



Using the PLSQL Compiler



Prior to Oracle Database 10g
The PL/SQL compiler translated
your source text to system code
**without applying many changes to
improve performance.**

Now, PL/SQL uses an optimizer that
can rearrange code for better
performance.

Using the PL/SQL Compiler

- **PL/SQL** uses a new optimizing compiler that can rearrange code for better performance.
- **PL/SQL** performance is improved across the board.
- Most improvements are automatic.
- The default optimization level improves performance for a broad range of **PL/SQL** operations.

PLSQL_OPTIMIZE_LEVEL
Default is 2



Changes in the PL/SQL Compiler

- Elimination of compiler-generated temporary operands
- Computation of some operations during compilation
- Reuse of some expression values
- Simplification or elimination of some branches and dead code elimination
- Avoidance of library calls by direct execution in the PL/SQL virtual machine of some operations
- All cursors correctly finalized upon exit from a cursor loop or a declare block
- Elimination of computations whose only effect is, as a side effect, to raise an exception



Initialization Parameters for PL/SQL Compilation

- PLSQL_CODE_TYPE
- PLSQL_OPTIMIZE_LEVEL
- PLSQL_CCFLAGS
- PLSQL_WARNINGS

```
SELECT name, value  
FROM v$parameter  
WHERE name = 'plsql_code_type'
```

Login as sysdba and : Grant select on [v_\\$parameter](#) to hr

The New Compiler Settings Since Oracle 10g

Compiler Option	Description
PLSQL_CODE_TYPE	Specifies the compilation mode for PL/SQL library units.
PLSQL_OPTIMIZE_LEVEL	Specifies the optimization level to be used to compile PL/SQL library units.
PLSQL_WARNINGS	Enables or disables the reporting of warning messages by the PL/SQL compiler.
PLSQL_CCFLAGS	Controls conditional compilation of each PL/SQL library unit independently.

In general, for the fastest performance, use the following setting:

```
PLSQL_CODE_TYPE = NATIVE  
PLSQL_OPTIMIZE_LEVEL = 2
```



Using the Initialization Parameters for PL/SQL Compilation

- **PLSQL_CODE_TYPE**: Specifies the compilation mode for PL/SQL library units

```
PLSQL_CODE_TYPE = { INTERPRETED | NATIVE }
```

INTERPRETED

PL/SQL library units will be compiled to PL/SQL bytecode format.
Such modules are executed by the PL/SQL interpreter engine.

NATIVE

PL/SQL library units (with the possible exception of top-level anonymous PL/SQL blocks) will be compiled to native (machine) code.
Such modules will be executed natively without incurring any interpreter overhead.

- When the value of this parameter is changed, it has no effect on PL/SQL library units that have already been compiled. The value of this parameter is stored persistently with each library unit.
- If a PL/SQL library unit is compiled native, all subsequent automatic recompilations of that library unit will use native compilation.

Using the Initialization Parameters for PL/SQL Compilation

- **PLSQL_CODE_TYPE**: Specifies the compilation mode for PL/SQL library units

```
PLSQL_CODE_TYPE = { INTERPRETED | NATIVE }
```

1- How can I change the **plsql_code_type**?

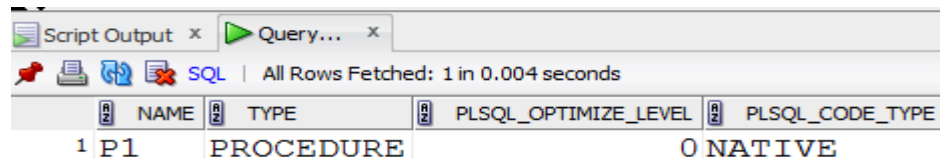
BY **ALTER SYSTEM** or **ALTER SESSION**

- *ALTER SESSION SET plsql_code_type=native;*
- *ALTER SESSION SET plsql_code_type= INTERPRETED;*

2- How can I display the setting of this parameter for objects?

