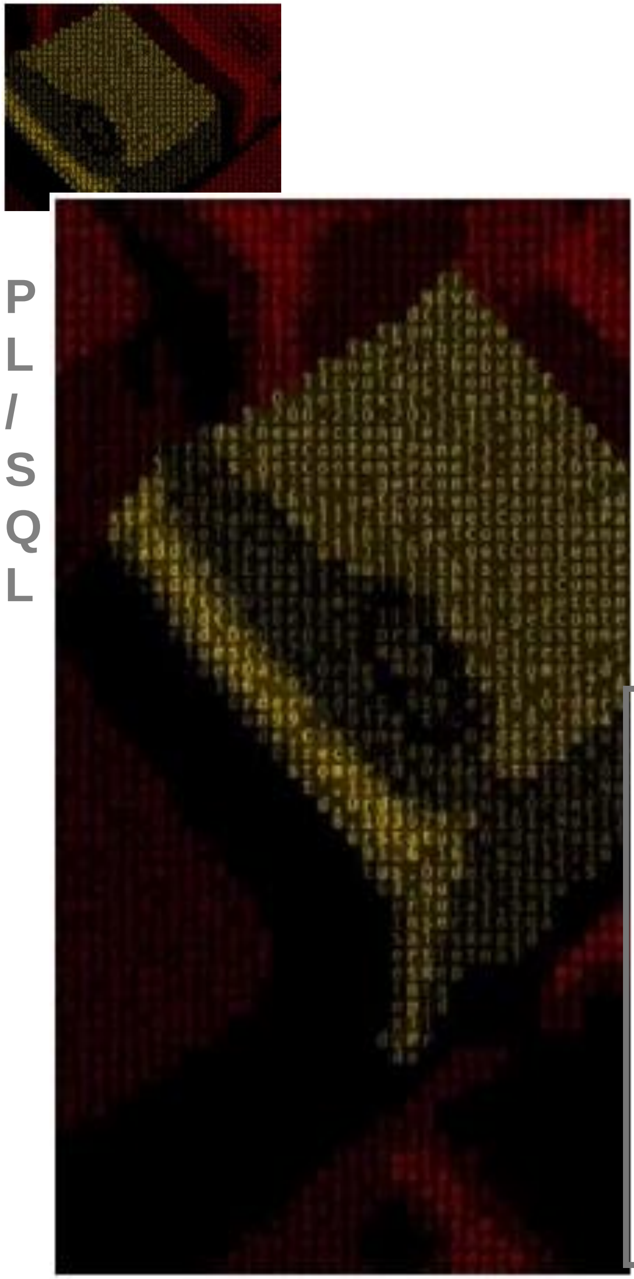


PL/SQL

Oracle11*g*:
PL/SQL Programming

Chapter 7

PL/SQL Packages

Chapter Objectives

P
L
/
S
Q
L

- After completing this lesson, you should be able to understand:
  - Creating packages
  - Invoking program units in packages
  - Including a forward declaration
  - Creating one-time-only procedures
  - Overloading program units
  - Managing restrictions on packaged functions used in SQL

Chapter Objectives (continued)

PL/SQL

- After completing this lesson, you should be able to understand (continued):
  - Using a cursor variable in a package
  - Granting execute privileges
  - Finding package information with data dictionary views
  - Deleting or removing packages

Packages

PL/SQL

- Containers that can hold multiple program units
- Add functionality
  - Private program units
  - Sharing variable values
  - Overloading
  - Ease privilege granting
  - Improve performance

Brewbean’s Challenge

PL/SQL

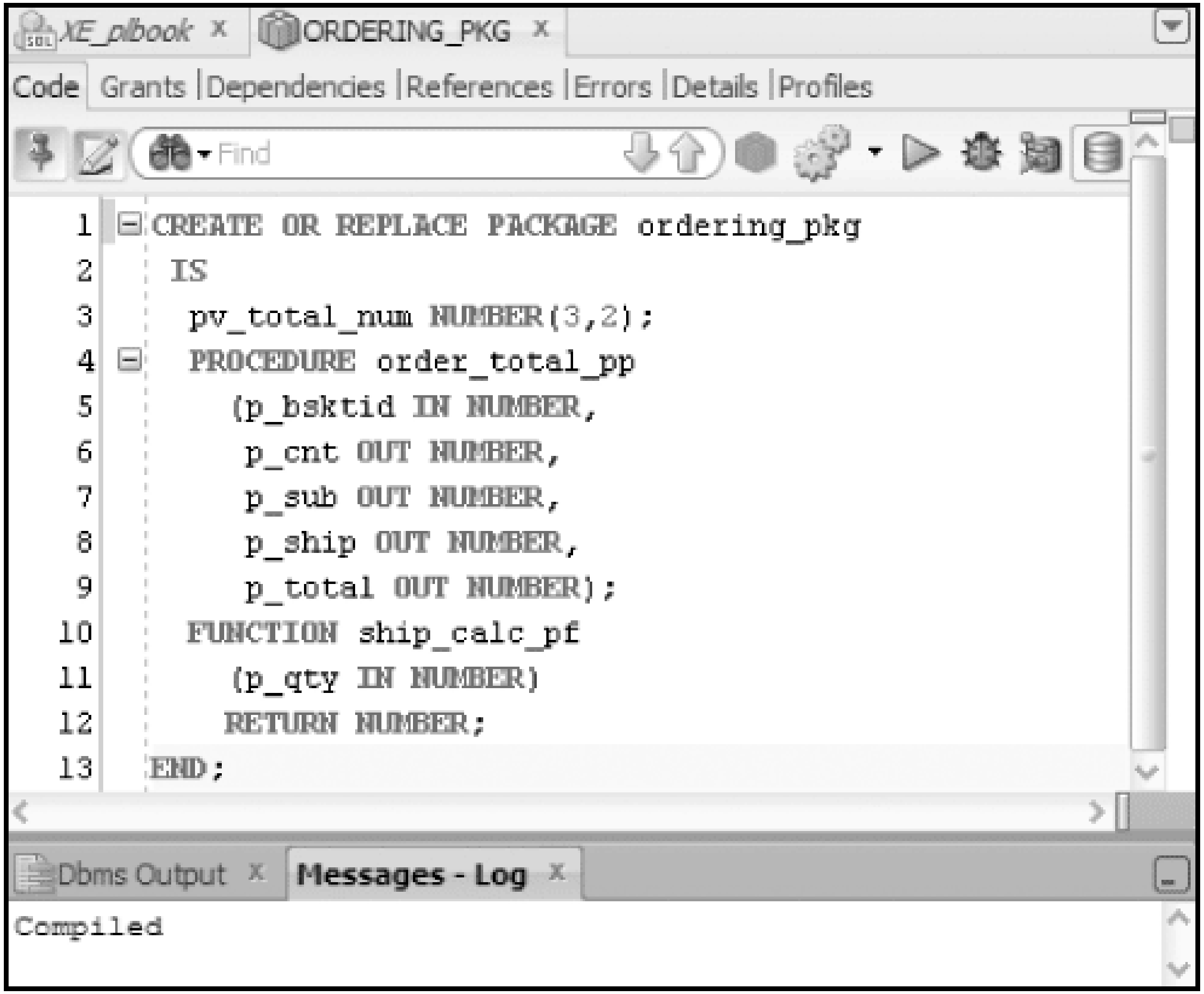
- Organize the many program units developed for the application
- Store values throughout a user session
- Enable a program unit to handle different data types for arguments
- Ease the granting of privileges to users

