Lab 12. Packages

Databases II

12.1 Packages

A package is a collection of PL/SQL objects grouped under one package name. Packages include procedures, functions, cursors, declarations, types, and variables.
Using packages as a method to bundle your functions and procedures offers numerous benefits:

- A well-designed package is a logical grouping of objects such as functions, procedures, global variables, and cursors. All the code (parse tree and pseudocode [p-code]) is loaded into memory (the Shared Global Area [SGA] of the Oracle Server) on the package's first call. Therefore, packages are often used in applications that use procedures and functions repeatedly.
- Packages allow you to incorporate some of the concepts involved in object-oriented programming, even though PL/SQL is not a "true" object-oriented programming language. Because all the package code has been loaded into memory, you can also write your code so that similar code fragments are placed in the package in a manner that allows multiple procedures and functions to call them.

The package's structure

1. Package specification - contains information about the package's contents, but not the code for the procedures and functions. It also contains declarations of global/public variables. Anything placed in the declaration section of a PL/SQL block may be coded in a package specification.

   The syntax for the package specification is as follows :

   ```
   PACKAGE package_name
        IS
        [declarations of variables and types]
        [specifications of cursors]
        [specifications of modules]
        END [package_name];
   ```

   Example:

   ```
   CREATE OR REPLACE PACKAGE  school_api as
      v_current_date DATE;
      PROCEDURE Discount_Cost;
      FUNCTION new_instructor_id RETURN instructor.instructor_id%TYPE;
   END school_api;
   ```

2. Package body - contains the actual executable code for the objects described in the package specification. The package body contains code for all procedures and functions described in the specification. It also may contain code for objects not declared in the specification.

   The syntax for the package body is as follows:

   ```
   PACKAGE BODY package_name
        IS
        [declarations of variables and types]
        [specification and SELECT statement of cursors]
        [specification and body of modules]
        [BEGIN
        executable statements]
    [EXCEPTION
   ```

```
             exception handlers]
             END [package_name];
```

Example

```
CREATE OR REPLACE PACKAGE BODY school_api AS
  PROCEDURE discount_cost IS
   CURSOR c_group_discount IS
     SELECT distinct s.course_no, c.description
       FROM section s, enrollment e, course c
      WHERE s.section_id = e.section_id
      GROUP BY s.course_no, c.description, e.section_id, s.section_id
      HAVING COUNT(*) >=8;
 BEGIN
   FOR r_group_discount IN c_group_discount LOOP
     UPDATE course
       SET cost = cost * .95
       WHERE course_no = r_group_discount.course_no;
     DBMS_OUTPUT.PUT_LINE('A 5% discount has been given to'
       ||r_group_discount.course_no||''||r_group_discount.description);
   END LOOP;
 END discount_cost;
 FUNCTION new_instructor_id RETURN instructor.instructor_id%TYPE IS
   v_new_instid instructor.instructor_id%TYPE;
 BEGIN
   SELECT INSTRUCTOR_ID_SEQ.NEXTVAL
     INTO v_new_instid
     FROM dual;
   RETURN v_new_instid;
 EXCEPTION
   WHEN OTHERS THEN
     DBMS_OUTPUT.PUT_LINE('Error!');
END new_instructor_id;
BEGIN
SELECT trunc(sysdate, 'DD')
  INTO v_current_date
  FROM dual;
END school_api;
```

RULES FOR THE PACKAGE BODY

You must follow several rules in the package body code:

- There must be an exact match between the cursor and module headers and their definitions in the package specification.

- Do not repeat in the body the declaration of variables, exceptions, types, or constants in the specification.
- Any element declared in the specification can be referenced in the body.

REFERENCING PACKAGE ELEMENTS

Use the following syntax when calling packaged elements from outside the package:

```
package_name.element
```

You do not need to qualify elements when they are declared and referenced inside the body of the package or when they are declared in a specification and referenced inside the body of the same package.

```
SET SERVEROUTPUT ON
DECLARE
  V_instructor_id instructor.instructor_id%TYPE;
BEGIN
  School_api.Discount_Cost;
  v_instructor_id := school_api.new_instructor_id;
  DBMS_OUTPUT.PUT_LINE('The new id is: '||v_instructor_id);
END;
```

Lab work:

1. Add a procedure to the school_api package called remove_student. This procedure accepts a student_id and returns nothing. Based on the student ID passed in, it removes the student from the database. If the student does not exist or if a problem occurs while removing the student (such as a foreign key constraint violation), let the calling program handle it.

2. Alter remove_student in the school_api package body to accept an additional parameter. This new parameter should be a VARCHAR2 and should be called p_ri. Make p_ri default to R. The new parameter may contain a value of R or C. If R is received, it represents DELETE RESTRICT, and the procedure acts as it does now. If there are enrollments for the student, the delete is disallowed. If a C is received, it represents DELETE CASCADE. This functionally means that the remove_student procedure locates all records for the student in all the Student Database tables. It removes them from the database before attempting to remove the student from the student table. Decide how to handle the situation when the user passes in a code other than CorR.