12

Understanding and Influencing the PL/SQL Compiler

Objectives

After completing this lesson, you should be able to do the following:

- Describe native and interpreted compilations
- List the features of native compilation
- Switch between native and interpreted compilations
- Set parameters that influence PL/SQL compilation
- Query data dictionary views on how PL/SQL code is compiled
- Use the compiler warning mechanism and the `DBMS_WARNING` package to implement compiler warnings

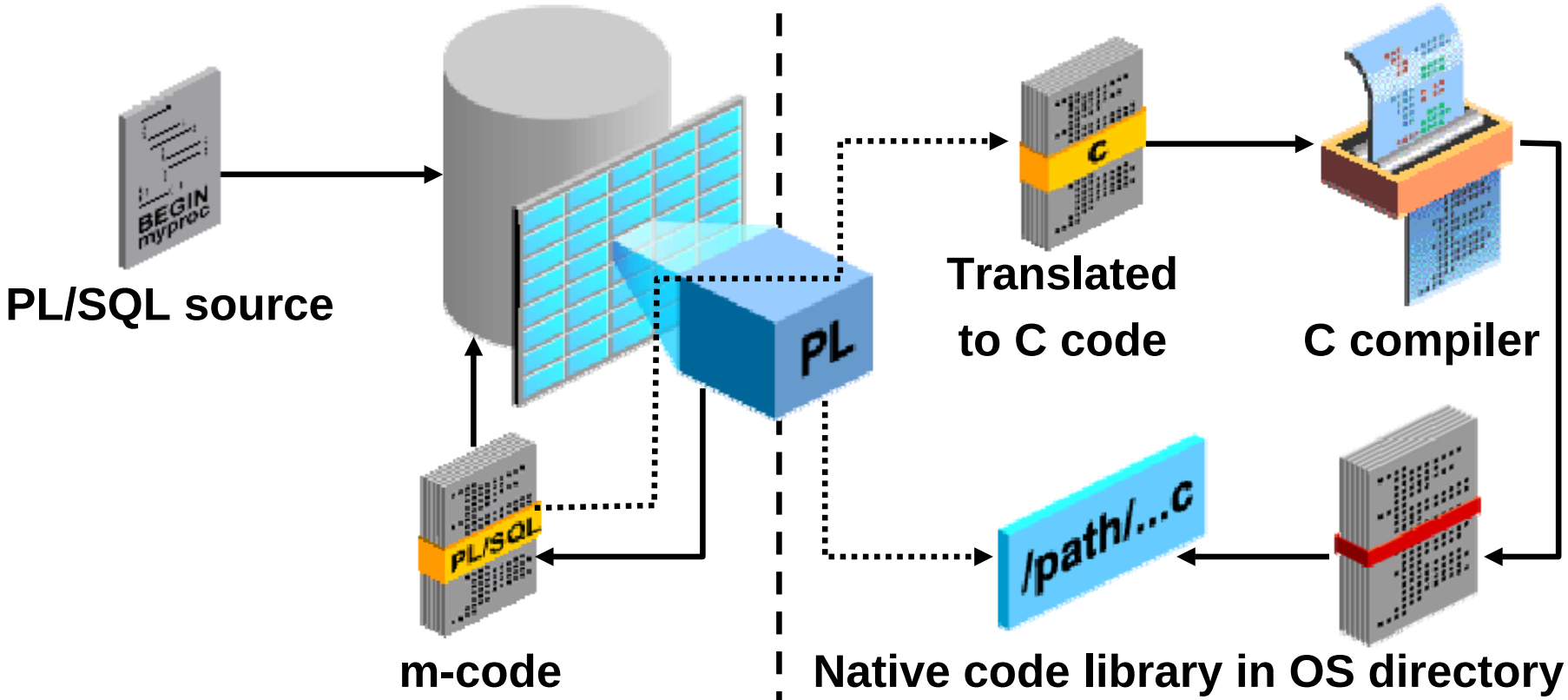
Native and Interpreted Compilation

Interpreted code

- Compiled to m-code
- Stored in the database

Natively compiled code

- Translated C and compiled
- Copied to a code library



Features and Benefits of Native Compilation

Native compilation:

