Functions and Procedures

Dr. Abu Raihan Mostofa Kamal

December 9, 2021

Functions and Procedures

Why Functions and Procedure? Modular Code. Modularization is the process by which you break up large blocks of code into smaller pieces (modules) that can be called by other modules. Modularization of code is analogous to normalization of data in RDBMS.

- **More reusable.** Once it is developed can be used by any other user.
- **More manageable.** Which would you rather debug: a 1,000-line program or five individual 200-line programs that call each other as needed? Our minds work better when we can focus on smaller tasks.
- **More readable.** Modules have names, and names describe behavior.
- **More reliable.** The code you produce will have fewer errors. The errors you do find will be easier to fix because they will be isolated within a module.

Functions and Procedures

Why Functions and Procedure? Modular Code. Modularization is the process by which you break up large blocks of code into smaller pieces (modules) that can be called by other modules. Modularization of code is analogous to normalization of data in RDBMS.

- **More reusable.** Once it is developed can be used by any other user.
- **More manageable.** Which would you rather debug: a 1,000-line program or five individual 200-line programs that call each other as needed? Our minds work better when we can focus on smaller tasks.
- **More readable.** Modules have names, and names describe behavior.
- **More reliable.** The code you produce will have fewer errors. The errors you do find will be easier to fix because they will be isolated within a module.

Functions and Procedures

Why Functions and Procedure? Modular Code. Modularization is the process by which you break up large blocks of code into smaller pieces (modules) that can be called by other modules. Modularization of code is analogous to normalization of data in RDBMS.

- **More reusable.** Once it is developed can be used by any other user.
- **More manageable.** Which would you rather debug: a 1,000-line program or five individual 200-line programs that call each other as needed? Our minds work better when we can focus on smaller tasks.
- **More readable.** Modules have names, and names describe behavior.
- **More reliable.** The code you produce will have fewer errors. The errors you do find will be easier to fix because they will be isolated within a module.

Functions and Procedures

Why Functions and Procedure? Modular Code. Modularization is the process by which you break up large blocks of code into smaller pieces (modules) that can be called by other modules. Modularization of code is analogous to normalization of data in RDBMS.

- **More reusable.** Once it is developed can be used by any other user.
- **More manageable.** Which would you rather debug: a 1,000-line program or five individual 200-line programs that call each other as needed? Our minds work better when we can focus on smaller tasks.
- **More readable.** Modules have names, and names describe behavior.
- **More reliable.** The code you produce will have fewer errors. The errors you do find will be easier to fix because they will be isolated within a module.

Forms of Modularization in PL/SQL

- **Procedure.** A program that performs one or more actions and is called as an executable PL/SQL statement. You can pass information into and out of a procedure through its parameter list.
- **Function.** A program that returns data through its RETURN clause, and is used just like a PL/SQL expression. You can pass information into a function through its parameter list.
- **Database trigger.** A set of commands that *are triggered to execute (e.g., log in, modify a row in a table, execute a DDL statement) when an event occurs* in the database.
- **Package.** A named collection of procedures, functions, types, and variables. A package is not really a module (it's more of a meta-module), but it is so closely related that.

Forms of Modularization in PL/SQL

- **Procedure.** A program that performs one or more actions and is called as an executable PL/SQL statement. You can pass information into and out of a procedure through its parameter list.
- **Function.** A program that returns data through its RETURN clause, and is used just like a PL/SQL expression. You can pass information into a function through its parameter list.
- **Database trigger.** A set of commands that *are triggered to execute (e.g., log in, modify a row in a table, execute a DDL statement) when an event occurs* in the database.
- **Package.** A named collection of procedures, functions, types, and variables. A package is not really a module (it's more of a meta-module), but it is so closely related that.

Forms of Modularization in PL/SQL

- **Procedure.** A program that performs one or more actions and is called as an executable PL/SQL statement. You can pass information into and out of a procedure through its parameter list.
- **Function.** A program that returns data through its RETURN clause, and is used just like a PL/SQL expression. You can pass information into a function through its parameter list.
- **Database trigger.** A set of commands that *are triggered to execute (e.g., log in, modify a row in a table, execute a DDL statement) when an event occurs* in the database.
- **Package.** A named collection of procedures, functions, types, and variables. A package is not really a module (it's more of a meta-module), but it is so closely related that.

