How to work with packages

A *package* is a container that organizes and groups related stored procedures and user-defined functions. In chapter 13, for example, you learned how to use the built-in package named DBMS_OUTPUT to print data to the output window. Now, you'll learn how to store your own procedures and functions within a package.

The Oracle documentation occasionally uses the term *subprogram* as a generic term that can refer to a procedure or a function. This makes sense if you recognize that procedures and functions are essentially small programs.

How to create a package

Figure 15-11 shows how to create a package. To start, you use the CREATE PACKAGE statement to create the *specification* for the package. A package specification lists the names and parameters for all of the procedures and functions that are contained within the package. In addition, it specifies the return types for its functions. In this figure, for instance, the first example specifies that the package named murach contains one procedure and one function. It specifies two parameters for the procedure. It specifies one parameter for the function. And it specifies the return type for the function.

Once you've created the specification for a package, you can use the CREATE PACKAGE BODY statement to create the *body* for the package. A package body contains the code for its procedures and functions. Within the body, the names, parameters, and return types must match the names, parameters, and return types in the specification. In this figure, you can see that the package specification matches the package body. However, if the specification doesn't match the body, an error is displayed when Oracle attempts to compile the package.

The syntax for defining the specification for a package

```
CREATE [OR REPLACE] PACKAGE package_name {IS | AS}
   prodedure_or_function_specification_1;
  [prodedure_or_function_specification_2;]...
END [package_name];
/
```

Code that defines the specification for a package named murach

```
CREATE OR REPLACE PACKAGE murach AS

  PROCEDURE update_invoices_credit_total
  (invoice_number_param VARCHAR2, credit_total_param NUMBER);

  FUNCTION get_vendor_id
  (vendor_name_param VARCHAR2)
  RETURN NUMBER;

END murach;
/
```

The syntax for defining the body for a package

```
CREATE [OR REPLACE] PACKAGE BODY package_name {IS | AS}
   prodedure_or_function_body_1;
  [prodedure_or_function_body_2;]...
END [package_name];
/
```

Code that defines the body for a package named murach

```
CREATE OR REPLACE PACKAGE BODY murach AS

  PROCEDURE update_invoices_credit_total
  (
    invoice_number_param    VARCHAR2,
    credit_total_param      NUMBER
  )
  AS
  BEGIN
    UPDATE invoices
    SET credit_total = credit_total_param
    WHERE invoice_number = invoice_number_param;

    COMMIT;
  EXCEPTION
    WHEN OTHERS THEN
      ROLLBACK;
  END;
```

Figure 15-11 How to create and drop packages (part 1 of 2)

In general, you can use the techniques you learned earlier in this chapter to call a procedure or function that's stored in package. However, you must preface the name of the procedure or function with the name of the package. In part 2 of this figure, for example, the CALL statement calls the procedure that's stored in the murach package, and the SELECT statement uses the function that's stored in the murach package. Note that storing this procedure and function in a package prevents a naming conflict with a procedure and function of the same name that aren't stored in a package.

In this figure, the package contains just one short procedure and one short function. In practice, though, a package might include dozens of procedures and functions and some of them might be far more complex than the ones in this chapter. A more realistic package might also include global variables that can be shared between procedures and functions.

How to drop a package

As you would expect, you use the DROP PACKAGE statement to drop a package. When you use this statement, the specification and the body for the package are dropped. In this figure, for example, the first DROP statement drops the specification and the body for the package named murach.

In some cases, though, you may only want to drop the body for the package. That way, other procedures or functions that reference the package specification can still be compiled. In that case, you can use the DROP PACKAGE BODY statement to drop only the body of the package.

Advantages of packages

Although you don't need to store your procedures and functions within a package, there are several advantages to using a package. First, packages help you organize your code so all related code is stored in the same package. This has the added benefit of avoiding naming conflicts with other procedures or functions. Second, packages provide some enhanced functionality. For example, if you declare a variable outside of a procedure or function, that variable is available within the package for the entire session. Third, packages generally provide improved performance since they are loaded into memory when they are first used. Last, package specifications reduce unnecessary recompiling of dependent procedures and functions in other packages.

Code that defines the body for a package named murach (continued)

```
    FUNCTION get_vendor_id
    (vendor_name_param VARCHAR2)
    RETURN NUMBER
    AS
      vendor_id_var NUMBER;
    BEGIN
      SELECT vendor_id
      INTO vendor_id_var
      FROM vendors
      WHERE vendor_name = vendor_name_param;

      RETURN vendor_id_var;
    END;

END murach;
/
```

A statement that calls a procedure that's stored in a package

```
CALL murach.update_invoices_credit_total('367447', 200);
```

A SELECT statement that calls a function that's stored in a package

```
SELECT invoice_number, invoice_total
FROM invoices
WHERE vendor_id = murach.get_vendor_id('IBM');
```

The response from the system

	INVOICE_NUMBER	INVOICE_TOTAL
1	QP58872	116.54
2	Q545443	1083.58

A statement that drops the specification and body for a package

```
DROP PACKAGE murach;
```

A statement that drops only the body for a package

```
DROP PACKAGE BODY murach;
```

Description

- A *package* is a container that allows you to organize and group related stored procedures and user-defined functions.
- The *specification* of a package lists the names and parameters for its procedures and functions. In addition, it specifies the return types for its functions.
- The *body* of a package contains the code for its procedures and functions. Within the body, the names, parameters, and return types must match the names, parameters, and return types in the specification.

Figure 15-11 How to create and drop packages (part 2 of 2)