- Uses a generic `makefile` that uses the following operating system software:
  - C compiler
  - Linker
  - Make utility
- Generates shared libraries that are copied to the file system and loaded at run time
- Provides better performance (up to 30% faster than interpreted code) for computation-intensive procedural operations

Considerations When Using Native Compilation

Consider the following:

- Debugging tools for PL/SQL cannot debug natively compiled code.
- Natively compiled code is slower to compile than interpreted code.
- Large amounts of natively compiled subprograms can affect performance due to operating system–imposed limitations when handling shared libraries. OS directory limitations can be managed by setting database initialization parameters:
  - `PLSQL_NATIVE_LIBRARY_SUBDIR_COUNT and`
  - `PLSQL_NATIVE_LIBRARY_DIR`

Parameters Influencing Compilation

- System parameters are set in the `init`*`SID`*`.ora` file or by using the `SPFILE`:

```
PLSQL_NATIVE_LIBRARY_DIR = full-directory-path-name
PLSQL_NATIVE_LIBRARY_SUBDIR_COUNT = count
```

- System or session parameters:

```
PLSQL_COMPILER_FLAGS = 'NATIVE' or 'INTERPRETED'
```

Switching Between Native and Interpreted Compilation

- Setting native compilation:
  - For the system:

```
ALTER SYSTEM SET plsql_compiler_flags='NATIVE';
```

  - For the session:

```
ALTER SESSION SET plsql_compiler_flags='NATIVE';
```

- Setting interpreted compilation:
  - For the system level:

```
ALTER SYSTEM
        SET plsql_compiler_flags='INTERPRETED';
```

  - For the session:

```
ALTER SESSION
        SET plsql_compiler_flags='INTERPRETED';
```

Viewing Compilation Information in the Data Dictionary

Query information in the following views:

- USER_STORED_SETTINGS
- USER_PLSQL_OBJECTS

Example:

```
SELECT param_value
FROM   user_stored_settings
WHERE  param_name = 'plsql_compiler_flags'
  AND  object_name = 'GET_EMPLOYEES';
```

Note: The PARAM_VALUE column has a value of NATIVE for procedures that are compiled for native execution; otherwise, it has a value of INTERPRETED.

Using Native Compilation

To enable native compilation, perform the following steps:

1. Edit the supplied `makefile` and enter appropriate paths and other values for your system.
2. Set the `PLSQL_COMPILER_FLAGS` parameter (at system or session level) to the value `NATIVE`. The default is `INTERPRETED`.
3. Compile the procedures, functions, and packages.
4. Query the data dictionary to see that a procedure is compiled for native execution.

Compiler Warning Infrastructure

The PL/SQL compiler in Oracle Database 10*g* has been enhanced to produce warnings for subprograms. Warning levels:

- Can be set:
  - Declaratively with the `PLSQL_WARNINGS` initialization parameter
  - Programmatically using the `DBMS_WARNINGS` package
- Are arranged in three categories: severe, performance, and informational
- Can be enabled and disabled by category or a specific message

Examples of warning messages:

SP2-0804: Procedure created with compilation warnings

PLW-07203: The `'IO_TBL'` parameter may benefit from use of the `NOCOPY` compiler hint.

Setting Compiler Warning Levels

Set the PLSQL_WARNINGS initialization parameter to enable the database to issue warning messages.

```
ALTER SESSION SET PLSQL_WARNINGS = 'ENABLE:SEVERE',
                  'DISABLE:INFORMATIONAL';
```

- The PLSQL_WARNINGS combine a qualifier value (ENABLE, DISABLE, or ERROR) with a comma-separated list of message numbers, or with one of the following modifier values:
  - ALL, SEVERE, INFORMATIONAL, or PERFORMANCE
- Warning messages use a PLW prefix.

  PLW-07203: The 'IO_TBL' parameter may benefit from use of the NOCOPY compiler hint.