From : **user_plsql_object_settings** (**DBA_ ALL_**)

- *select * from user_plsql_object_settings where name ='P1'*



Script Output x Query... x

SQL | All Rows Fetched: 1 in 0.004 seconds

	NAME	TYPE	PLSQL_OPTIMIZE_LEVEL	PLSQL_CODE_TYPE
1	P1	PROCEDURE	0	NATIVE

Using the Initialization Parameters for PL/SQL Compilation

- **PLSQL_CODE_TYPE**: Specifies the compilation mode for PL/SQL library units

```
PLSQL_CODE_TYPE = { INTERPRETED | NATIVE }
```

Note:

The `PLSQL_CODE_TYPE` parameter in Oracle Database 10g replaced the following obsolete parameters:

- `PLSQL_NATIVE_C_COMPILER`
- `PLSQL_NATIVE_MAKE_FILE_NAME`
- `PLSQL_NATIVE_C_COMPILER`
- `PLSQL_NATIVE_MAKE_UTLITY`
- `PLSQL_NATIVE_LINKER`



- **PLSQL_OPTIMIZE_LEVEL: Specifies the optimization level to be used to compile PL/SQL library units**

```
PLSQL_OPTIMIZE_LEVEL = { 0 | 1 | 2 | 3 }
```

0: Maintains the evaluation order and hence the pattern of side effects, exceptions, and package initializations of Oracle9i and earlier releases. Also removes the new semantic identity of `BINARY_INTEGER` and `PLS_INTEGER` and restores the earlier rules for the evaluation of integer expressions. Although code will run somewhat faster than it did in Oracle9i, use of level 0 will forfeit most of the performance gains of PL/SQL starting with Oracle Database 10g.

1: Applies a wide range of optimizations to PL/SQL programs including the elimination of unnecessary computations and exceptions, but generally does not move source code out of its original source order

2: Applies a wide range of modern optimization techniques beyond those of level 1 including changes which may move source code relatively far from its original location

3: This value is new in Oracle Database 11g. It applies a wide range of optimization techniques

This enables procedure inlining, which is an optimization process that replaces procedure calls with a copy of the body of the procedure to be called.



- **PLSQL_OPTIMIZE_LEVEL: Specifies the optimization level to be used to compile PL/SQL library units**

```
PLSQL_OPTIMIZE_LEVEL = { 0 | 1 | 2 | 3 }
```

The PLSQL_OPTIMIZE_LEVEL Parameter (continued)

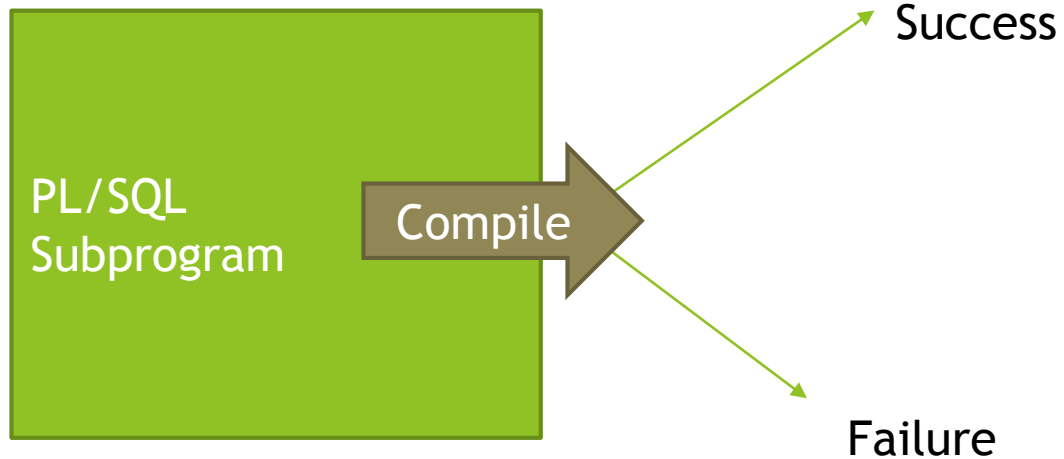
Generally, setting this parameter to 2 pays off in terms of better execution performance. If, however, the compiler runs slowly on a particular source module or if optimization does not make sense for some reason (for example, during rapid turnaround development), then setting this parameter to 1 results in almost as good a compilation with less use of compile-time resources. The value of this parameter is stored persistently with the library unit.



