Take Full Advantage of the PL/SQL Compiler

Steven Feuerstein
Oracle Developer Advocate for PL/SQL
Oracle Corporation

Email: steven.feuerstein@oracle.com
Twitter: @sfonplsql
Blog: stevenfeuersteinonplsql.blogspot.com
YouTube: Practically Perfect PL/SQL

ORACLE

Just in case I live in the future, even for a moment....

Safe Harbor Statement

The following is intended to outline our general product direction. It is intended for information purposes only, and may not be incorporated into any contract. It is not a commitment to deliver any material, code, or functionality, and should not be relied upon in making purchasing decisions. The development, release, and timing of any features or functionality described for Oracle's products remains at the sole discretion of Oracle.

Most referenced code is available in my demo.zip file from the
PL/SQL Learning Library: oracle.com/oll/plsql or direct download from **http://v.gd/sfdemo**

ORACLE

Resources for PL/SQL developers

- oracle.com/plsql – official home of PL/SQL
- oracle.com/oll – Oracle Learning Library
  - Download demo.zip file with all scripts **http://v.gd/05JIWC**
- plsqlchallenge.oracle.com - weekly PL/SQL quizzes, and more
- asktom.oracle.com – 'nuff said
- livesql.oracle.com – script repository and 12/7 12c database
- oracle-developer.net - great content from Adrian Billington
- oracle-base.com - great content from Tim Hall

Compiler Features

- Automatic, transparent optimization of code
- Compile-time warnings framework to help you improve the quality and performance of your code.
- Conditional compilation
  - *You* decide what code should be compiled/ignored!

The Optimizing Compiler

- The PL/SQL compiler now has the ability to automatically optimize your code.
  - The compiler rearranges your code.
  - Compile time increases, runtime performance improves.
- You choose the level of optimization :
  - 0 No optimization
  - 1 Smaller scale change, less impact on compile times
  - 2 Most aggressive, maximum possible code transformations, biggest impact on compile time. [default]
  - 3 (Oracle11g) In-lining of local subprograms, in addition to all the optimization performed at level 2
- Stick with the default, unless you have a clear need for an exception.

The PL/SQL Optimizer: High Level View

- The optimizer takes advantage of "freedoms" to optimize code.
  - In essence, changing the *route* that the runtime engine takes to get from point A to point B.
- Some examples:
  - Unless otherwise specified, operands of an expression operator may be evaluated in any order.
  - Operands of a commutative operator may be commuted.
  - The actual arguments of a call or a SQL statement may be evaluated in any order (including default actual arguments).
- Optimization does not change the *logical* behavior of your code.
  - Optimization should not, for example, cause any of your regression tests to suddenly fail!
- Check out "Freedom, Order and PL/SQL Optimizations" on oracle.com/plsql!
  - Also "What a surprise! Or not!" via blogs.oracle.com/plsql-and-ebr

Some Optimization Examples

```
... A + B ...
...
... A + B ...
```

```
T := A + B;
... T ...
...
... T ...
```

T is a generated variable. We never see it. And one operation is saved.

```
for i in 1 .. 10 loop
   A := B + C;
   ...
end loop;
```

```
A := B + C;
for i in 1 .. 10 loop
   ...
end loop;
```

Automatic relocation of a loop invariant. Avoid repetitive computations.

```
FOR rec in (SELECT ...)
LOOP
   ... do stuff
END LOOP;
```

```
SELECT ...
  BULK COLLECT INTO ...
  FROM ...
```

Execute cursor FOR loop at BULK COLLECT levels of performance.

10g_optimize_cfl.sql

Things to Keep in Mind

```
my_function () * NULL
```

- The PL/SQL runtime engine will *almost always* execute your subprograms, even if the optimizer detects that the results of that subprogram call are "not needed."
  - Exception: function result cache
- You cannot rely on a specific order of evaluation of arguments in a subprogram call or even when package initialization takes place.
  - The compiler will even *avoid* initialization of a package if it not needed (using a TYPE for example).

In-lining optimization

```
ALTER SESSION SET PLSQL_OPTIMIZE_LEVEL = 3;
```

- A new level, 3, tells Oracle to automatically search out opportunities to "inline" code for nested subprograms.
  - This means that a pointer to the subprogram is replaced with the implementation of the subprogram.
- Oracle's own tests have shown 10-20% performance improvement.
  - Depends on how many local modules you create and how often they are used.
- Note: compile code size increases.

