

ORACLE[®]

ACADEMY

Database Programming with PL/SQL

15-2

Displaying Compiler Warning Messages



Objectives

This lesson covers the following objectives:

- Explain the similarities and differences between a warning and an error
- Compare and contrast the warning levels which can be set by the `PLSQL_WARNINGS` parameter
- Set warning levels by calling the `DBMS_WARNING` server-supplied package from within a PL/SQL program

Purpose

- Imagine that you and your family have just moved to live in an unfamiliar city.
- You will study at a new school, traveling there and back by public bus.
- You look up possible bus routes, and find routes 24 and 67 will take you from home to school, and back again.
- You choose route 24, board the bus on the first morning, and find that it takes 40 minutes to reach your school.

Purpose

- Later, you find that route 67 takes only 15 minutes.
- What if someone had told you in advance that route 67 is faster?
- You wouldn't have wasted your time.

Errors and Warnings

- Routes 24 and 67 both execute successfully (they take you to school), but one is better than the other.
- A third route (48) passes your house, but would not take you anywhere near your school.
- We could say that taking route 48 causes an error, because it doesn't work.
- Route 24 won't cause an error (it does take you to school), but be warned, it is not the best route.



Errors and Warnings

- What is wrong with this PL/SQL code?

```
CREATE OR REPLACE PROCEDURE count_emps IS
  v_count  PLS_INTEGER;
BEGIN
  SELECT COUNT(*) INTO v_count FROM countries;
  DBMS_OUTPUT.PUT_LINE(v_counter);
END;
```

- Clearly, V_COUNTER is not declared.
- The PL/SQL compiler detects the error and the procedure is not compiled.
- This is like bus route 48, it doesn't work!

```
Error at line 4: PLS-00201: identifier 'V_COUNTER' must be declared
2.  v_count  PLS_INTEGER;
3.  BEGIN
4.  SELECT COUNT(*) INTO v_counter FROM countries;
5.  DBMS_OUTPUT.PUT_LINE(v_count);
6.  END;
```

Errors and Warnings

- What is wrong with this PL/SQL code?

```
CREATE OR REPLACE PROCEDURE unreachable IS
  c_const CONSTANT BOOLEAN := TRUE;
BEGIN
  IF c_const THEN
    DBMS_OUTPUT.PUT_LINE('TRUE');
  ELSE
    DBMS_OUTPUT.PUT_LINE('NOT TRUE');
  END IF;
END unreachable;
```



Errors and Warnings

- The procedure will compile without errors, but the `ELSE` branch can never be executed.
- This is like bus route 24; it will compile successfully, but could be coded better.
- Shouldn't the PL/SQL compiler warn you about this? It can!

```
CREATE OR REPLACE PROCEDURE unreachable IS
  c_const CONSTANT BOOLEAN := TRUE;
BEGIN
  IF c_const THEN
    DBMS_OUTPUT.PUT_LINE('TRUE');
  ELSE
    DBMS_OUTPUT.PUT_LINE('NOT TRUE');
  END IF;
END unreachable;
```

Two PL/SQL Initialization Parameters

```
ALTER SESSION SET PLSQL_WARNINGS = 'ENABLE:ALL';
CREATE OR REPLACE PROCEDURE unreachable IS
  c_const CONSTANT BOOLEAN := TRUE;
BEGIN
  IF c_const THEN
    DBMS_OUTPUT.PUT_LINE('TRUE');
  ELSE
    DBMS_OUTPUT.PUT_LINE('NOT TRUE');
  END IF;
END unreachable;
```

Procedure created.

```
SELECT line, position, text, attribute FROM USER_ERRORS
WHERE name = 'UNREACHABLE';
```

LINE	POSITION	TEXT	ATTRIBUTE
4	6	PLW-06002: Unreachable code	WARNING

PL/SQL Compiler Warnings

- In PL/SQL, an *error* occurs when the code cannot be compiled successfully.
- Application Express automatically displays the first error message; you can see all the errors by querying the Data Dictionary view, `USER_ERRORS`.
- A *warning* occurs when the code compiles successfully, but it could be coded better.
- By default, the PL/SQL compiler does not produce warning messages, but you can tell the compiler to produce them, and also the types (categories) of warnings that you want.

Categories of PL/SQL Warning Messages

- There are three categories of warning messages:
 - SEVERE: code that can cause unexpected behavior or wrong results when executed
 - PERFORMANCE: code that can cause execution speed to be slow
 - INFORMATIONAL: other poor coding practices (for example, code that can never be executed)
- The keyword ALL is shorthand for all three categories.