Package Specification

PL/SQL

- Contains declarations for program units, variables, exceptions, cursors, and types
- Declare program units with the header only
- Order of declarations important if one construct refers to another in the specification

Package Specification (continued)



```
CREATE OR REPLACE PACKAGE ordering_pkg
 IS
  pv_total_num NUMBER(3,2);
  PROCEDURE order_total_pp
    (p_bsktid IN NUMBER,
     p_cnt OUT NUMBER,
     p_sub OUT NUMBER,
     p_ship OUT NUMBER,
     p_total OUT NUMBER);
  FUNCTION ship_calc_pf
    (p_qty IN NUMBER)
   RETURN NUMBER;
END;
```

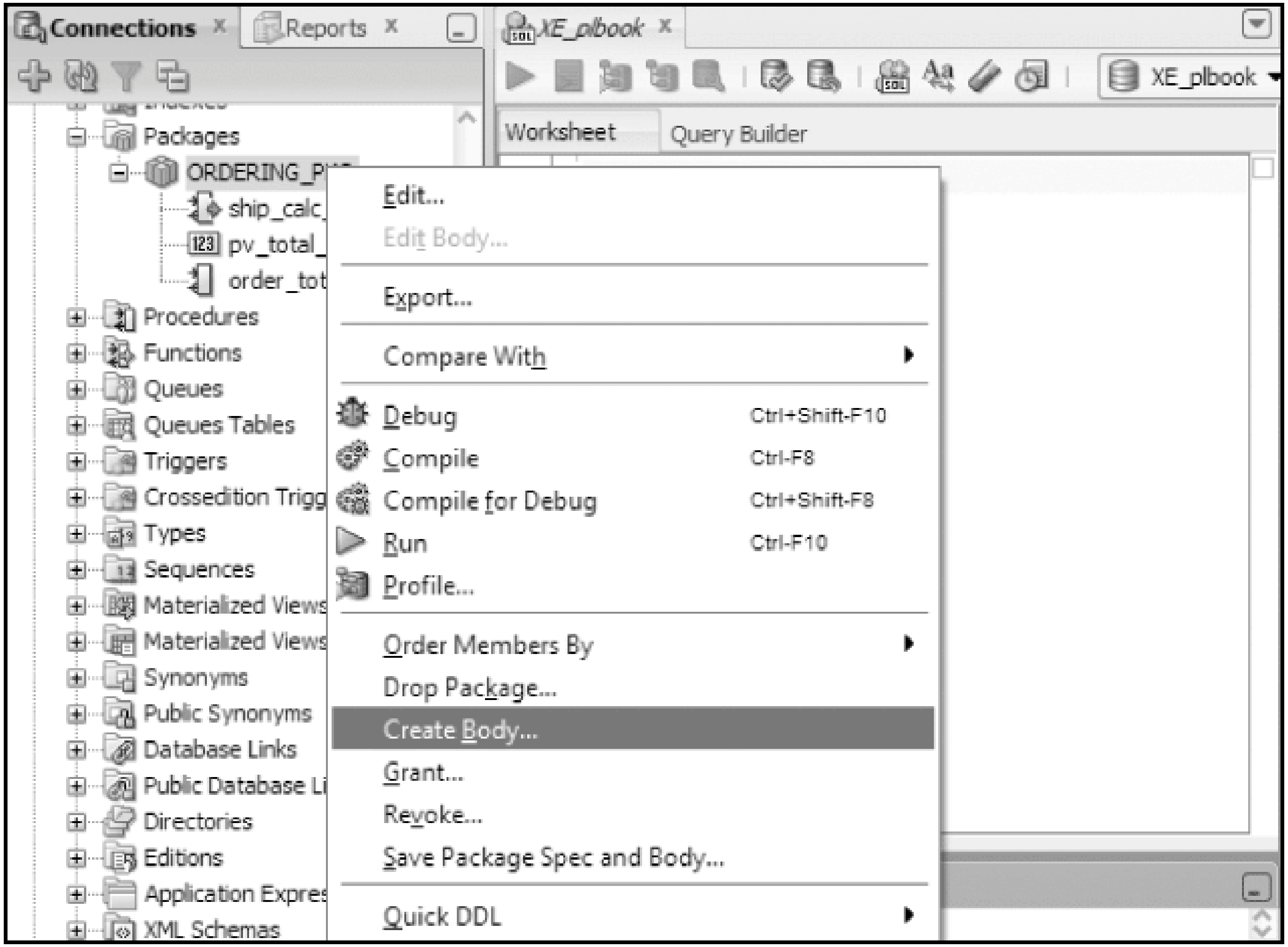
```
Compiled
```

Package Body

- Contains the entire program unit code for those declared in the specification
- Use program unit name in END statement to make more readable
- Also can declare any new constructs not in the specification; however, these can only be used inside this package

P
L
/
S
Q
L

Package Body



Invoking Package Constructs

PL/SQL

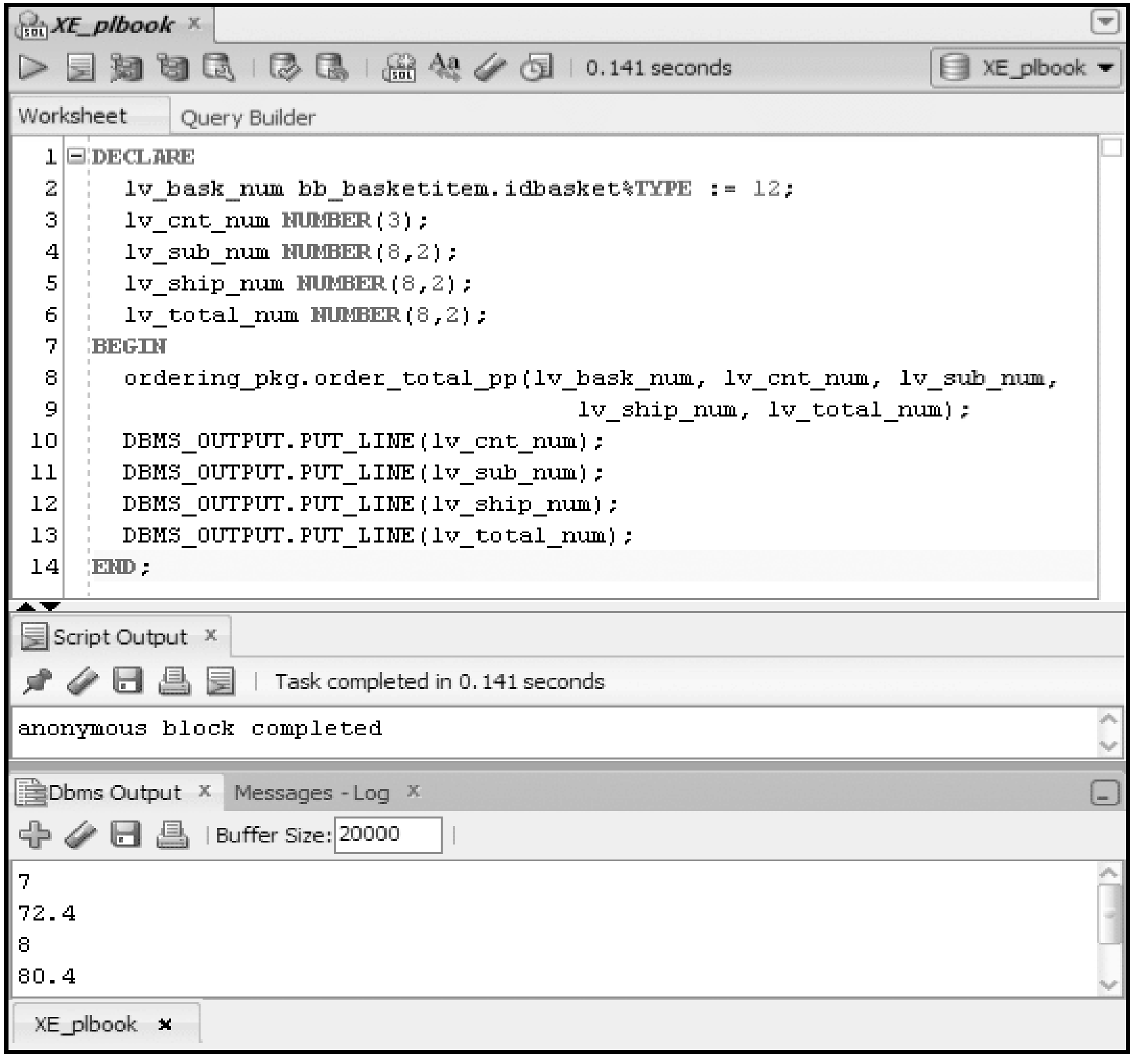
- Call packaged program units the same way as we handled stand-alone program units except add a package name prefix

