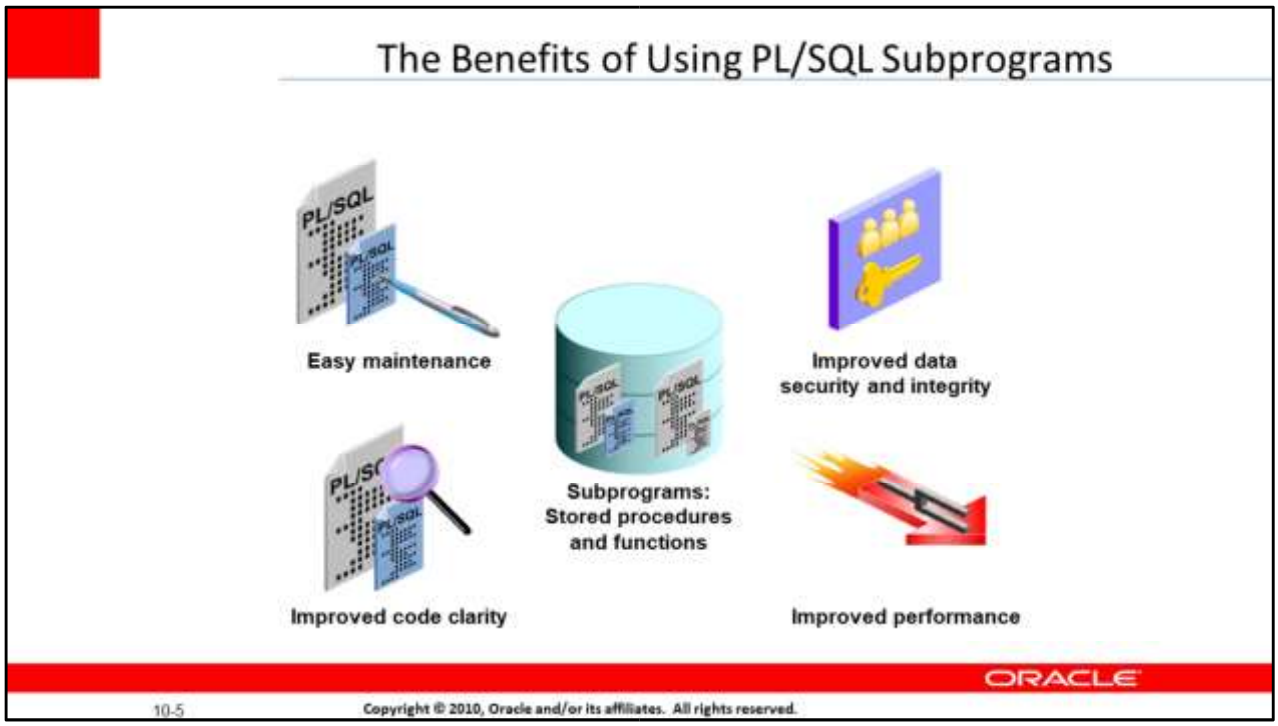
Subprogram Parts

A subprogram consists of a specification (spec) and a body. To declare a subprogram, you must provide the spec, which includes descriptions of any parameters. To define a subprogram, you must provide both the spec and the body. You can either declare a subprogram first and define it later in the same block or subprogram, or declare and define it at the same time.

Subprogram Types

PL/SQL has two types of subprograms: procedures and functions. Typically, you use a procedure to perform an action and a function to compute and return a value. A procedure and a function have the same structure, except that only a function has some additional items such as the RETURN clause or the RETURN statement. The RETURN clause specifies the data type of the return value (required). A RETURN statement specifies the return value (required). Functions are covered

in more detail in the next lesson titled “Creating Functions and Debugging Subprograms.”
Subprograms can be grouped into PL/SQL packages, which make code even more reusable and maintainable. Packages are covered in the packages lessons: Lessons 3 and 4.



Benefits of Subprograms

Procedures and functions have many benefits due to modularizing of code: Easy maintenance is realized because subprograms are located in one place. Modifications need to be done in only one place to affect multiple applications and minimize excessive testing.

Improved data security can be achieved by controlling indirect access to database objects from nonprivileged users with security privileges. The subprograms are by default executed with definer's right. The execute privilege does not allow a calling user direct access to objects that are accessible to the subprogram.