Enabling PL/SQL Compiler Warnings

- There are two ways to enable compiler warning categories:
 - Using the initialization parameter `PLSQL_WARNINGS`
 - Using the `DBMS_WARNING` server-supplied package
 - Application Express does not automatically display any warning messages; you must `SELECT` them from `USER_ERRORS` after compiling your program.
- Therefore, you can see warnings only for named subprograms, not for anonymous blocks.

Using PLSQL_WARNINGS

- You must set the value of the PLSQL_WARNINGS initialization parameter to one or more comma-separated strings.
- Each string enables or disables a category of warning messages.



Using PLSQL_WARNINGS

Examples:

- This parameter enables the `PERFORMANCE` category, leaving other categories unchanged.

```
ALTER SESSION SET PLSQL_WARNINGS = 'ENABLE:PERFORMANCE';
```

- The first parameter enables all three categories, and the second disables the `SEVERE` category, leaving the `PERFORMANCE` and `INFORMATIONAL` categories enabled.

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

Using PLSQL_WARNINGS

Which categories will be enabled after each of the following statements are executed in the same database session?

```
ALTER SESSION SET PLSQL_WARNINGS = 'DISABLE:ALL';  
...  
ALTER SESSION SET PLSQL_WARNINGS = 'ENABLE:PERFORMANCE';  
...  
ALTER SESSION SET PLSQL_WARNINGS = 'ENABLE:SEVERE';  
...  
ALTER SESSION SET PLSQL_WARNINGS =  
    'ENABLE:ALL', 'DISABLE:SEVERE';  
...  
ALTER SESSION SET PLSQL_WARNINGS =  
    'DISABLE:SEVERE', 'ENABLE:ALL';
```


Using PLSQL_WARNINGS Example

- Look at this code.
- It will compile without errors, but could be better.

```
CREATE OR REPLACE PACKAGE bigarg IS
  TYPE emptab_type IS TABLE OF employees%ROWTYPE
                        INDEX BY PLS_INTEGER;
  PROCEDURE getallemps
    (p_emptab OUT emptab_type);
END bigarg;
```

- Remember the NOCOPY hint?
- It passes large OUT and IN OUT arguments by reference instead of by value, which is faster.
- Let's get the compiler to warn us about this.

Using PLSQL_WARNINGS Example

```
ALTER SESSION
  SET PLSQL_WARNINGS = 'ENABLE:PERFORMANCE';

CREATE OR REPLACE PACKAGE bigarg IS
  TYPE emptab_type IS TABLE OF employees%ROWTYPE
    INDEX BY PLS_INTEGER;

  PROCEDURE getallemps
    (p_emptab OUT emptab_type);
END bigarg;
```

Package created.

```
SELECT line, position, text, attribute FROM USER_ERRORS
  WHERE name = 'BIGARG';
```

LINE	POSITION	TEXT	ATTRIBUTE
5	6	PLW-07203: parameter 'P_EMPTAB' may benefit from use of the NOCOPY compiler hint	WARNING

Using PLSQL_WARNINGS: A Second Example

Have you noticed that warning codes start with PLW-, while error codes start with PLS-?

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

CREATE OR REPLACE FUNCTION noreturn (p_in IN NUMBER) RETURN NUMBER
IS v_bool BOOLEAN;
BEGIN
  IF p_in < 10 THEN v_bool := TRUE;
  ELSE v_bool := FALSE;
  END IF;
END noreturn;

SELECT line, position, text, attribute
FROM USER_ERRORS WHERE name = 'NORETURN';
```

LINE	POSITION	TEXT	ATTRIBUTE
1	1	PLW-05005: function NORETURN returns without value at line 8	WARNING

Treating Warnings as Errors

We can tell the compiler to treat specific warnings as errors and not compile the program:

```
ALTER SESSION SET PLSQL_WARNINGS =
  'ENABLE:SEVERE', 'ERROR:05005';

CREATE OR REPLACE FUNCTION noreturn
  (p_in IN NUMBER) RETURN NUMBER IS
  v_bool BOOLEAN;
BEGIN
  IF p_in < 10 THEN v_bool := TRUE;
  ELSE v_bool := FALSE;
  END IF;
END noreturn;
```

```
Error at line 1: PLS-05005: function NORETURN returns without value at line 8
1. CREATE OR REPLACE FUNCTION noreturn
2.   (p_in IN NUMBER) RETURN NUMBER IS
3.   v_bool BOOLEAN;
```

Using DBMS_WARNING

- You can also set and change warning categories using the DBMS_WARNING server-supplied package.
- This allows you to set the same warning categories as the PLSQL_WARNINGS parameter, but also allows you to save your previous warning settings in a PL/SQL variable.
- This is useful if you want to change your settings, compile some PL/SQL programs, then change settings back to what they were at the beginning.



Using DBMS_WARNING

DBMS_WARNING contains three types of subprograms - SET_* , ADD_* and GET_* .

- SET_* procedures replace all previous warning settings with the new settings.
- ADD_* procedures change only the specified warning settings, leaving the others unaltered. These have the same effect as the PLSQL_WARNINGS initialization parameter.
- GET_* functions don't change any settings; they store the current settings in a PL/SQL variable.

Using DBMS_WARNING.ADD_*

- Here is an ADD_* procedure:

