Creating Procedures, Functions, and Packages

Chapter 9a

Procedures and Functions
A procedure or function is a named PL/SQL block. It may be stored on the database individually or normally stored in the database within package specifications, which is a wrapper for a group of named blocks.
Procedures and Functions

- Advantage of this is that when a block is placed on the database it is parsed at the time it is stored.
- When it is subsequently executed, oracle already has the block compiled and it is therefore much faster.
- Good way of grouping application functionality together and exposing only function calls (not the code itself).
Procedures and Functions

Benefits

- Improved data security and integrity
  - control indirect access to objects from non privileged users with security privileges
  - ensure that related actions are performed together, or not at all, by funneling actions for related tables through a single path
Procedures and Functions

Benefits

- Improved performance
  - avoid reparsing for multiple users by exploiting shared SQL
  - avoid PL/SQL parsing at run time by parsing at compile time
  - reduce the number of calls to the database and decrease network traffic by bundling commands
Procedures and Functions

Benefits

- Improved maintenance
  - modify routines online without interfering with other users
  - modify one routine to affect multiple applications
  - modify one routine to eliminate duplicate testing
Procedures and Functions

Creating

- Creating a Procedure
- Creating a Function
- Dropping Procedures and Functions
Creating a Procedure

- Procedure Body
- Parameters and Modes
- Constraints on Formal Parameters
- Positional and Named Notation
- Parameter Default Values
Creating a Procedure

Procedure Body

- The syntax for creating a procedure is:

  ```sql
  CREATE [OR REPLACE] PROCEDURE procedure_name
  [(parameter_name [mode] [NOCOPY] datatype
  [(:= | DEFAULT) value], ...)] {IS | AS}
  procedure_body
  ```

- The body of a procedure is a PL/SQL block with a declarative, executable, and exception sections.

- There is no DECLARE keyword in a procedure.
  - IS or AS keyword is used to begin the declarative section.
Creating a Procedure
Appropriate and Accurate Names

Naming Procedures:
- a procedure is an executable statement
- a command to the PL/SQL compiler
- name the procedure to be a command of a VerbSubject format such as:
  - CalculateTotalOrder
  - DisplayErrors
  - ConfirmNewEntries
  - AddCustomer
Creating a Procedure

Procedure Body

- A procedure can read in a list of values but won't directly return any value.
- May indirectly return values through the parameter list.
- Procedure name can optionally be included after the final END statement.
  - Name must match the name of the procedure.
Creating a Procedure
Parameters and Modes

- Subprograms may have a parameter list
  - contains one or more parameters that allow you to pass information back and forth between the subprogram
- parameters are defined by its:
  - Name
  - Datatype
  - Mode
  - Default value (optional)
Creating a Procedure
Parameters and Modes

- Syntax for a parameter is:
  parameter_name [mode] [NOCOPY] datatype
  [(:= | DEFAULT) value]

- Where:
  - parameter_name is the name of parameter
  - mode is [IN | OUT | IN OUT]
  - NOCOPY means that a copy is not made
  - datatype can be any PL/SQL or programmer-defined datatype, but cannot be constrained by a size
  - value is a default value of the parameter
Creating a Procedure
Parameters and Modes

- Mode of a parameter specifies whether the parameter can be read from or written to:
  - IN – Read-only
  - OUT – Write-only
  - IN OUT – Read / Write

- If the mode is not explicitly defined, it defaults to IN

- Practice always to specify the mode of a parameter
Creating a Procedure
Parameters and Modes

- NOCOPY makes the parameter a call by reference instead of a call by value
  - normally, PL/SQL passes IN/OUT parameters by value
Creating a Procedure
Constraints on Formal Parameters

When a procedure is called:
- actual parameters are passed from the calling statement
- map to the formal parameters in the procedure body
- constraints
  - cannot be specified in the procedure body
  - are passed through actual parameters
Creating a Procedure
Constraints on Formal Parameters

- A formal parameter can be constrained by using %TYPE in the procedure definition

- When %TYPE is used on a formal parameter, the formal parameter constraints override the actual constraints
Creating a Procedure
Positional and Named Notation

- There are three ways to map the ordering of the passing of actual parameters to a procedure
  - positional
  - named notation
  - blend of both
Creating a Procedure
Positional and Named Notation

- Positional
  - Names used for the formal and actual parameters are independent
  - More concise than named notation
  - Parameters with default values must be at the end of the list
Creating a Procedure
Positional and Named Notation

- Named notation
  - Formal and actual parameters are both specified for each argument in the calling statement
  - Clearly illustrates association between actual and formal parameters
  - Order of formal and actual parameters is independent
  - More difficult to maintain
  - Allows default values for formal parameters regardless of which parameter has the default
Creating a Procedure