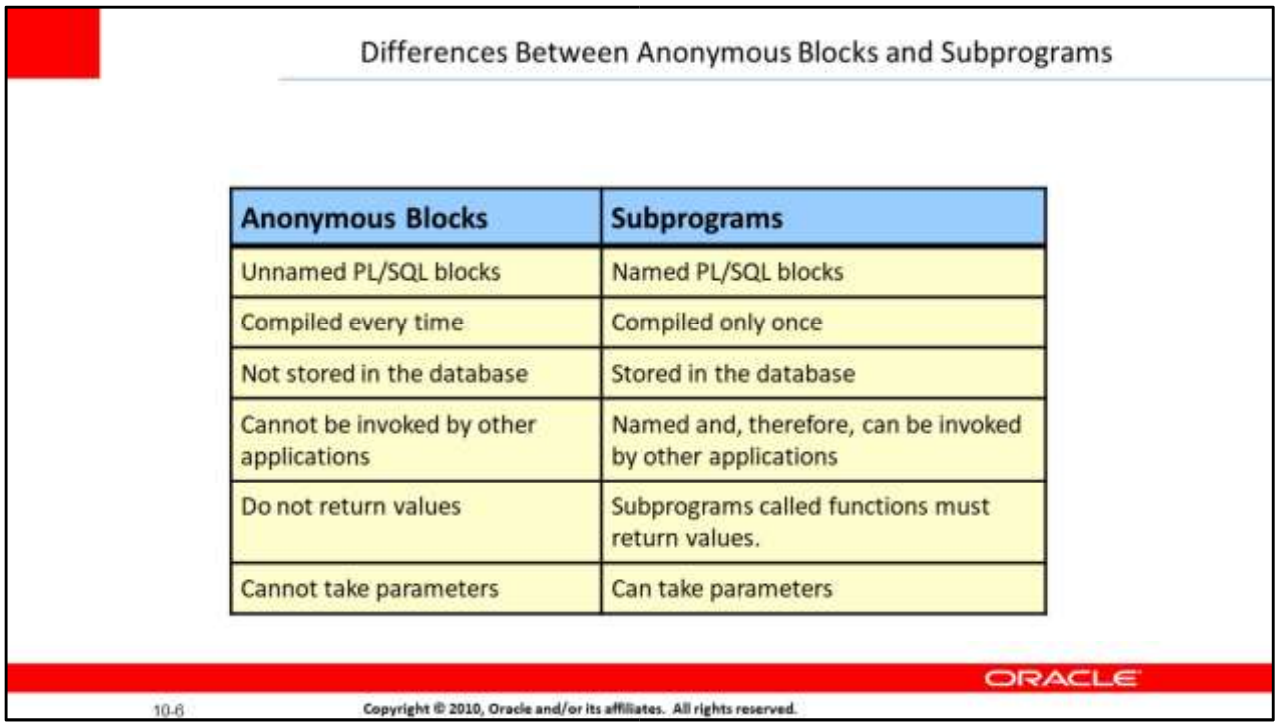
Data integrity is managed by having related actions performed together or not at all.

Improved performance can be realized from reuse of parsed PL/SQL code that becomes available in the shared SQL area of the server.

Subsequent calls to the subprogram avoid parsing the code again. Because PL/SQL code is parsed at compile time, the parsing overhead of SQL statements is avoided at run time. Code can be written to reduce the number of network calls to the database, and therefore, decrease network traffic.

Improved code clarity can be attained by using appropriate names and

conventions to describe the action of the routines, thereby reducing the need for comments and enhancing the clarity of the code.



Differences Between Anonymous Blocks and Subprograms

The table in the slide not only shows the differences between anonymous blocks and subprograms, but also highlights the general benefits of subprograms.

Anonymous blocks are not persistent database objects. They are compiled and executed

only once. They are not stored in the database for reuse. If you want to reuse, you must rerun the script that creates the anonymous block, which causes recompilation and execution.

Procedures and functions are compiled and stored in the database in a compiled form.

They are recompiled only when they are modified. Because they are stored in the database, any application can make use of these subprograms based on appropriate permissions. The calling application can pass parameters to the procedures if the procedure is designed to accept parameters. Similarly, a calling application can retrieve a value if it invokes a function or a procedure.

Program Units 1 - 6