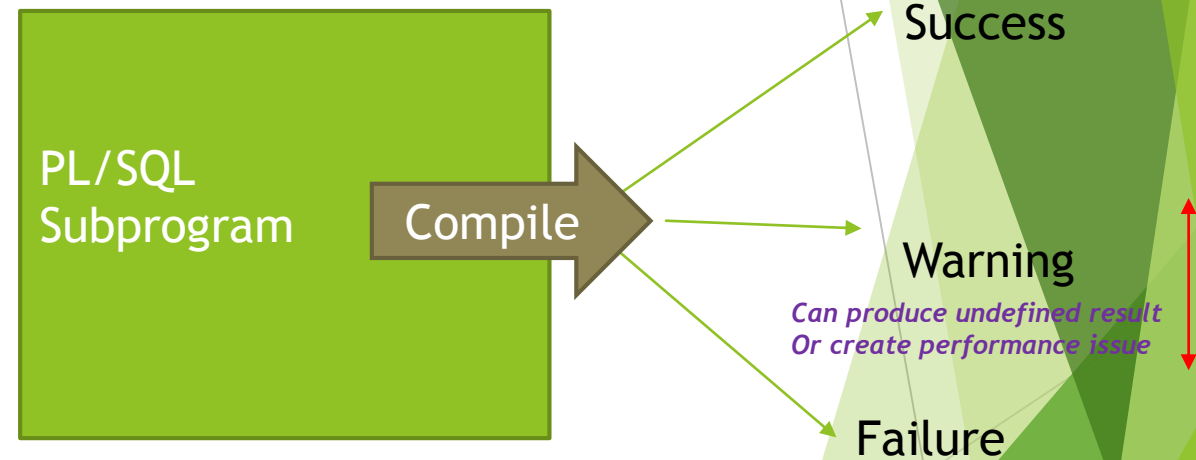
Overview of PL/SQL Compile-Time Warnings for Subprograms