  package_name.program_unit_name(args,…);

- Reference other packaged constructs such as a variable also using a package name prefix

  package_name.variable_name

Invoking Package Constructs



```
DECLARE
  lv_bask_num bb_basketitem.idbasket%TYPE := 12;
  lv_cnt_num NUMBER(3);
  lv_sub_num NUMBER(8,2);
  lv_ship_num NUMBER(8,2);
  lv_total_num NUMBER(8,2);
BEGIN
  ordering_pkg.order_total_pp(lv_bask_num, lv_cnt_num, lv_sub_num,
                              lv_ship_num, lv_total_num);
  DBMS_OUTPUT.PUT_LINE(lv_cnt_num);
  DBMS_OUTPUT.PUT_LINE(lv_sub_num);
  DBMS_OUTPUT.PUT_LINE(lv_ship_num);
  DBMS_OUTPUT.PUT_LINE(lv_total_num);
END;
```

Task completed in 0.141 seconds

```
anonymous block completed
```

Buffer Size: 20000

```
7
72.4
8
80.4
```

Package Construct Scope

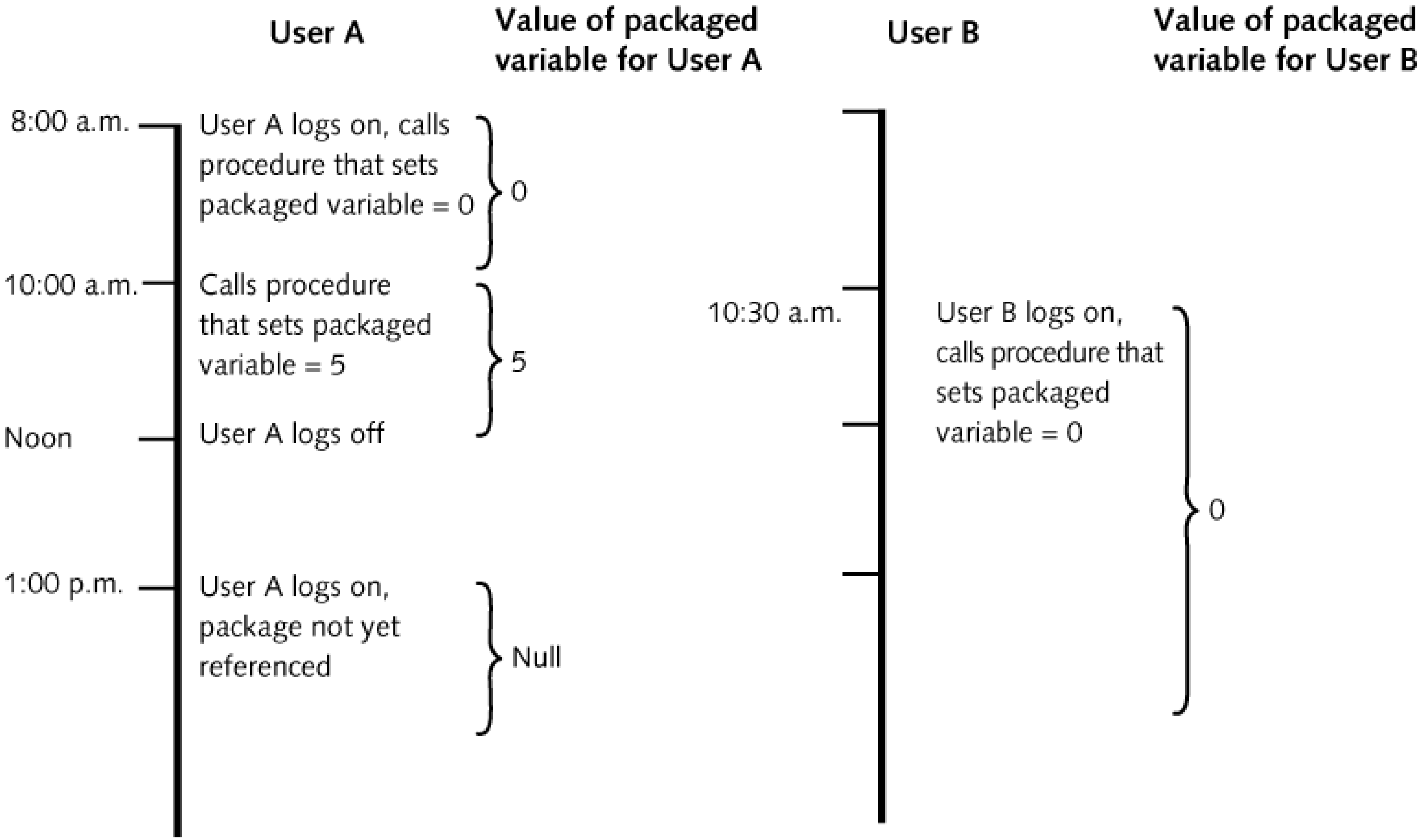
PL/SQL

- Any constructs declared in the specification are public and can be referenced from inside or outside the package
- Any constructs in the body only are private and can only be referenced by other constructs within the same package body

Package Global Constructs

- Constructs declared in the specification such as variables, cursors, types, and exceptions are global
- Global means that the value will persist throughout a user session
- Each user session maintains a separate instance of the packaged construct

Package Global Constructs (continued)