Guidelines for Using PLSQL_WARNINGS

The PLSQL_WARNINGS setting:

- Can be set to DEFERRED at the system level
- Is stored with each compiled subprogram
- That is current for the session is used by default when recompiling with:
  - A CREATE OR REPLACE statement
  - An ALTER...COMPILE statement
- That is stored with the compiled subprogram is used when REUSE SETTINGS is specified when recompiling with an ALTER...COMPILE statement

DBMS_WARNING Package

The DBMS_WARNING package provides a way to programmatically manipulate the behavior of the current system or session PL/SQL warning settings. Using DBMS_WARNING subprograms, you can:

- Query existing settings
- Modify the settings for specific requirements or restore original settings
- Delete the settings

Example: Saving and restoring warning settings for a development environment that calls your code that compiles PL/SQL subprograms and suppresses warnings due to business requirements

Using DBMS_WARNING Procedures

Package procedures change PL/SQL warnings:

```
ADD_WARNING_SETTING_CAT(w_category,w_value,scope)
ADD_WARNING_SETTING_NUM(w_number,w_value,scope)
SET_WARNING_SETTING_STRING(w_value, scope)
```

- All parameters are IN parameters and have the VARCHAR2 data type. However, the w_number parameter is a NUMBER data type.
- Parameter string values are not case-sensitive.
- The w_value parameters values are ENABLE, DISABLE, and ERROR.
- The w_category values are ALL, INFORMATIONAL, SEVERE, and PERFORMANCE.
- The scope value is either SESSION or SYSTEM. Using SYSTEM requires the ALTER SYSTEM privilege.

Using DBMS_WARNING Functions

Package functions read PL/SQL warnings:

```
GET_CATEGORY(w_number) RETURN VARCHAR2
GET_WARNING_SETTING_CAT(w_category)RETURN VARCHAR2
GET_WARNING_SETTING_NUM(w_number) RETURN VARCHAR2
GET_WARNING_SETTING_STRING RETURN VARCHAR2
```

- GET_CATEGORY returns a value of ALL, INFORMATIONAL, SEVERE, or PERFORMANCE for a given message number.
- GET_WARNING_SETTING_CAT returns ENABLE, DISABLE, or ERROR as the current warning value for a category name, and GET_WARNING_SETTING_NUM returns the value for a specific message number.
- GET_WARNING_SETTING_STRING returns the entire warning string for the current session.

Using DBMS_WARNING: Example

Consider the following scenario:
Save current warning settings, disable warnings for the PERFORMANCE category, compile a PL/SQL package, and restore the original warning setting.

```
CREATE PROCEDURE compile(pkg_name VARCHAR2) IS
  warn_value VARCHAR2(200);
  compile_stmt VARCHAR2(200) :=
    'ALTER PACKAGE '|| pkg_name ||' COMPILE';
BEGIN
  warn_value :=   -- Save current settings
      DBMS_WARNING.GET_WARNING_SETTING_STRING;
  DBMS_WARNING.ADD_WARNING_SETTING_CAT( -- change
      'PERFORMANCE', 'DISABLE', 'SESSION');
  EXECUTE IMMEDIATE compile_stmt;
  DBMS_WARNING.SET_WARNING_SETTING_STRING(--restore
      warn_value, 'SESSION');
END;
```

Using DBMS_WARNING: Example

To test the `compile` procedure, you can use the following script sequence:

```
DECLARE
  PROCEDURE print(s VARCHAR2) IS
  BEGIN
    DBMS_OUTPUT.PUT_LINE(s);
  END;
BEGIN
  print('Warning settings before: '||
        DBMS_WARNING.GET_WARNING_SETTING_STRING);
  compile('my_package');
  print('Warning settings after: '||
        DBMS_WARNING.GET_WARNING_SETTING_STRING);
END;
/
SHOW ERRORS PACKAGE MY_PACKAGE
```

Summary

In this lesson, you should have learned how to:

- Switch between native and interpreted compilations
- Set parameters that influence native compilation of PL/SQL programs
- Query data dictionary views that provide information about PL/SQL compilation settings
- Use the PL/SQL compiler warning mechanism:
  - Declaratively by setting the `PLSQL_WARNINGS` parameter
  - Programmatically using the `DBMS_WARNING` package

Practice 12: Overview

This practice covers the following topics:

- Enabling native compilation for your session and compiling a procedure
- Creating a subprogram to compile a PL/SQL procedure, function, or a package; suppressing warnings for the `PERFORMANCE` compiler warning category; and restoring the original session warning settings
- Executing the procedure to compile a PL/SQL package containing a procedure that uses a PL/SQL table as an `IN OUT` parameter without specifying the `NOCOPY` hint