plsql_warnings

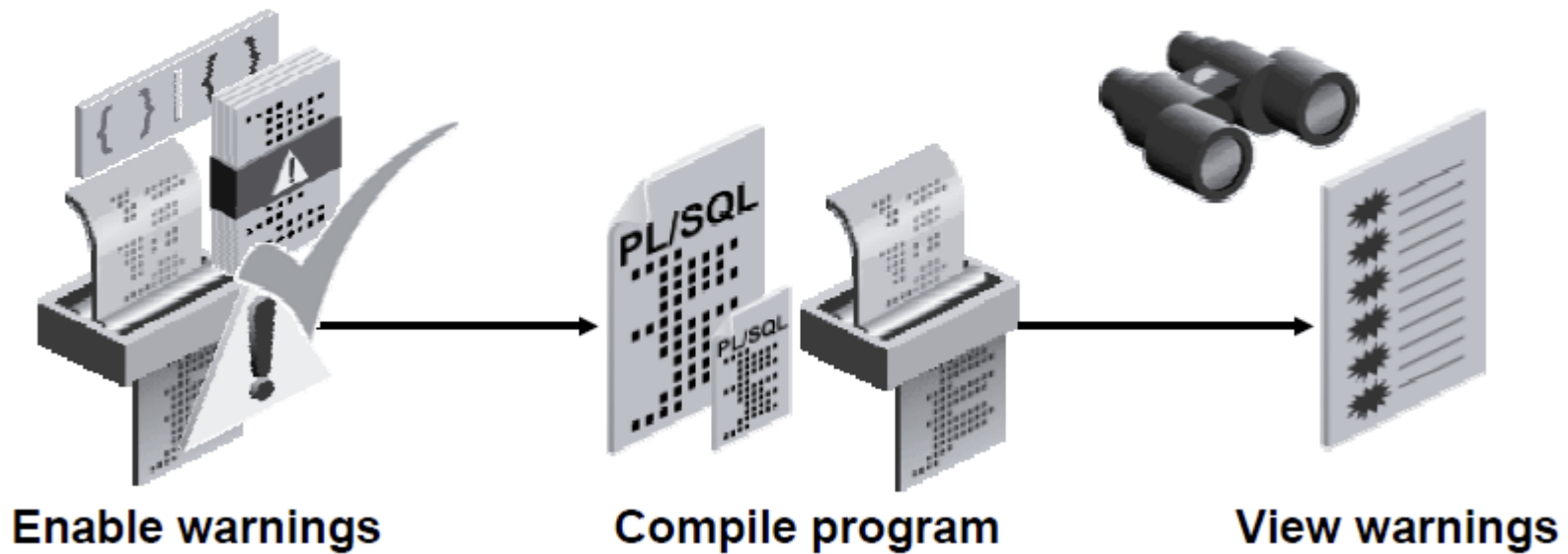
Before Oracle 10g



Oracle 10g, 11g, 12c



Starting with Oracle 10g, the PL/SQL compiler has been enhanced to produce warnings for subprograms.



Examples of warning messages:

```
Procedure p1  
( IO_TBL out datatype-collection )  
Is  
Begin  
...  
End;
```

Compile

SP2-0804: Procedure created with compilation warnings

PLW-07203: Parameter ' IO_TBL ' may benefit from use of the NOCOPY compiler hint

With the PL/SQL compiler-warning feature, compiling a PL/SQL program could have additional possible outcomes:

- Success with compilation warnings
- Failure with compilation errors and compilation warnings



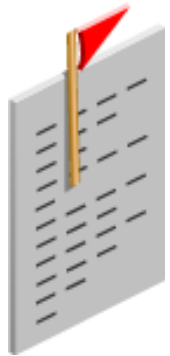
Benefits of Compiler Warnings

- Make programs more robust and avoid problems at run time
- Identify potential performance problems
- Identify factors that produce undefined results

Notes:

- Anonymous Blocks do not produce any warnings.
- All PL/SQL warning messages use the prefix PLW.

Categories of PL/SQL Compile-Time Warning Messages



SEVERE



PERFORMANCE



INFORMATIONAL



ALL

SEVERE: Messages for conditions that may cause unexpected behavior or wrong results, such as aliasing problems with parameters

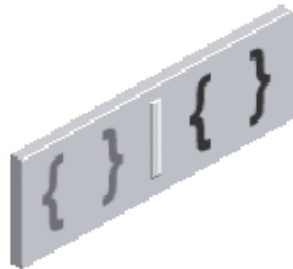
PERFORMANCE: Messages for conditions that may cause performance problems, such as passing a `VARCHAR2` value to a `NUMBER` column in an `INSERT` statement

INFORMATIONAL: Messages for conditions that do not have an effect on performance or correctness, but that you may want to change to make the code more maintainable, such as unreachable code that can never be executed

Setting the Warning Messages Levels

You can set warning levels using one of the following methods:

- **Declaratively:**
 - Using the `PLSQL_WARNINGS` initialization parameter
- **Programmatically:**
 - Using the `DBMS_WARNING` package



`PLSQL_WARNINGS`
initialization parameter



`DBMS_WARNING`
package

Setting Compiler Warning Levels: Using PLSQL_WARNINGS

```
ALTER [SESSION|SYSTEM]
PLSQL_WARNINGS = 'value_clause1'[, 'value_clause2']...
```

value_clause = Qualifier Value : Modifier Value

Qualifier Value = { ENABLE | DISABLE | ERROR }

Modifier Value =
{ ALL | SEVERE | INFORMATIONAL | PERFORMANCE |
{ integer | (integer [, integer] ...) } }

Examples

```
ALTER SESSION SET PLSQL_WARNINGS='DISABLE:ALL';

ALTER SESSION SET PLSQL_WARNINGS='ENABLE:INFORMATIONAL';

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

```
ALTER SESSION
SET plsql_warnings = 'enable:severe',
                    'enable:performance',
                    'disable:informational';
```

```
ALTER SESSION SET PLSQL_WARNINGS='ENABLE:SEVERE',
                                'DISABLE:PERFORMANCE' , 'ERROR:05003';
```

Treat plw 05003 as error

5000-5999 for severe
6000-6249 for informational
7000-7249 for performance

Viewing the Current Value of the PLSQL_WARNINGS

1- v\$parameter