User A

Value of packaged variable for User A

User B

Value of packaged variable for User B

8:00 a.m. — User A logs on, calls procedure that sets packaged variable = 0 — 0

10:00 a.m. — Calls procedure that sets packaged variable = 5 — 5

Noon — User A logs off

1:00 p.m. — User A logs on, package not yet referenced — Null

10:30 a.m. — User B logs on, calls procedure that sets packaged variable = 0 — 0

Package Specification

PL/SQL

- A specification can exist without a body
- Used to store often referenced static values
- Example

```
CREATE OR REPLACE PACKAGE metric_pkg IS
  cup_to_liter CONSTANT NUMBER := .24;
  pint_to_liter CONSTANT NUMBER := .47;
  qrt_to_liter CONSTANT NUMBER := .95;
END;
```

Improving Processing Efficiency

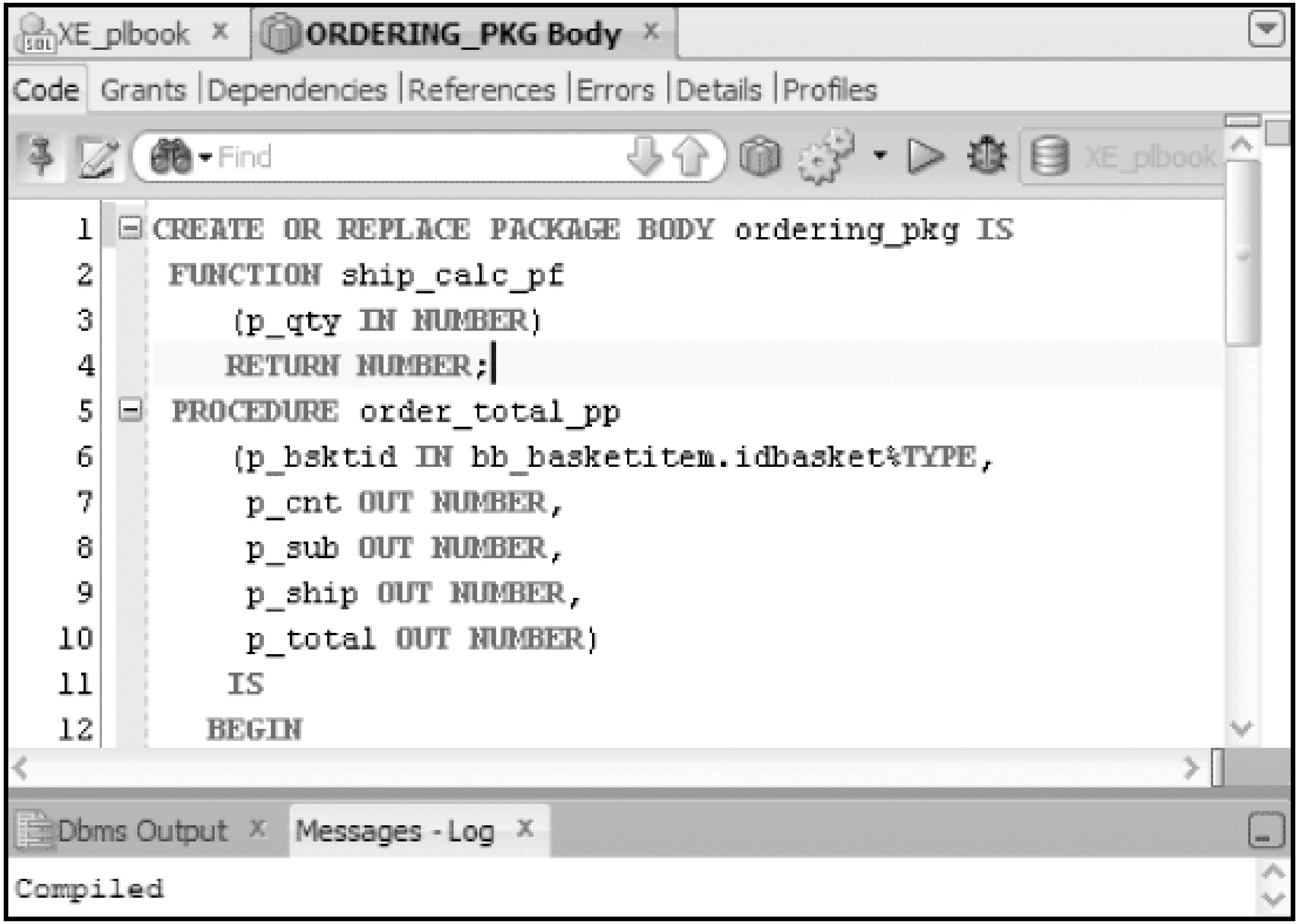
PL/SQL

- Packaged constructs such as variables and cursors are stored in memory
- After the initial call, values can then be retrieved from cache in subsequent calls
- Package code is also cached

Forward Declarations

- Private program units must be ordered so that any referenced unit is located prior to the calling program unit in the package body
- You need a workaround if you want to organize program units in the body
- Forward declarations eliminate the order problem
- A forward declaration is the program unit header at the top of the package body

Forward Declarations



```
CREATE OR REPLACE PACKAGE BODY ordering_pkg IS
 FUNCTION ship_calc_pf
     (p_qty IN NUMBER)
     RETURN NUMBER;
 PROCEDURE order_total_pp
     (p_bsktid IN bb_basketitem.idbasket%TYPE,
      p_cnt OUT NUMBER,
      p_sub OUT NUMBER,
      p_ship OUT NUMBER,
      p_total OUT NUMBER)
     IS
   BEGIN
```

