Working with Oracle SQL

Chapter 10:

Creating Stored Procedures, Functions, and Packages

Chapter Objectives

In this chapter, we will discuss

- Creating procedures and functions
- Package procedures and functions
- Debug syntax and runtime errors

Chapter Concepts

Procedures and Functions

Packages

Debugging

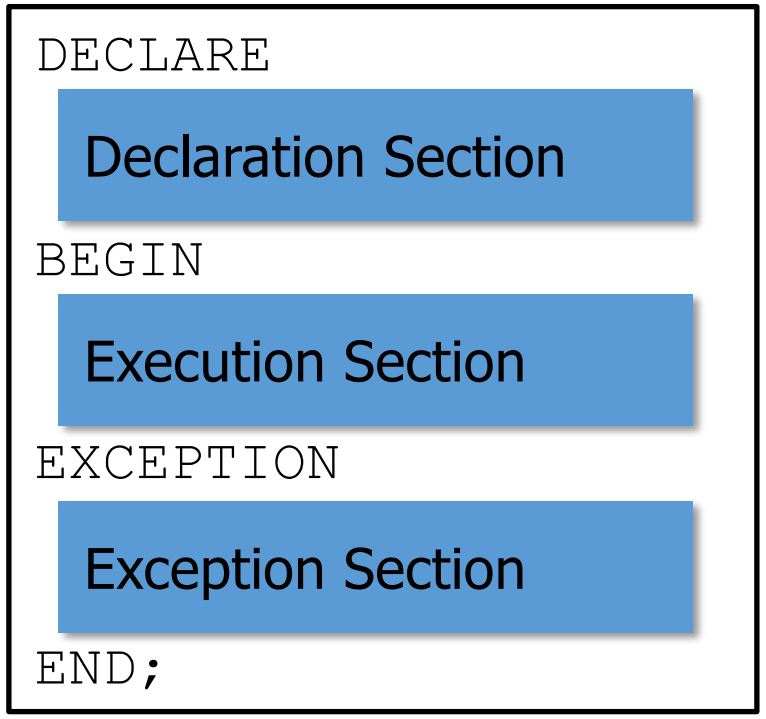
Chapter Summary

Named Blocks in PL/SQL

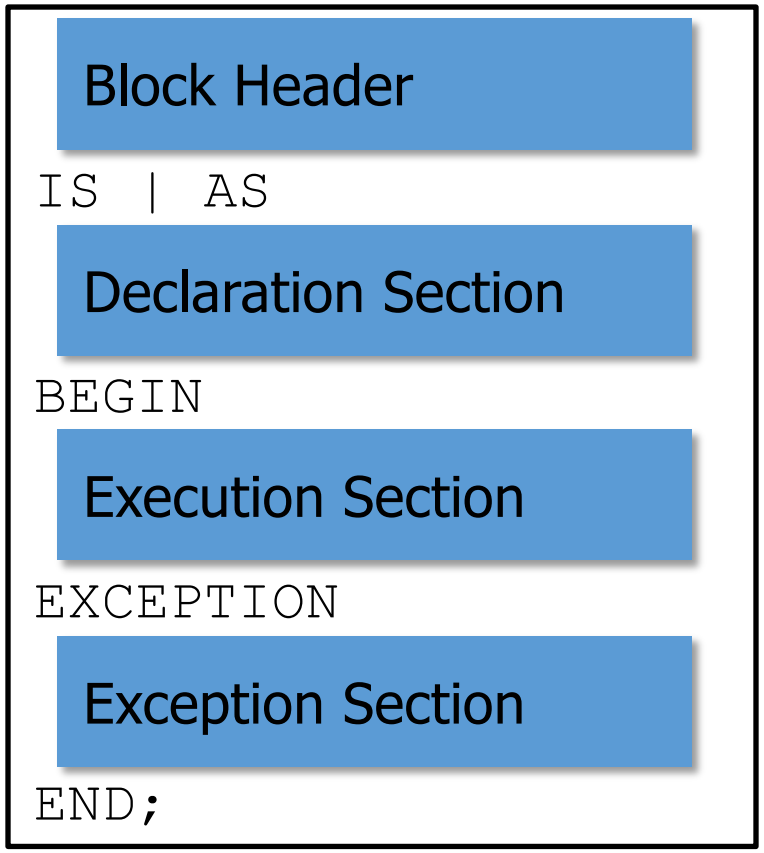
- So far, all the blocks we have seen have been *anonymous*
  - Anonymous blocks are compiled and executed when presented
- Blocks can also be *named*
  - Named blocks are subprograms (procedures and functions)
    - Executed when explicitly called from other blocks
- Subprograms can be stored in the database
  - On their own or contained inside of a package
  - Stored subprograms are visible in the session
    - And can be called by other programs