```
select name,value
from v$parameter
where name='plsql_warnings'
```

Query Result x

All Rows Fetched: 1 in 0.006 seconds

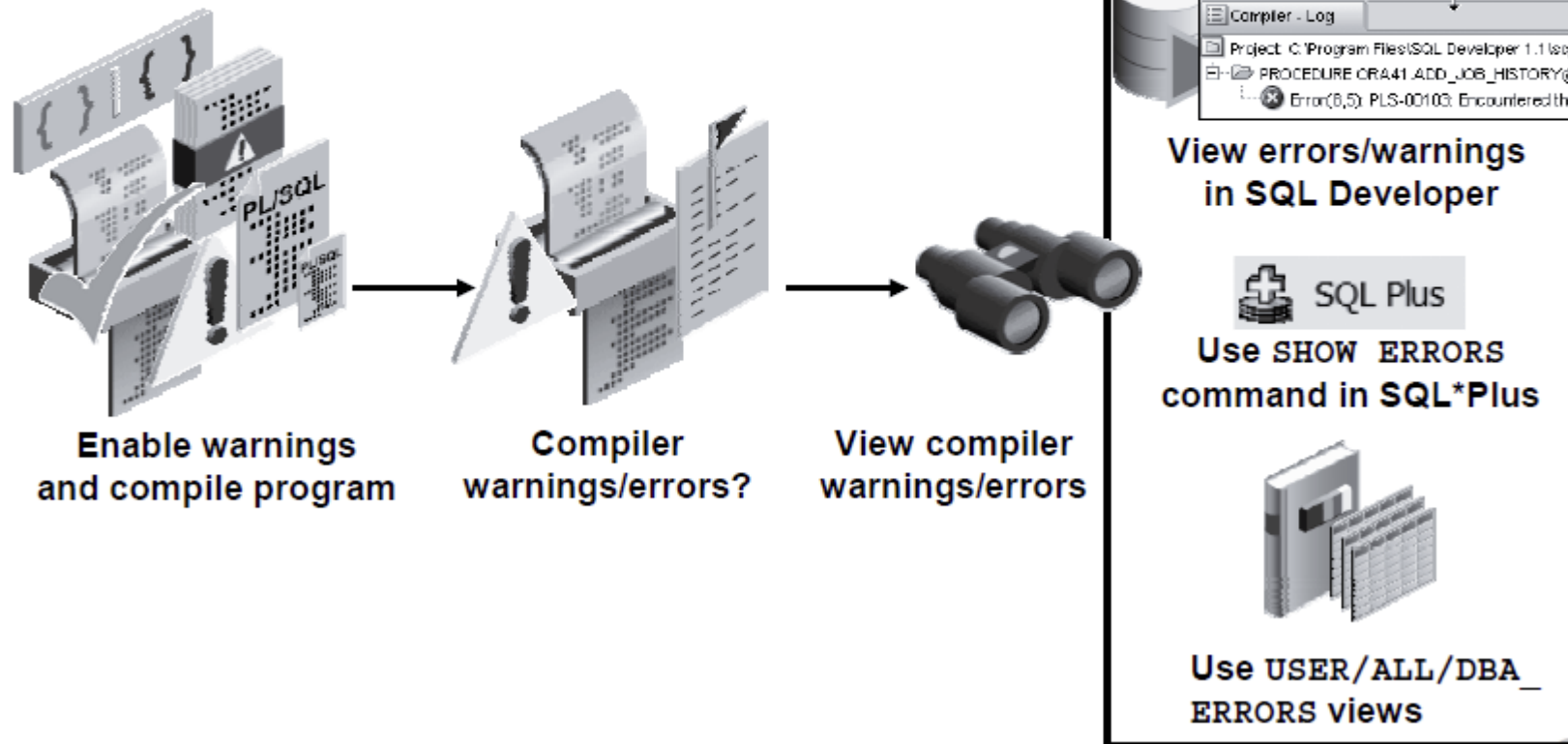
NAME	VALUE
1 plsql_warnings	DISABLE:ALL

2- dbms_warning.get_warning_setting_string()