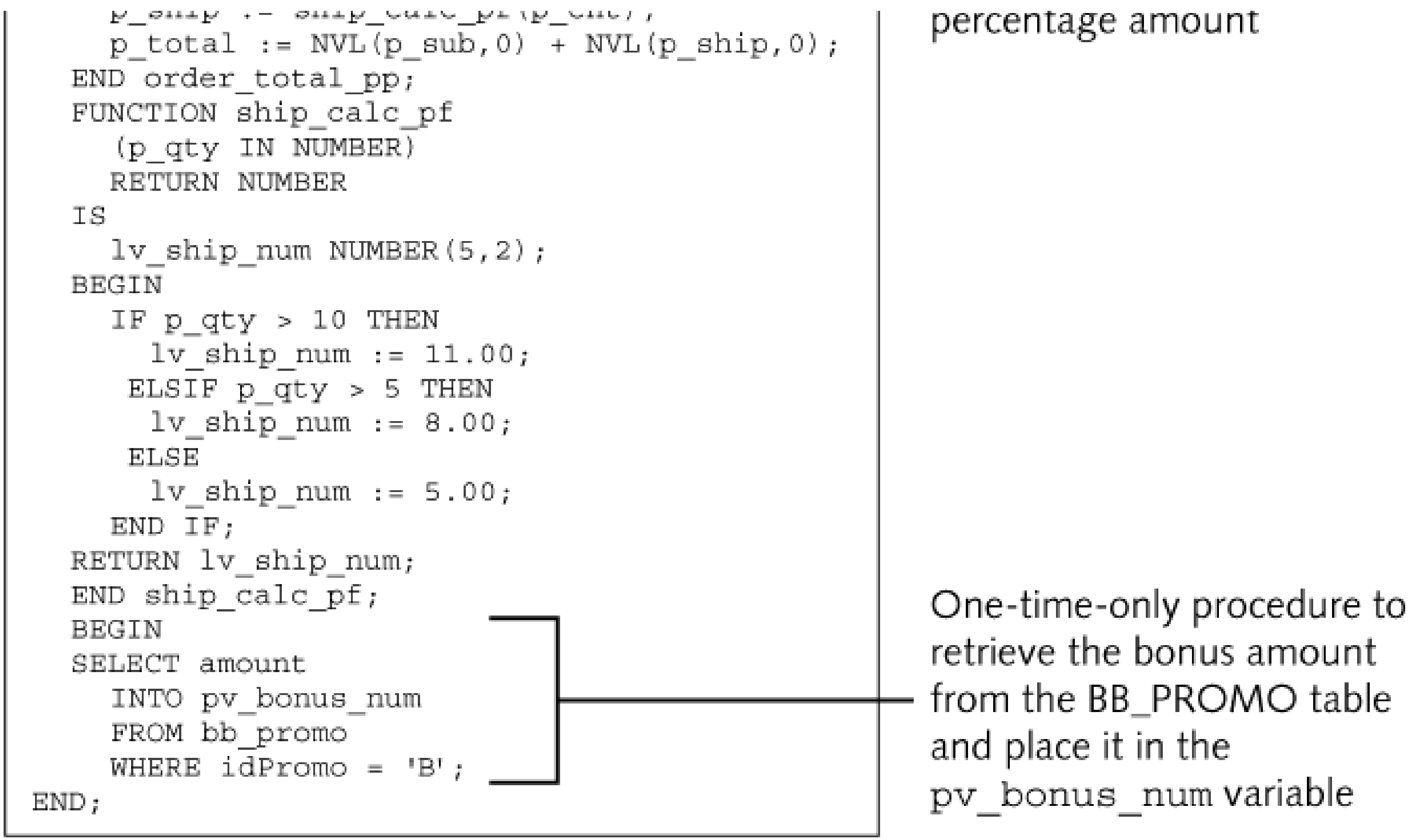
```
Compiled
```

One Time Only Procedure

- Used when user needs a dynamic action to occur on the initial call to a package
- It is an anonymous block placed at the end of a package body (no END statement!)
- Only executes on initial call to the package
- Typically used to populate global constructs

One Time Only Procedure

P
L
/
S
Q
L



```
    p_ship := ship_calc_pf(p_cnt);
    p_total := NVL(p_sub,0) + NVL(p_ship,0);
  END order_total_pp;
  FUNCTION ship_calc_pf
    (p_qty IN NUMBER)
    RETURN NUMBER
  IS
    lv_ship_num NUMBER(5,2);
  BEGIN
    IF p_qty > 10 THEN
      lv_ship_num := 11.00;
     ELSIF p_qty > 5 THEN
      lv_ship_num := 8.00;
     ELSE
      lv_ship_num := 5.00;
    END IF;
  RETURN lv_ship_num;
  END ship_calc_pf;
  BEGIN
  SELECT amount
    INTO pv_bonus_num
    FROM bb_promo
    WHERE idPromo = 'B';
END;
```

percentage amount

One-time-only procedure to retrieve the bonus amount from the BB_PROMO table and place it in the pv_bonus_num variable

Overloading Program Units

PL/SQL

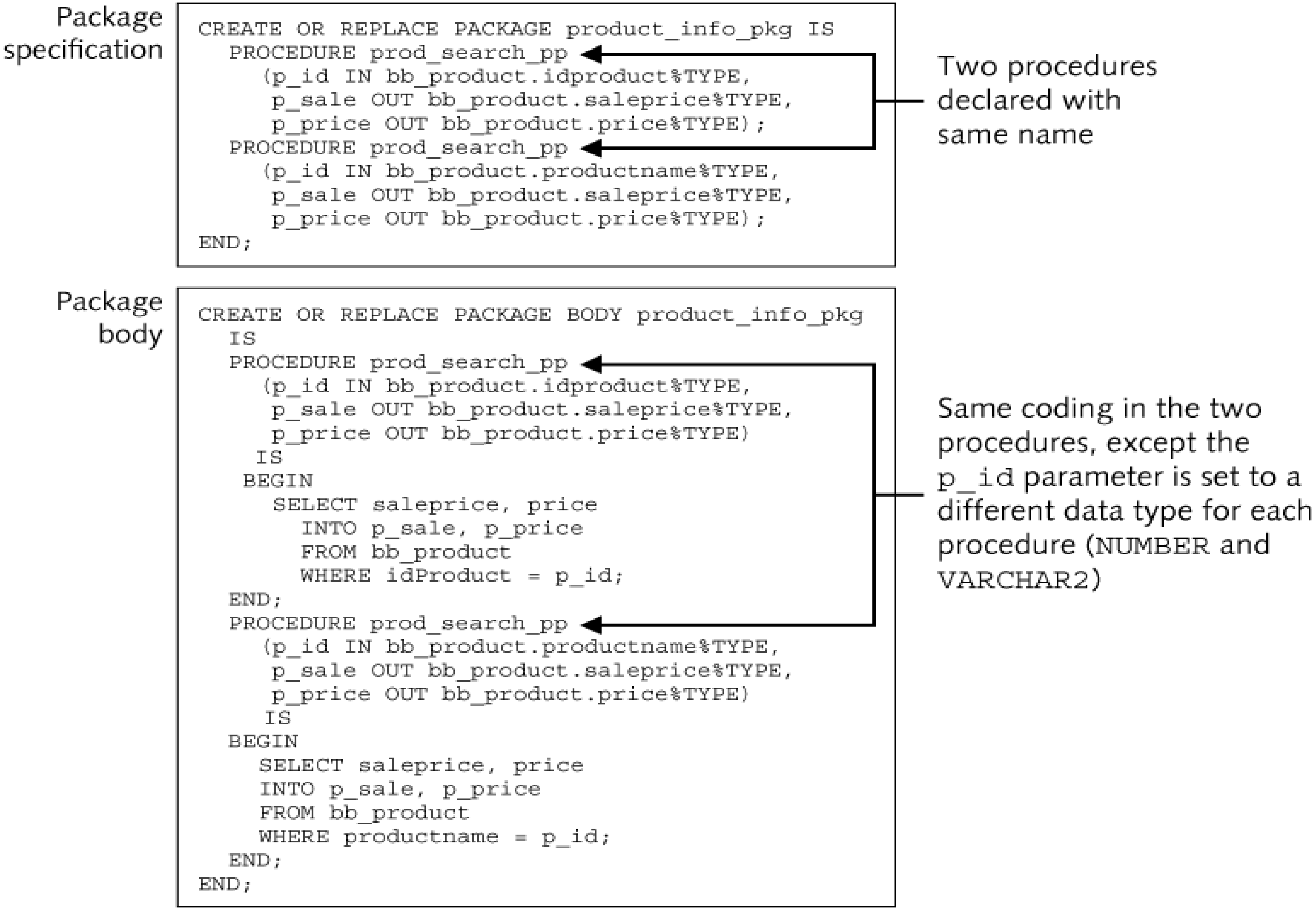
- Overloading is the creation of more than one program unit with the same name
- The program units must differ by at least one of the following:
  - Number of parameters
  - Parameter data type families
  - Listed order

