**Packages**

Package is used to group procedures and functions together into packages. Packages allow you to encapsulate related functionality into one self-contained unit. By modularizing your PL/SQL code through the use of packages, you build up your own libraries of code that other programmers can reuse. In fact, the Oracle database comes with a library of packages, which allow you to access external files, manage the database, generate HTML, and much more; to see all the packages, you should consult the Oracle Database PL/SQL Packages and Types Reference manual from Oracle Corporation.

Packages are typically made up of two components: a specification and a body. The package specification lists the available procedures, functions, types, and objects. You can make the items listed in the specification available to all database users, and I refer to these items as being public (although only users you have granted privileges to access your package can use it). The specification doesn’t contain the code that makes up the procedures and functions; the code is contained in the package body.

Any items in the body that are not listed in the specification are private to the package. Private items can be used only inside the package body. By using a combination of public and private items, you can build up a package whose complexity is hidden from the outside world. This is one of the primary goals of all programming: hide complexity from your users.

**Creating Package Specification**

You create a package specification using the CREATE PACKAGE statement. The simplified syntax for the

CREATE PACKAGE statement is as follows:

CREATE [OR REPLACE] PACKAGE package\_name {IS | AS} package\_specification END package\_name;

where

package\_name is the name of the package. package\_specification lists the public procedures, functions, types, and objects available to your package’s users. The following example creates a specification for a package named product\_package:

CREATE PACKAGE product\_package AS

TYPE t\_ref\_cursor IS REF CURSOR;

FUNCTION get\_products\_ref\_cursor RETURN t\_ref\_cursor;

PROCEDURE update\_product\_price ( p\_product\_id IN products.product\_id%TYPE, p\_factor IN NUMBER );

END product\_package;

/

**Creating Package Body**

You create a package body using the CREATE PACKAGE BODY statement. The simplified syntax for the CREATE PACKAGE BODY statement is as follows:

CREATE [OR REPLACE] PACKAGE BODY package\_name {IS | AS} package\_body END package\_name;

where package\_name is the name of the package, which must match the package name in the specification. package\_body contains the code for the procedures and functions. The following example creates the package body for product\_package:

CREATE PACKAGE BODY product\_package AS FUNCTION get\_products\_ref\_cursor RETURN t\_ref\_cursor IS v\_products\_ref\_cursor t\_ref\_cursor; BEGIN -- get the REF CURSOR OPEN v\_products\_ref\_cursor FOR SELECT product\_id, name, price FROM products;

-- return the REF CURSOR RETURN v\_products\_ref\_cursor; END get\_products\_ref\_cursor; PROCEDURE update\_product\_price ( p\_product\_id IN products.product\_id%TYPE, p\_factor IN NUMBER ) AS v\_product\_count INTEGER; BEGIN -- count the number of products with the -- supplied product\_id (will be 1 if the product exists) SELECT COUNT(\*) INTO v\_product\_count FROM products WHERE product\_id = p\_product\_id; -- if the product exists (v\_product\_count = 1) then -- update that product's price IF v\_product\_count = 1 THEN UPDATE products SET price = price \* p\_factor WHERE product\_id = p\_product\_id; COMMIT; END IF; EXCEPTION WHEN OTHERS THEN ROLLBACK; END update\_product\_price; END product\_package; /

**Calling Procedures and Functions in a package**

SELECT product\_package.get\_products\_ref\_cursor FROM dual;

CALL product\_package.update\_product\_price(3, 1.25);

SELECT price FROM products WHERE product\_id = 3;

**Getting information about package**

SELECT object\_name, procedure\_name FROM user\_procedures WHERE object\_name = 'PRODUCT\_PACKAGE';

**Drop Package**

**DROP PACKAGE product\_package;**