```
declare
s varchar2(100);
begin
s:= dbms_warning.get_warning_setting_string();
dbms_output.put_line(s);
end;
```



Viewing the Compiler Warnings: Using SQL Developer, SQL*Plus, or Data Dictionary Views



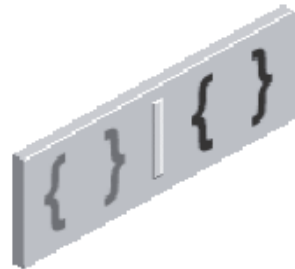
Guidelines for Using PLSQL_WARNINGS

- The settings for the PLSQL_WARNINGS parameter are stored along with each compiled subprogram.
- If you recompile the subprogram using one of the following statements, the current settings for that session are used:
 - CREATE OR REPLACE
 - ALTER ... COMPILE
- If you recompile the subprogram using the ALTER ... COMPILE statement with the REUSE SETTINGS clause, the original setting stored with the program is used.

Setting the Warning Messages Levels

You can set warning levels using one of the following methods:

- **Declaratively:**
 - Using the `PLSQL_WARNINGS` initialization parameter
- **Programmatically:**
 - Using the `DBMS_WARNING` package

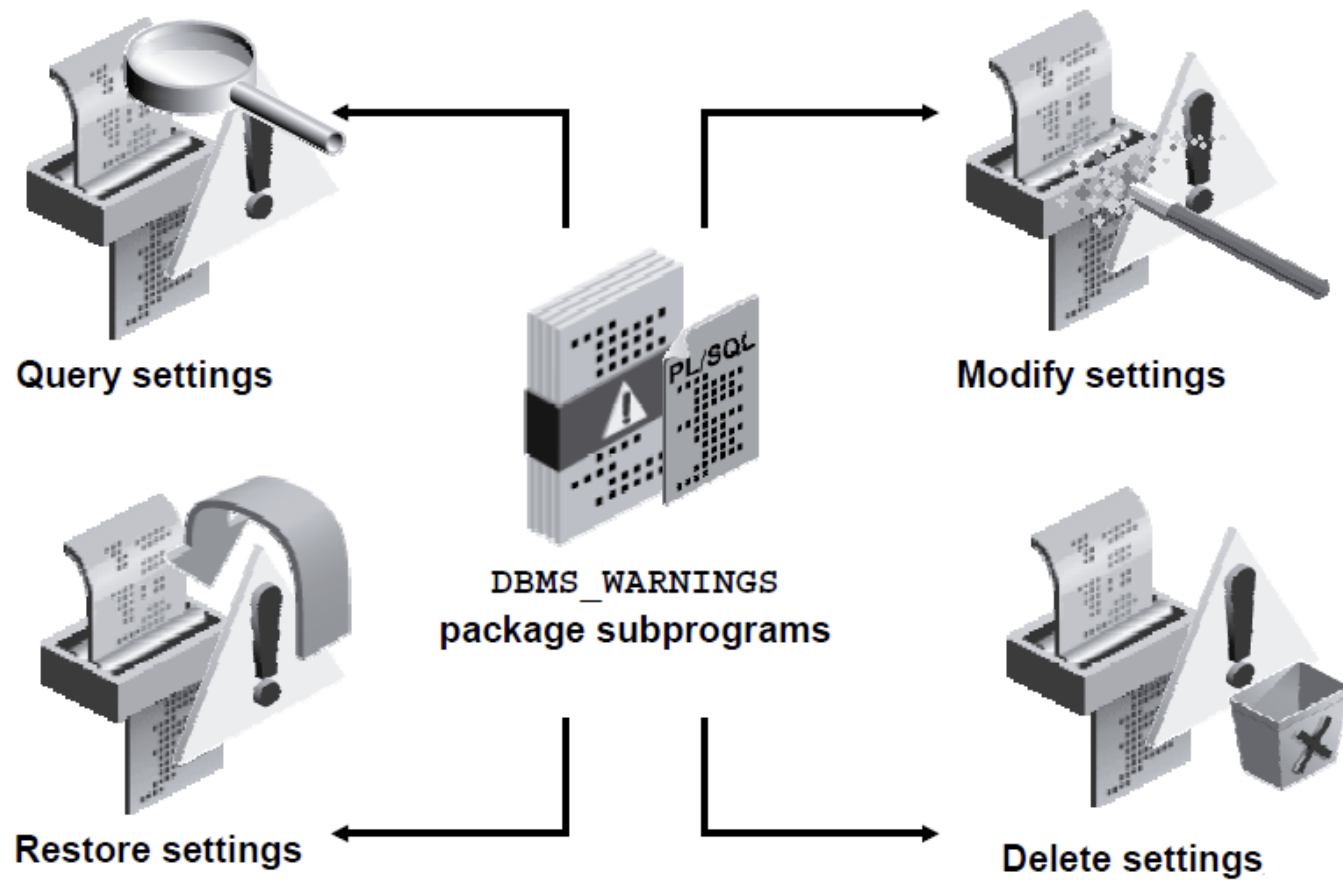


`PLSQL_WARNINGS`
initialization parameter



`DBMS_WARNING`
package

Setting Compiler Warning Levels: Using the DBMS_WARNING Package



Using the DBMS_WARNING Package Subprograms

Scenario	Subprograms to Use
Set warnings	ADD_WARNING_SETTING_CAT (procedure) ADD_WARNING_SETTING_NUM (procedure)
Query warnings	GET_WARNING_SETTING_CAT (function) GET_WARNING_SETTING_NUM (function) GET_WARNING_SETTING_STRING (function)
Replace warnings	SET_WARNING_SETTING_STRING (procedure)
Get the warnings' categories names	GET_CATEGORY (function)

Examples