Overloading Program Units (continued)

P
L
/
S
Q
L

- Allows a particular program unit to accept various sets of arguments
- Some Oracle-supplied functions are overloaded, such as TO_CHAR, which can accept various data types as an argument
- Overloading can only be accomplished with a package

Overloading Program Units (continued)



Package specification

```
CREATE OR REPLACE PACKAGE product_info_pkg IS
  PROCEDURE prod_search_pp
    (p_id IN bb_product.idproduct%TYPE,
     p_sale OUT bb_product.saleprice%TYPE,
     p_price OUT bb_product.price%TYPE);
  PROCEDURE prod_search_pp
    (p_id IN bb_product.productname%TYPE,
     p_sale OUT bb_product.saleprice%TYPE,
     p_price OUT bb_product.price%TYPE);
END;
```

Two procedures declared with same name

Package body

```
CREATE OR REPLACE PACKAGE BODY product_info_pkg
  IS
  PROCEDURE prod_search_pp
    (p_id IN bb_product.idproduct%TYPE,
     p_sale OUT bb_product.saleprice%TYPE,
     p_price OUT bb_product.price%TYPE)
    IS
   BEGIN
     SELECT saleprice, price
       INTO p_sale, p_price
       FROM bb_product
       WHERE idProduct = p_id;
  END;
  PROCEDURE prod_search_pp
    (p_id IN bb_product.productname%TYPE,
     p_sale OUT bb_product.saleprice%TYPE,
     p_price OUT bb_product.price%TYPE)
     IS
  BEGIN
    SELECT saleprice, price
    INTO p_sale, p_price
    FROM bb_product
    WHERE productname = p_id;
  END;
END;
```

Same coding in the two procedures, except the `p_id` parameter is set to a different data type for each procedure (`NUMBER` and `VARCHAR2`)

Packaged Function Restrictions

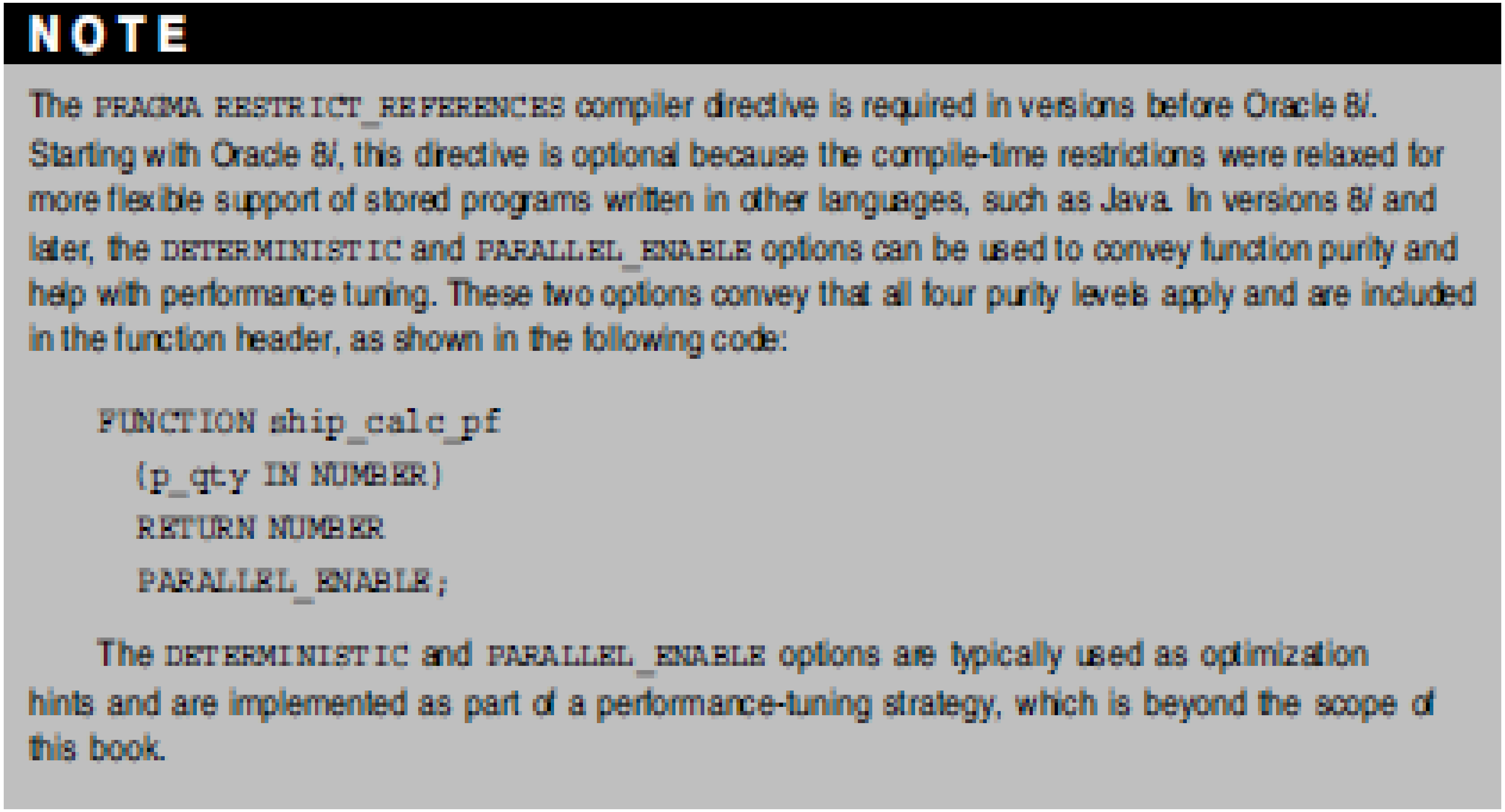
- Function purity level defines what structures the function reads or modifies
- Important to indicate purity level in package specification to discover errors at compile time rather than run time
- Add the following statement in the specification:

PRAGMA RESTRICT_REFERENCES(program_unit_name, purity levels,…)

Purity Levels

Level Acronym	Level Name	Level Description
WNDS	**Writes No Database State**	**Function does not modify any database tables (No DML)**
RNDS	**Reads No Database State**	**Function does not read any tables (No select)**
WNPS	**Writes No Package State**	**Function does not modify any packaged variables (packaged variables are variables declared in a package specification)**
RNPS	**Reads No Package State**	**Function does not read any packaged variables**