11g_inline*.sql

Selective Inlining with PRAGMA

```
PRAGMA INLINE (subprogram, 'YES')
```

- You can also keep the optimization level at 2 and request inlining explicitly for specific subprogram invocations with a new INLINE pragma.
- Inlining applies to the following statements:
  - Assignment, CALL, conditional, CASE, CONTINUE-WHEN, EXECUTE IMMEDIATE, EXIT-WHEN, LOOP, RETURN
- You can also request inlining for all executions of the subprogram by placing the PRAGMA before the declaration of the subprogram.
- Inlining, like NOCOPY, is a request and can be rejected by the compiler.
- Under some circumstance, inlining can result in *slower* code.

Warnings help you build *better* code

- Your code compiles without errors. Great, you can run that program!
- But does it use the PL/SQL language *optimally*?
- PL/SQL offers a compile-time warnings feature to answer this question.
  - Automatically informs you of ways to *improve* the quality or performance of your code.
- Available warnings are documented in the Oracle Database Error Messages document: http://docs.oracle.com/database/121/ERRMG/toc.htm
  - PLS prefix = PL/SQL compiler error
  - PLW prefix = compile-time warning

Enable and Disable Warnings

- To use compiler warnings, you must turn them on for session or for a particular program unit.
  - By default, warnings are *disabled*.
- Can specify individual warnings or categories.
- SQL Developer Preferences/PL/SQL Compiler offers UI access.

```
ALTER SESSION [ENABLE | DISABLE |ERROR]:
   [ALL|SEVERE|INFORMATIONAL|PERFORMANCE|warning_number]

REM To enable all warnings in your session:
ALTER SESSION SET plsql_warnings = 'enable:all';

REM If you want to enable warning message number 06002 and all warnings in
REM the performance category, and treat 5005 as a "hard" compile error:
ALTER PROCEDURE my_procedure SET plsql_warnings =
   'enable:06002', 'enable:performance', 'ERROR:05005';
```

Checking for Warnings

- The USER_ERRORS data dictionary view shows both "hard" errors and compilation warnings.
- Use the SHOW ERRORS command in SQL*Plus.
- IDEs will usually display warnings within the edit window.
- Or run your own query against USER_ERRORS.

Example: check for unreachable code

- There may be lines of code that could never, ever execute.

```
SQL> CREATE OR REPLACE PROCEDURE unreachable_code IS
2 x NUMBER := 10;
3 BEGIN
4 IF x = 10 THEN
5 x := 20;
6 ELSE
7 x := 100; -- unreachable code
8 END IF;
9 END unreachable_code;
10 /
SP2-0804: Procedure created with compilation warnings

SQL> show err
Errors for PROCEDURE UNREACHABLE_CODE:

LINE/COL ERROR
-------- -----------------------------------
7/7 PLW-06002: Unreachable code
```

plw6002.sql

Useful warnings added in 11.1

- PLW-6017: something's going to raise an error!
  - Such as VARCHAR2(1) := 'abc'....FINALLY!
- PLW-6009: OTHERS exception handler does not re-raise an exception.
- More feedback on impact of optimization
  - PLW-6007: Notification that entire subprograms were removed
- PLW-7205: warning on mixed use of integer types
  - Namely, SIMPLE_INTEGER mixed with PLS_INTEGER and BINARY_INTEGER
- PLW-7206: unnecessary assignments
- Lots of PRAGMA INLINE-related warnings

plw*.sql files

The Oracle Knows: an error will occur

```
CREATE OR REPLACE PROCEDURE plw6017
IS
   c   VARCHAR2 (1) := 'abc';
BEGIN
```

- One big frustration I have had with compile-time warnings is that it did not flag code like you see above. What could be more basic?
- This (and more) is finally addressed in Oracle11g with the PLW-06017 warning.

```
PLW-06017: an operation will raise an exception
```

plw6017.sql

Treating a warning as "hard" compile error

- You might identify a warning that reflects such bad coding practices, that you want to ensure it *never* makes its way into production code.
  - Just set the warning as an error and stop the use of that program "in its tracks."
- "Function does not return value" is a prime example.
  - You *never* want this error to appear to users. Too embarrassing.

```
ALTER SESSION SET PLSQL_WARNINGS='ERROR:5005'
```

plw5005.sql

Conclusions - Compile-time Warnings

- Review the available warnings. Identify those which are of greatest importance to you.
  - And with each new release of Oracle check for additions.
- Consider setting up scripts to enable different sets of warnings to match different development scenarios and to ignore those "nuisance" warnings.
- Or go radical: enable ALL warnings as ERRORS, and go for a 100% clean compile every single time!

```
ALTER SESSION SET PLSQL_WARNINGS='ERROR:ALL'
```

Conditional Compilation

- Compile selected parts of a program based on *conditions* you provide with various compiler *directives*.
- With conditional compilation you can:
  - Write code that will compile and run under different versions of Oracle (relevant for future releases).
  - Run different code for test, debug and production phases. That is, compile debug statements in and out of your code.
  - Expose private modules for unit testing, but hide them in production.