The Structure of PL/SQL Blocks

Anonymous Block



Named Block



Procedures and Functions

- Use the `CREATE` statement to store procedures and functions in the database
  - The `REPLACE` option recreates the program if it already exists
- Functions are the same as procedures but contain two additional clauses
  - A `RETURN` datatype in the definition
  - A `RETURN` statement in the body
    - Stops execution and returns a single value to the calling program
- Parameters are used to pass data to and from procedures and functions

```
parameter_name [parameter_mode] datatype [default_value_clause]
```

  - Datatype does not include length
  - Three parameter modes: `IN` (default mode), `OUT`, `IN OUT`
  - Name parameters distinctly (`p_`, `parm_`, `in_`) or scope with the name of the block
- Procedures are called in a standalone statement
  - Functions are called when they are used in a statement

Procedures and Function Example

- Create a procedure to make William Smith a programmer
  - Call a function to check if the current number of IT Programmers allows him to be moved into that role

Would normally anchor these declarations, but this illustrates that parameters do not have a size

```
CREATE OR REPLACE PROCEDURE p_change_job(
    job_id      IN VARCHAR2,
    employee_id IN NUMBER
)
IS
BEGIN
    IF f_can_promote(p_change_job.job_id) THEN
        UPDATE employees e
        SET    e.job_id      = p_change_job.job_id
        WHERE  e.employee_id =
p_change_job.employee_id;
    END IF;
END;
```

```
CREATE OR REPLACE FUNCTION f_can_promote(
    job_id IN jobs.job_id%TYPE
) RETURN BOOLEAN
IS
    count_job NUMBER(3);
BEGIN
    SELECT COUNT(*)
    INTO   f_can_promote.count_job
    FROM   employees e
    WHERE  e.job_id = f_can_promote.job_id;

    RETURN (count_job < 5);
END;
```

Calling Stored Procedures

- Call a procedure in an anonymous block, or another procedure:

```
BEGIN
    p_change_job('IT_PROG', 171);
END;
```

- Can also pass parameters by name:

```
BEGIN
    p_change_job(job_id => 'IT_PROG',
                 employee_id => 171);
END;
```

  - Especially useful if there are optional parameters:

```
BEGIN
    p_change_job(employee_id => 171);
END;
```

  - But best practice in all cases

```
CREATE OR REPLACE PROCEDURE p_change_job(
    job_id      IN jobs.job_id%TYPE := 'IT_PROG',
    employee_id IN employees.employee_id%TYPE
)
IS …
```

Dropping Procedures and Functions

- Use the `DROP` statement to remove a procedure or function from the database
- Syntax:

```
DROP PROCEDURE procedure_name;

DROP FUNCTION function_name;
```

- Example:
  - Remove `f_can_promote` function from the database

```
DROP FUNCTION f_can_promote;
```

Chapter Concepts

Procedures and Functions

Packages

Debugging

Chapter Summary

What Is a Package?

- Set of logically related objects
  - Groups procedures and functions
  - Stored in the database
- Example:
  - Define a package that will contain all processing associated with the `employees` table

Package Example

Application

```
BEGIN
   emp_maint.delete_employee(...);
END;
```

```
BEGIN
   emp_maint.update_employee(...);
END;
```

```
BEGIN
   emp_maint.delete_employee(...);
END;
```

Database

```
PACKAGE emp_maint IS
```

```
PROCEDURE delete_employee(...)
IS
BEGIN
...
END delete_employee;
```

```
PROCEDURE update_employee(...)
IS
BEGIN
...
END update_employee;
```

```
END emp_maint;
```