Purity Levels



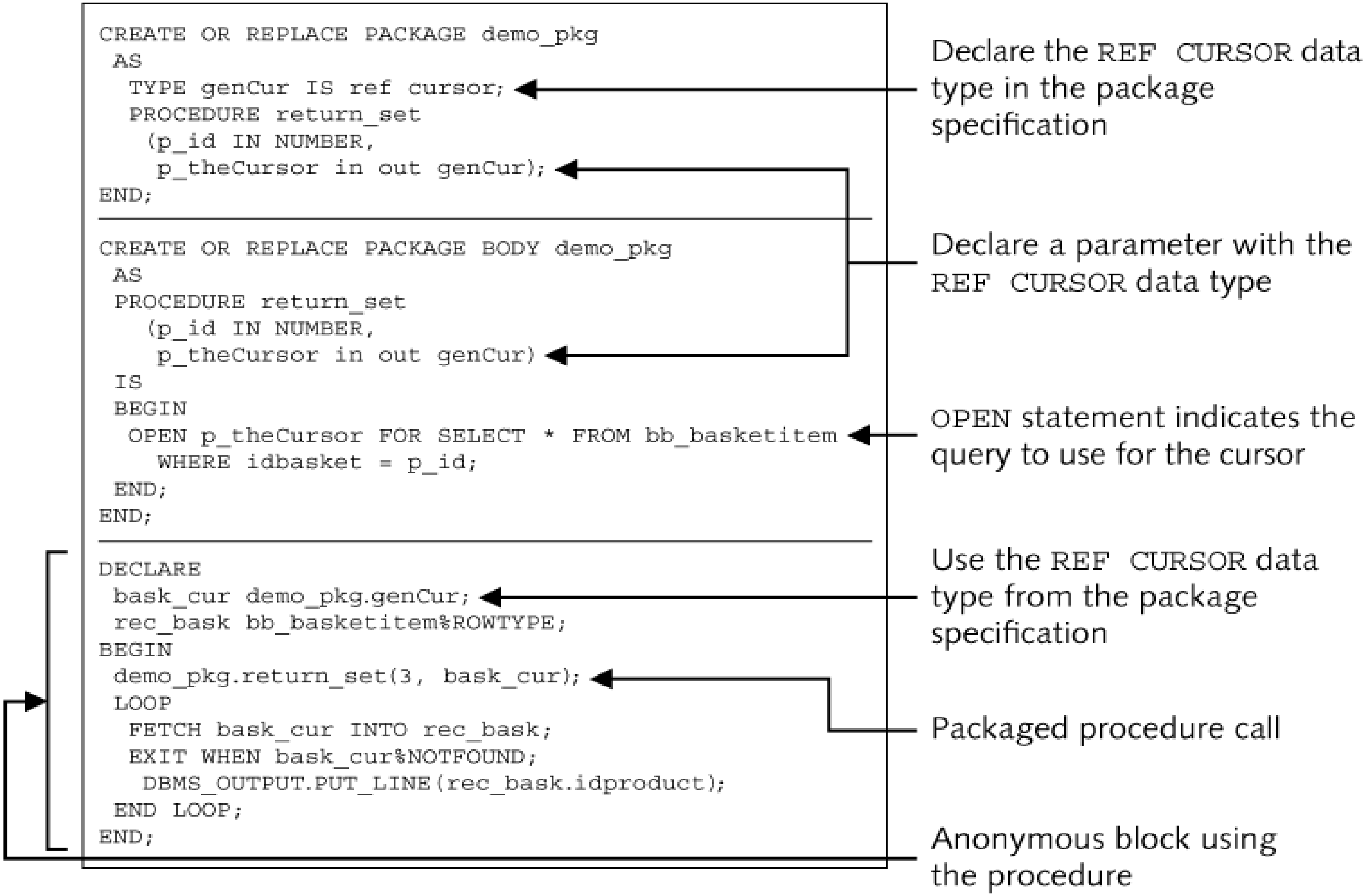
NOTE

The PRAGMA RESTRICT_REFERENCES compiler directive is required in versions before Oracle 8*i*. Starting with Oracle 8*i*, this directive is optional because the compile-time restrictions were relaxed for more flexible support of stored programs written in other languages, such as Java. In versions 8*i* and later, the DETERMINISTIC and PARALLEL_ENABLE options can be used to convey function purity and help with performance tuning. These two options convey that all four purity levels apply and are included in the function header, as shown in the following code:

```
FUNCTION ship_calc_pf
   (p_qty IN NUMBER)
   RETURN NUMBER
   PARALLEL_ENABLE;
```

The DETERMINISTIC and PARALLEL_ENABLE options are typically used as optimization hints and are implemented as part of a performance-tuning strategy, which is beyond the scope of this book.

REF CURSOR Parameter



```
CREATE OR REPLACE PACKAGE demo_pkg
 AS
  TYPE genCur IS ref cursor;
  PROCEDURE return_set
   (p_id IN NUMBER,
    p_theCursor in out genCur);
END;
```

```
CREATE OR REPLACE PACKAGE BODY demo_pkg
 AS
 PROCEDURE return_set
   (p_id IN NUMBER,
    p_theCursor in out genCur)
 IS
 BEGIN
  OPEN p_theCursor FOR SELECT * FROM bb_basketitem
    WHERE idbasket = p_id;
 END;
END;
```

```
DECLARE
 bask_cur demo_pkg.genCur;
 rec_bask bb_basketitem%ROWTYPE;
BEGIN
 demo_pkg.return_set(3, bask_cur);
 LOOP
  FETCH bask_cur INTO rec_bask;
  EXIT WHEN bask_cur%NOTFOUND;
   DBMS_OUTPUT.PUT_LINE(rec_bask.idproduct);
 END LOOP;
END;
```

Declare the REF CURSOR data type in the package specification

Declare a parameter with the REF CURSOR data type

OPEN statement indicates the query to use for the cursor

Use the REF CURSOR data type from the package specification

Packaged procedure call

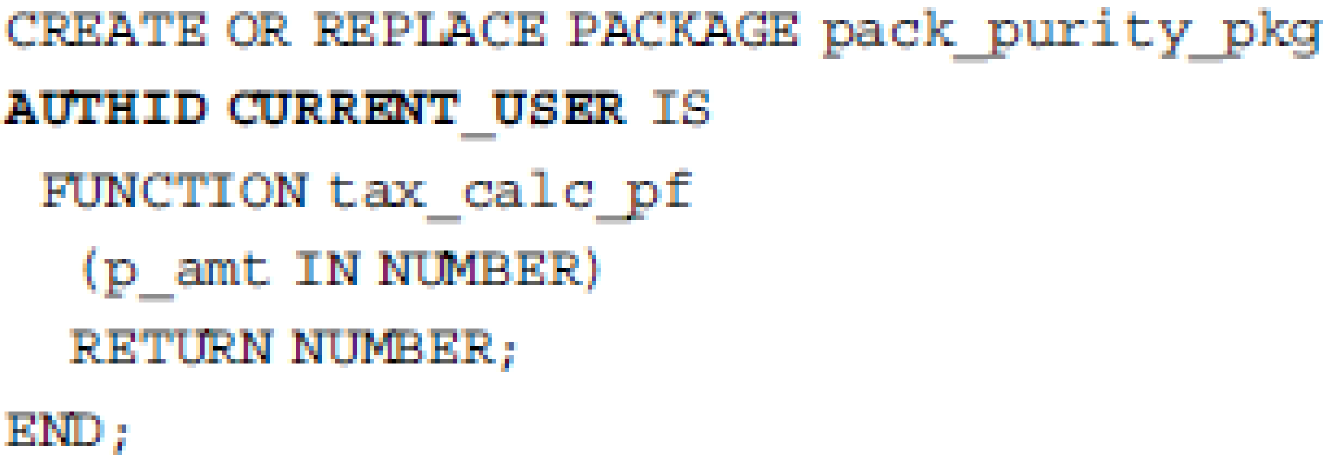
Anonymous block using the procedure

Execute Privileges

- Avoids issuing privileges to all database objects
- If you issue EXECUTE privilege on a package, the user will assume the package owner rights for the period of execution
  - Called definer-rights
- You can override this default by adding AUTHID CURRENT_USER in the specification
- Adds security by avoiding the direct access issue of privileges to database objects