Positional and Named Notation

- positional and named notation
  - can be mixed in the same call
  - first arguments must be specified by position
  - remaining arguments can be specified by name
Creating a Procedure
Parameter Default Values

- Similar to variable declarations
  - formal parameters can have default values
- IN Parameters with default values do not need to be passed in the calling statement
- When it is passed
  - the value of the actual parameter is used instead
- When using default values
  - make them the last parameters in the formal argument list
Exiting a Procedure

- A procedure completes its execution when it reaches the procedure’s closing END statement.
- Program control returns to the next statement immediately after the calling statement.
Exiting a Procedure

- Believe it or not, RETURN statements can also be used in procedures
  - RETURN statement does not take an expression
  - Simply halts execution of the procedure and returns control to the calling program
- Feel free to use a RETURN statement just before the END statement as a matter of style
Functions

- Functions are similar to procedures
  - Both take arguments which can be of any mode
  - Although only IN arguments make logical sense within a function call

- Difference is how each is called
  - procedures are called by a statement
  - functions are called as part of an expression
  - function calls are rvalues in expressions
Creating a Function

- Function Syntax
- Return Statement
- Function Style
Creating a Function

Function Syntax

- The syntax for creating a function is:

  ```sql
  CREATE [OR REPLACE] FUNCTION function_name
  [(parameter_name [mode] [NOCOPY] datatype
  [(:= | DEFAULT) value], ...)]
  RETURN return_type {IS | AS}
  function_body
  ```

- The body of a function is a PL/SQL block with a declarative, executable, and exception sections.

- There is no DECLARE keyword in a function.
  - IS or AS keyword is used to begin the declarative section.
Creating a Function
Appropriate and Accurate Names

- Naming Functions:
  - a function is used like an expression in an executable statement
  - returns, or represents, a value
  - name the function to be a noun of a DescribeReturnedValue format such as:
    - NetAmount
    - CompanyName
    - StateSalesTax
    - BestSeller
Creating a Function

Return Statement

- A function can read in a list of values but will explicitly return a single result.
- When calling a function its value returned is:
  - normally assigned to a variable
  - used in an expression
  - used in a program control structure condition test (IF, WHILE, FOR statements, etc)
Creating a Function

Return Statement

- Inside the body of a function there must be at least one RETURN statement
  - has an expression associated with it
  - returns control to the calling environment with a value
Creating a Function

Return Statement

- The general syntax of the RETURN statement is:
  
  ```
  RETURN expression;
  ```

- There can be more than one return statement in a function
  - Only one will be executed each time the function is called
A function can be called anywhere an expression of the same type can be used

- in an assignment statement
  \[ v_{\text{Sales07}} := \text{Book\_Sales} (2007); \]
- to set a default value
  \[ v_{\text{Sales07}} \text{ NUMBER DEFAULT Book\_Sales} (2007); \]
- in a Boolean expression
  \[ \text{IF (Book\_Sales} (2007)) > 96000 \text{ THEN …}; \]
- in a SQL statement
  \[ \text{SELECT … WHERE (Book\_Sales} (2007) > 96000); \]
- As an argument in a parameter list
  \[ \text{Apply\_Discount (v\_Company\_ID, Book\_Sales} (2007)); \]
Procedures vs. Functions

- Procedures and Functions share features:
  - can return more than one value through the OUT, IN OUT parameters
    - Not recommended with functions since the purpose of a function is to return a value as the result of executing the function
  - have declarative, executable and exception sections
  - accept default values for the parameters
  - called using positional or named notation
  - can accept NOCOPY parameters
Procedures and Functions

Dropping

- Similar to dropping a table, procedures and functions can also be dropped from the data dictionary.

- The syntax for dropping a subprogram is:
  - `DROP PROCEDURE procedure_name;`
  - `DROP FUNCTION function_name;`
Local Subprograms

Benefits of Local Subprograms

- There are two central reasons to create local subprocedures:
  - reduce the size of the module by reducing it of common repetitive code
  - improve the readability of code
  - provide a testing area for a future stored subprograms
Subprogram Locations

- Stored Subprograms and the Data Dictionary
  - when a subprogram is created using the CREATE command, it is stored in the database
  - available to all users with proper privilege

- Local Subprograms
  - Do not need to use the word CREATE in the procedure or function statement
  - only visible in the block in which it is declared
  - its scope extends from the point of declaration until the end of the block
Local Subprograms

- A local subprogram is a procedure that is defined in the declaration section of a PL/SQL block (anonymous or named).
- Must appear at end of the declaration section of the block they are defined in.
- A subprogram defined in a declaration section may only be referenced within its defining block.