Employees table

Package Components

- Each package consists of a specification and a body
  - Declared separately
- Package specification is used to declare package components
  - Declared objects are public (available to any PL/SQL block)
  - Each entry is a procedure or function declaration

```
CREATE [OR REPLACE] PACKAGE package_name
[ IS | AS ]
    object_declarations
END package_name;
```

- Package body is used to define the package content
  - Contains definition and body of grouped procedures and functions
    - Must contain the code for all procedures and functions declared in the specification
    - Can also contain local procedures and functions used only within the package

```
CREATE [OR REPLACE] PACKAGE BODY package_name
[ IS | AS ]
    object_body_definitions
END package_name;
```

Creating a Package: Example

- Convert the previous procedure and function into a package

Package specification contains declaration

Package body contains definitions

Items must be defined or declared before they are used

```
CREATE OR REPLACE PACKAGE pack_promote
IS
  PROCEDURE p_change_job(
    job_id IN jobs.job_id%TYPE := 'IT_PROG',
    employee_id IN employees.employee_id%TYPE
  );
END pack_promote;
```

```
CREATE OR REPLACE PACKAGE BODY pack_promote IS
  FUNCTION f_can_promote(
    job_id IN jobs.job_id%TYPE
  ) RETURN BOOLEAN
  IS
    count_job NUMBER(3);
  BEGIN
    SELECT COUNT(*)
    INTO   f_can_promote.count_job
    FROM   employees e
    WHERE  e.job_id = f_can_promote.job_id;

    RETURN (count_job < 5);
  END f_can_promote;

  PROCEDURE p_change_job(
    job_id IN jobs.job_id%TYPE,
    employee_id IN employees.employee_id%TYPE
  )
  IS
  BEGIN
    IF f_can_promote(p_change_job.job_id) THEN
      UPDATE employees e
      SET    e.job_id      = p_change_job.job_id
      WHERE  e.employee_id = p_change_job.employee_id;
    END IF;
  END p_change_job;
END pack_promote;
```

Calling Packaged Procedures and Functions

- A packaged procedure or function is called using the dot notation
- Syntax:

```
package_name.procedure_name(parameter1, ..., parameterN);

variable_name  := package_name.function_name(parameter1, ...,
parameterN);
```

- Example:

```
BEGIN
    pack_promote.p_change_job('IT_PROG', 171);
END;
```

```
BEGIN
    pack_promote.p_change_job(job_id => 'IT_PROG',
                              employee_id => 171);
END;
```

Dropping Package Specification and Body

- Use the `DROP` statement to remove package specification and/or package body
- Syntax:

```
DROP PACKAGE [BODY] package_name;
```

- Rules:
  - `BODY` is an optional key word
    - If it is specified, the package body is dropped
    - If it is not specified, both the body and specification are dropped
- Example:
  - Remove `pack_promote` body and specification

```
DROP PACKAGE pack_promote;
```

Advantages of Packages

- Modular
  - Logical group of multiple related procedures and functions
- Information hiding
  - Hides local (private) procedures and functions
- Persistent variables
  - Package variables are global variables that retain values during a session
- Better performance
  - When an object within a package is referenced, the whole package is loaded into memory; any subsequent calls to other objects within the package require no disk I/O
- Avoid dependency problems among procedures and functions
  - Can recompile the package body without invalidating procedures and functions that call it

Chapter Concepts

Procedures and Functions

Packages

Debugging

Chapter Summary

Debugging Compilation Errors

- Procedures, functions, and packages are compiled when they are created
  - A message is displayed if an error occurs when the `CREATE` command is issued
  - Compilation errors can be viewed using the `SHOW ERRORS` command
- Syntax:

