12 Working with Packages

✅Use package types and variables

Package example: 10.9

```
-- declare a public type, cursor, and variable
CREATE OR REPLACE PACKAGE mypack AUTHID DEFINER IS
    TYPE emprectyp IS RECORD (id NUMBER, last_name VARCHAR2(50));
    CURSOR mycur RETURN emprectyp;
    v_num NUMBER;
END mypack;
/
CREATE OR REPLACE PACKAGE BODY mypack AS
    CURSOR mycur RETURN emprectyp IS
    SELECT id, last_name FROM emp;
BEGIN
    v_num := 8;
END mypack;
/
-- use public variable and type
DECLARE
    myrec mypack.emprectyp;
BEGIN
    SELECT id, last_name INTO myrec FROM emp WHERE id = mypack.v_num;
    DBMS_OUTPUT.PUT_LINE(myrec.id || myrec.last_name);
END;
/
```

✅Use packaged constants and functions in SQL

STANDARD package: 10.10

- You can use a packaged function in SQL simply by calling the function in a `SELECT` statement with its qualified package name: `SELECT package_name.function_name FROM table_name;`
- You can't use packaged constants directly in SQL, but you can create a wrapper function (getter) for them and use that function in SQL to refer to the constant

How STANDARD package defines the PL/SQL environment

- A package named `STANDARD` defines the PL/SQL environment
- The package specification declares public types, variables, exceptions, subprograms, which are available automatically to PL/SQL programs
- For example, package `STANDARD` declares function `ABS`, which returns the absolute value of its argument, as follows: `FUNCTION ABS (n NUMBER) RETURN NUMBER;`

- The contents of package `STANDARD` are directly visible to applications. You need not qualify references to its contents by prefixing the package name
- For example, you might invoke `ABS` from a database trigger, stored subprogram, Oracle tool, or 3GL application, as follows: `abs_diff := ABS(x - y);`
- If you declare your own version of `ABS`, your local declaration overrides the public declaration. You can still invoke the SQL function by specifying its full name: `abs_diff := STANDARD.ABS(x - y);`
- Most SQL functions are overloaded. For example, package `STANDARD` contains these declarations:

```
FUNCTION TO_CHAR (right DATE) RETURN VARCHAR2;
FUNCTION TO_CHAR (left NUMBER) RETURN VARCHAR2;
FUNCTION TO_CHAR (left DATE, right VARCHAR2) RETURN VARCHAR2;
FUNCTION TO_CHAR (left NUMBER, right VARCHAR2) RETURN VARCHAR2;
```

- PL/SQL resolves an invocation of `TO_CHAR` by matching the number and data types of the formal and actual parameters

✅ Use ACCESSIBLE BY to restrict access to package subprograms

ACCESSIBLE BY: 10.1

- The **ACCESSIBLE BY clause** of the package specification lets you specify a white list of PL/SQL units that can access the package. You use this clause in situations like these:
  - You implement a PL/SQL application as several packages—one package that provides the application programming interface (API) and helper packages to do the work
    - You want clients to have access to the API, but not to the helper packages. Therefore, you omit the `ACCESSIBLE BY` clause from the API package specification and include it in each helper package specification, where you specify that only the API package can access the helper package
  - You create a utility package to provide services to some, but not all, PL/SQL units in the same schema
    - To restrict use of the package to the intended units, you list them in the `ACCESSIBLE BY` clause in the package specification