```
EXECUTE DBMS_WARNING.ADD_WARNING_SETTING_CAT (-  
    warning_category      IN      VARCHAR2,  
    warning_value         IN      VARCHAR2,  
    scope                 IN      VARCHAR2);
```

BEGIN

dbms_warning.add_warning_setting_cat('SEVERE', 'ENABLE', 'SESSION');

END;

```
EXECUTE DBMS_WARNING.ADD_WARNING_SETTING_NUM (-  
    warning_number        IN      NUMBER,  
    warning_value         IN      VARCHAR2,  
    scope                 IN      VARCHAR2);
```

EXEC dbms_warning.add_warning_setting_num(6002, 'DISABLE', 'SESSION');



Examples.... Continue

```
EXECUTE DBMS_WARNING.SET_WARNING_SETTING_STRING (-  
    warning_value      IN      VARCHAR2,  
    scope              IN      VARCHAR2);
```

`exec dbms_warning.set_warning_setting_string('ENABLE:ALL', 'SESSION');`

```
dbms_warning.get_category(warning_number IN PLS_INTEGER)  
RETURN VARCHAR2
```

```
-- severe  
SELECT dbms_warning.get_category(5000)  
FROM dual;
```

```
-- informational  
SELECT dbms_warning.get_category(6002)  
FROM dual;
```

```
-- performance  
SELECT dbms_warning.get_category(7203)  
FROM dual;
```

Examples.... Continue

GET_WARNING_SETTING_CAT

Returns the specific warning category setting for the current session

```
dbms_warning.get_warning_setting_cat(warning_category IN
VARCHAR2)
RETURN VARCHAR2
```

```
SELECT dbms_warning.get_warning_setting_cat('SEVERE')
FROM dual; → ENABLE:SEVERE

SELECT dbms_warning.get_warning_setting_cat('INFORMATIONAL')
FROM dual; → ENABLE:INFORMATIONAL

SELECT dbms_warning.get_warning_setting_cat('PERFORMANCE')
FROM dual; → ENABLE:PERFORMANCE
```

GET_WARNING_SETTING_NUM

Returns the specific warning number setting for the current session

```
dbms_warning.get_warning_setting_num(warning_number IN
PLS_INTEGER)
RETURN VARCHAR2
```

```
SELECT dbms_warning.get_warning_setting_num(5000)
FROM dual; → ENABLE: 5000

SELECT dbms_warning.get_warning_setting_num(6002)
FROM dual;

SELECT dbms_warning.get_warning_setting_num(7203)
FROM dual;
```

Using the New PLW 06009 Warning Message

- A new PLW warning is available in Oracle Database 11g.
- This warning indicates that the OTHERS handler of your PL/SQL subroutine can exit without executing:
 - Some form of RAISE, or
 - A call to the standard procedure `RAISE_APPLICATION_ERROR`
- A good programming practice suggests that OTHERS handlers must always pass an exception upward.



► Thank You