Execute Privileges

P
L
/
S
Q
L

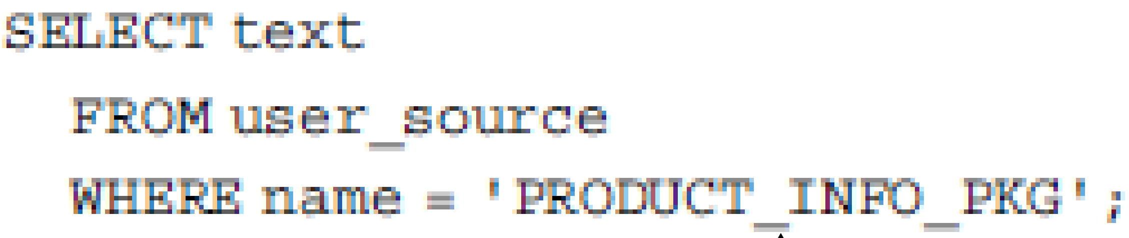


```
CREATE OR REPLACE PACKAGE pack_purity_pkg
AUTHID CURRENT_USER IS
 FUNCTION tax_calc_pf
   (p_amt IN NUMBER)
   RETURN NUMBER;
END;
```

Data Dictionary Information

PL/SQL

- Text column of USER_SOURCE view will display the source code of the entire package – specification and body
- Use a WHERE clause on the name column to select only one package
- The USER_OBJECTS view can be used to determine what packages exist in the database

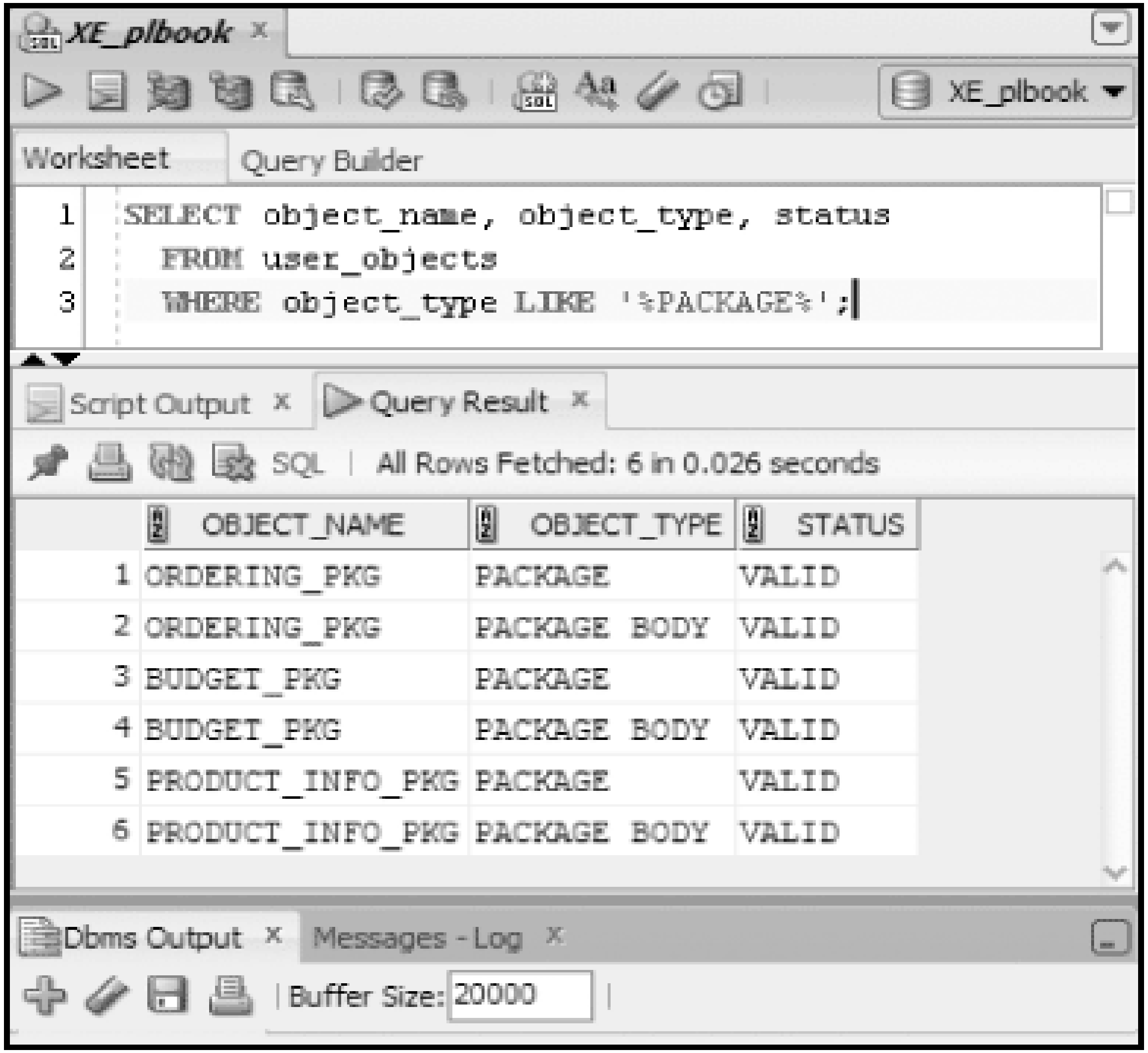


```
SELECT text
  FROM user_source
  WHERE name = 'PRODUCT_INFO_PKG';
```

Upper case

Data Dictionary Information

P
L
/
S
Q
L



```
SELECT object_name, object_type, status
  FROM user_objects
  WHERE object_type LIKE '%PACKAGE%';
```

All Rows Fetched: 6 in 0.026 seconds

	OBJECT_NAME	OBJECT_TYPE	STATUS
1	ORDERING_PKG	PACKAGE	VALID
2	ORDERING_PKG	PACKAGE BODY	VALID
3	BUDGET_PKG	PACKAGE	VALID
4	BUDGET_PKG	PACKAGE BODY	VALID
5	PRODUCT_INFO_PKG	PACKAGE	VALID
6	PRODUCT_INFO_PKG	PACKAGE BODY	VALID

Deleting Packages

- To delete specification and body:

  DROP PACKAGE *package_name*;

- To delete the body only:

  DROP PACKAGE BODY *package_name*;

Summary

PL/SQL

- A package can have two parts: a specification and a body
- Packages allow both public and private constructs
- Global construct values persist
- Forward declaration enables program unit organization
- One time only procedures only execute on the initial call to the package

Summary (continued)

PL/SQL

- Overloading allows program units to accept different sets of arguments
- Address function purity levels
- Granting the **EXECUTE** privilege on a package enables definer-rights
- A REF CURSOR can pass a set of data between program units
- The **USER_SOURCE** data dictionary view is used to retrieve package source code
- The **DROP** statement is used to delete packages