Forms of Modularization in PL/SQL

- **Procedure.** A program that performs one or more actions and is called as an executable PL/SQL statement. You can pass information into and out of a procedure through its parameter list.
- **Function.** A program that returns data through its RETURN clause, and is used just like a PL/SQL expression. You can pass information into a function through its parameter list.
- **Database trigger.** A set of commands that *are triggered to execute (e.g., log in, modify a row in a table, execute a DDL statement) when an event occurs* in the database.
- **Package.** A named collection of procedures, functions, types, and variables. A package is not really a module (it's more of a meta-module), but it is so closely related that.

Procedure

A procedure is a module that performs one or more actions. It has **no return statement**. Computed values can be returned by **OUT** parameters. A procedure call is a **standalone executable** statement in PL/SQL

Syntax:

```
 [CREATE [OR REPLACE]]
PROCEDURE procedure_name[(parameter[, parameter]...)]
[AUTHID {DEFINER | CURRENT_USER}] {IS | AS}
[PRAGMA AUTONOMOUS_TRANSACTION;]
[local declarations]
BEGIN
executable statements
[EXCEPTION
exception handlers]
END [name];
```

Function

- A function is a module that returns data through its **RETURN** clause, rather than in an OUT or IN OUT argument.
- It may have IN OUT parameters.
- Unlike a procedure call, which is a standalone executable statement, a call to a function can exist only as part of an executable statement, such as an element in an expression or the value assigned as the **default in a declaration of a variable** as well as **within a SELECT statement**.

Function

- A function is a module that returns data through its **RETURN** clause, rather than in an OUT or IN OUT argument.
- It may have IN OUT parameters.
- Unlike a procedure call, which is a standalone executable statement, a call to a function can exist only as part of an executable statement, such as an element in an expression or the value assigned as the **default in a declaration of a variable** as well as **within a SELECT statement**.

Function

- A function is a module that returns data through its **RETURN** clause, rather than in an OUT or IN OUT argument.
- It may have IN OUT parameters.
- Unlike a procedure call, which is a standalone executable statement, a call to a function can exist only as part of an executable statement, such as an element in an expression or the value assigned as the **default in a declaration of a variable** as well as **within a SELECT statement**.

Function:Syntax

```
[CREATE [OR REPLACE]]
FUNCTION function_name[(parameter[, parameter]...)]
RETURN RETURN_TYPE
[AUTHID {DEFINER | CURRENT_USER}] {IS | AS}
[PRAGMA AUTONOMOUS_TRANSACTION;]
[local declarations]
BEGIN
executable statements
[EXCEPTION
exception handlers]

RETURN STATEMENT;
END [name];
```

Common in Procedure and Function

- Has a name.
- Can take parameters, and can return values.
- Is **stored** in the data dictionary.
- Can be called by many users.

Note: The term **stored procedure** is sometimes used generically for both stored procedures and stored functions. Use the `user_source` for source code of each subprogram.

Common in Procedure and Function

- Has a name.
- Can take parameters, and can return values.
- Is **stored** in the data dictionary.
- Can be called by many users.

Note: The term **stored procedure** is sometimes used generically for both stored procedures and stored functions. Use the `user_source` for source code of each subprogram.

Common in Procedure and Function

- Has a name.
- Can take parameters, and can return values.
- Is **stored** in the data dictionary.
- Can be called by many users.

Note: The term **stored procedure** is sometimes used generically for both stored procedures and stored functions. Use the `user_source` for source code of each subprogram.

Common in Procedure and Function

- Has a name.
- Can take parameters, and can return values.
- Is **stored** in the data dictionary.
- Can be called by many users.

Note: The term **stored procedure** is sometimes used generically for both stored procedures and stored functions. Use the `user_source` for source code of each subprogram.

Common in Procedure and Function

- Has a name.
- Can take parameters, and can return values.
- Is **stored** in the data dictionary.
- Can be called by many users.

Note: The term **stored procedure** is sometimes used generically for both stored procedures and stored functions. Use the `user_source` for source code of each subprogram.

Common in Procedure and Function

- Has a name.
- Can take parameters, and can return values.
- Is **stored** in the data dictionary.
- Can be called by many users.

Note: The term **stored procedure** is sometimes used generically for both stored procedures and stored functions. Use the `user_source` for source code of each subprogram.