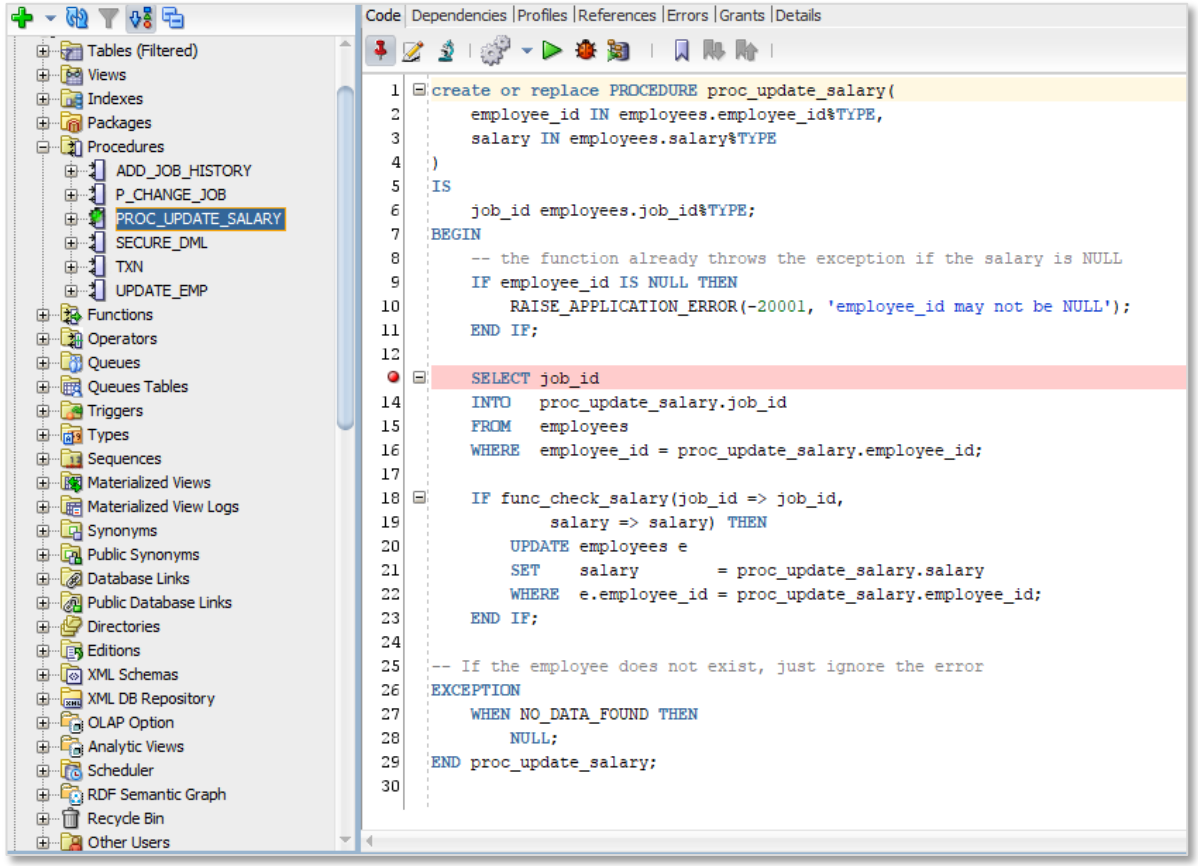
```
SHOW ERRORS [PROCEDURE | FUNCTION | PACKAGE | PACKAGE
   BODY object_name]
```

- `SHOW ERRORS` command
  - Shows compilation errors for the most recently created object or the named object
  - Displays line and column number of the error (`LINE/COL`) and the error itself (`ERROR`)
- `SHOW ERRORS` is a SQL*Plus command that queries the `user_errors` dictionary view

Debugging in SQL Developer

- To use the SQL Developer debugger, the procedure must be compiled in debug mode:
  - Right-click in explorer view and select **Compile for Debug**
  - Or run `ALTER PROCEDURE name COMPILE DEBUG`

- To start a debug session:
  - Right-click in explorer view and choose **DEBUG** from the menu, which will prompt for parameters
  - Write an anonymous block that sets up parameters, right-click it, and choose **DEBUG** from the menu
- Set breakpoints by opening the procedure from explorer view



Exercise 10.1: Stored Procedures, Functions, and Packages

30 min

- Please complete this exercise in your Exercise Manual

Chapter Concepts

Procedures and Functions

Packages

Debugging

Chapter Summary

Chapter Summary

In this chapter, we have discussed:

- Creating procedures and functions
- Packaging procedures and functions
- Debugging syntax and runtime errors