```
BEGIN
  DBMS_WARNING.ADD_WARNING_SETTING_CAT
    ( 'PERFORMANCE' , 'ENABLE' , 'SESSION' );
END;
```

- This enables the PERFORMANCE warning category, leaving other category settings unchanged.
- The third argument, 'SESSION', is required.
- This has exactly the same effect as:

```
ALTER SESSION
  SET PLSQL_WARNINGS = 'ENABLE:PERFORMANCE';
```

Using DBMS_WARNING.SET_*

- Here is the SET_* procedure:

```
BEGIN
  DBMS_WARNING.SET_WARNING_SETTING_STRING
    ( 'ENABLE:SEVERE' , 'SESSION' );
END;
```

- This disables all warning categories, then enables the SEVERE category.
- This has exactly the same effect as:

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


Using DBMS_WARNING.GET_*

- Here is a GET_* function:

```
DECLARE
  v_string  VARCHAR2(200);
BEGIN
  v_string :=
    DBMS_WARNING.GET_WARNING_SETTING_STRING;
  DBMS_OUTPUT.PUT_LINE(v_string);
END;
```

- This returns all the current warning settings into a VARCHAR2 variable, whose value is then displayed:

```
DISABLE: INFORMATIONAL,DISABLE: PERFORMANCE,ENABLE: SEVERE
Statement processed.
```

Using DBMS_WARNING.GET_*

- Here is another GET_* function:

```
DECLARE
  v_string  VARCHAR2(200);
BEGIN
  v_string :=
    DBMS_WARNING.GET_CATEGORY(7203);
  DBMS_OUTPUT.PUT_LINE(v_string);
END;
```

```
PERFORMANCE
```

```
Statement processed.
```

- This returns the warning category of a PLW- warning number: PLW-07203 is in the PERFORMANCE category.

Using DBMS_WARNING.GET_*

- Of course, we can call DBMS_WARNING.GET_* directly from DBMS_OUTPUT.PUT_LINE:

```
BEGIN
  DBMS_OUTPUT.PUT_LINE
    (DBMS_WARNING.GET_WARNING_SETTING_STRING);
END;
```

```
DISABLE: INFORMATIONAL,DISABLE: PERFORMANCE,ENABLE: SEVERE
Statement processed.
```

- There are several more subprograms in DBMS_WARNING, but the ones you have seen are the most useful.

Using GET_* and SET*_ to Save and Restore Warning Settings

We can save our current warning settings, change them to compile a specific PL/SQL program, and then restore our original settings correctly (even if we don't remember what they were):

```
DECLARE
  v_settings VARCHAR2(200);
BEGIN
  v_settings := DBMS_WARNING.GET_WARNING_SETTING_STRING;
  DBMS_WARNING.SET_WARNING_SETTING_STRING
    ('ENABLE:SEVERE', 'SESSION');
  ALTER PROCEDURE myproc COMPILE;
  DBMS_WARNING.SET_WARNING_SETTING_STRING
    (v_settings, 'SESSION');
END;
```

DBMS_WARNING: Putting it all Together

You want to recompile all your PL/SQL package bodies with all warning categories enabled, and then restore the original warning settings:

```
DECLARE
  CURSOR v_mypacks IS SELECT object_name FROM USER_OBJECTS
                       WHERE object_type = 'PACKAGE BODY';
  v_settings      VARCHAR2(200);
  v_compile_stmt  VARCHAR2(200);
BEGIN
  v_settings := DBMS_WARNING.GET_WARNING_SETTING_STRING;
  DBMS_WARNING.SET_WARNING_SETTING_STRING('ENABLE:ALL','SESSION');
  FOR v_packname IN v_mypacks LOOP
    v_compile_stmt :=
      'ALTER PACKAGE '||v_packname.object_name||' COMPILE BODY';
    EXECUTE IMMEDIATE v_compile_stmt;
  END LOOP;
  DBMS_WARNING.SET_WARNING_SETTING_STRING(v_settings,'SESSION');
END;
```

Terminology

Key terms used in this lesson included:

- DBMS_WARNING Server-Supplied Package
- PL/SQL Compiler Errors
- PL/SQL Compiler Warnings

Summary

In this lesson, you should have learned how to:

- Explain the similarities and differences between a warning and an error
- Compare and contrast the warning levels which can be set by the `PLSQL_WARNINGS` parameter
- Set warning levels by calling the `DBMS_WARNING` server-supplied package from within a PL/SQL program