A finely-nuanced feature of PL/SQL

- Conditional compilation affects how your code is compiled and therefore executed.
- It is not something to employ casually.
  - This training will serve as an *introduction*.
- Before using conditional compilation, check out Bryn Llewellyn's detailed whitepaper on the topic.
  - 100 pages covering all common use cases
  - See URL below or search for "conditional compilation white paper".

http://bit.ly/eXxJ9Q

Three types of compiler *directives*

- Inquiry directives: $$identifier
  - Use the $$identifier syntax to refer to conditional compilation flags. These inquiry directives can be referenced within an $IF directive, or used independently in your code.
- Selection directives: $IF
  - Use the $IF directive to evaluate expressions and determine which code should be included or avoided.
  - Can reference inquiry directives and package static constants.
- Error directives: $ERROR
  - Use the $ERROR directive to report compilation errors based on conditions evaluated when the preprocessor prepares your code for compilation.

Example: toggle inclusion of tracing

- Set up conditional compilation of debugging and tracing with special "CC" flags that are placed into the compiler settings for a program.
  - Only integer and Boolean values are allowed.

```
ALTER SESSION SET PLSQL_CCFLAGS = 'oe_debug:true, oe_trace_level:10'
/

CREATE OR REPLACE PROCEDURE calculate_totals
IS
BEGIN
$IF $$oe_debug AND $$oe_trace_level >= 5
$THEN
   DBMS_OUTPUT.PUT_LINE ('Tracing at level 5 or higher');
$END
   application_logic;
END calculate_totals;
/
```

cc_debug_trace.sql
cc_expose_private.sql
cc_max_string.sql
cc_plsql_compile_settings.sql

Access to post-processed code

- You can display or retrieve post-processed code with the DBMS_PREPROCESSOR package.
  - Oracle is careful to preserve both horizontal and vertical whitespace so runtime stack and error information correlates to your actual source code.

```
CREATE OR REPLACE PROCEDURE
   post_processed
IS
BEGIN
$IF $$PLSQL_OPTIMIZE_LEVEL = 1
$THEN
   -- Slow and easy
  NULL;
$ELSE
   -- Fast and modern and easy
   NULL;
$END
END post_processed;
/
```

```
BEGIN
   DBMS_PREPROCESSOR.PRINT_POST_PROCESSED_SOURCE
   ('PROCEDURE', USER, 'POST_PROCESSED');
END;
/

PROCEDURE post_processed
IS
BEGIN




   -- Fast and modern and easy
   NULL;

END post_processed;
```

cc_postprocessed.sql

Error directive example

- If my program has been compiled with optimization level 1 (less aggressive) or 0 (disabled), then raise an error.
  - You can in this way add "meta-requirements" to your code definitions.

```
SQL> CREATE OR REPLACE PROCEDURE long_compilation
  2  IS
  3  BEGIN
  4  $IF $$plsql_optimize_level < 2
  5  $THEN
  6     $error 'Program must be compiled with full optimization' $end
  7  $END
  8     NULL;
  9  END long_compilation;
 10  /
```

cc_opt_level_check.sql

Using DBMS_DB_VERSION

- This package contains a set of Boolean constants showing absolute and relative version information.

```
PROCEDURE insert_rows ( rows_in IN otn_demo_aat ) IS
BEGIN
$IF DBMS_DB_VERSION.VER_LE_10_1
$THEN
   BEGIN
      ...
      FORALL indx IN 1 .. l_dense.COUNT
         INSERT INTO otn_demo VALUES l_dense (indx);
   END;
$ELSE
   FORALL indx IN INDICES OF rows_in
      INSERT INTO otn_demo VALUES rows_in (indx);
$END
```

cc_bf_or_number.sql
cc_version_check.sql

Conclusions – Conditional Compilation

- Conditional compilation is a very powerful and useful feature.
- It is not terribly difficult to learn, but it is hard for developers to be confident of working with and maintaining code that contains "$" syntax elements.
- Keep it in mind when you encounter a clear and compelling use case, such as writing code that must run under multiple versions of Oracle.

Compiler Features - Summary

- Optimizer
  - Go with the default and enjoy the performance!
- Compile-time warnings
  - Turn them all on!
  - Make them all errors!
  - Well, at least give it a try. ☺
- Conditional compilation
  - Powerful feature for specific use cases

ORACLE®