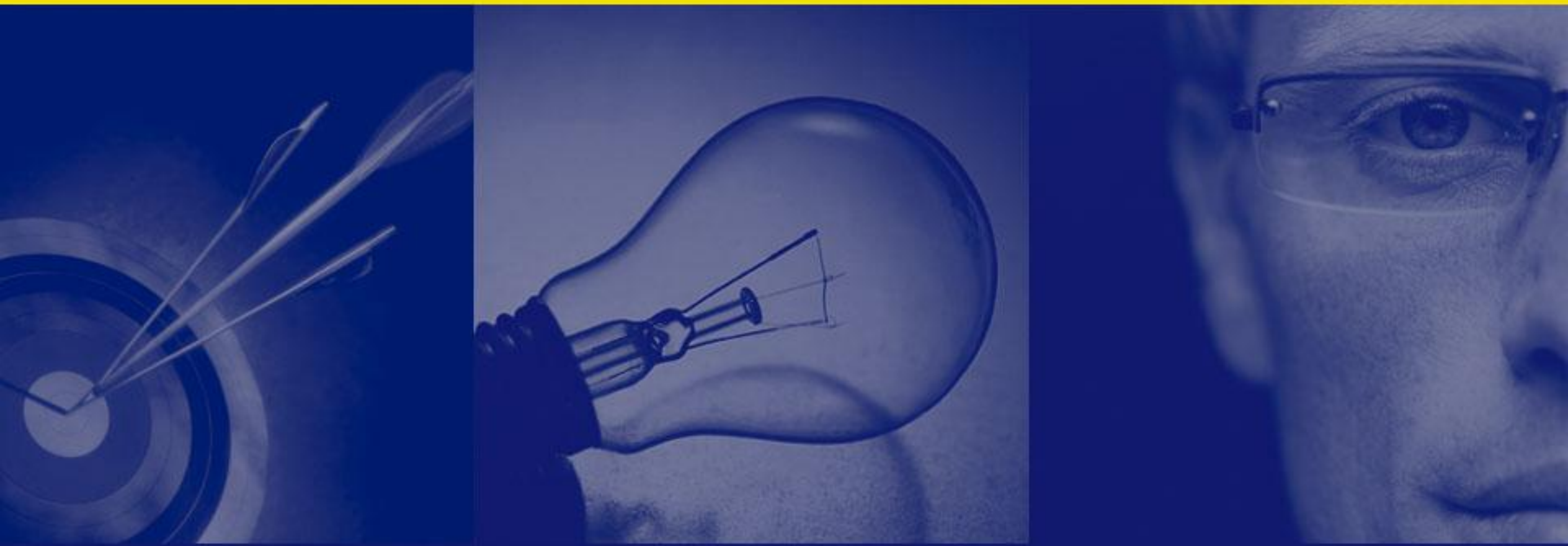
Effective Management of PL/SQL-based Applications



Steven Feuerstein
PL/SQL Evangelist, Quest Software
www.ToadWorld.com/SF
steven.feuerstein@quest.com

How to benefit most from this session

- **Watch, listen, *ask questions*. Then afterwards....**
- **Download and use any of my the training materials, available at my "cyber home" on Toad World, a portal for Toad Users and PL/SQL developers:**

PL/SQL Obsession **http://www.ToadWorld.com/SF**

- **Download and use any of my scripts (examples, performance scripts, reusable code) from the demo.zip, available from the same place.**

filename_from_demo_zip.sql

- **You have my permission to use *all* these materials to do internal trainings and build your own applications.**
  - **But they should not considered production ready.**
  - ***Test them* and modify them to fit your needs.**

And some other incredibly fantastic and entertaining websites for PL/SQL

PLSQLChallenge.com

Play Daily. Win Monthly. Brag Often.

Covers Versions Through Oracle Database 11g Release 2
5th Edition
Oracle PL/SQL
Programming
O'REILLY
Steven Feuerstein with Bill Pribyl

Effective Management of PL/SQL

- Where and how to store code
- Fully leverage the PL/SQL compiler
- Analyze memory usage of PL/SQL code
- Code analysis with data dictionary views
- Managing dependencies and invalidations

Where and how to store source code

- **Use version control software.**
- **Do not save the original version of your code in the database.**
  - Too easy to experience "lost updates"
  - Always save to files and check in those files.
- **It's never too soon to make a backup.**
  - Just copy files to other directories
- **Separate type and package specs and bodies.**
  - Need to be able to recompile bodies while leaving the specification intact.

Fully Leverage the Oracle10g PL/SQL Compiler

- **Oracle demonstrates its long-term commitment to PL/SQL in Oracle10g with major enhancements to the PL/SQL compiler.**
  - Automatic, transparent optimization of code
  - Compile-time warnings framework to help you improve the quality of your code.
  - Conditional compilation: you get to decide what code should be compiled/ignored.
- **Oracle11g offers enhancements to both compile-time warnings and optimization.**

An optimizing compiler

- **The PL/SQL compiler now has the ability to automatically optimize your code.**
  - At the time of compilation, Oracle rearranges your code to improve performance.
- **You can choose the level of optimization through the plsql_optimize_level setting:**
  - 3 (Oracle11g) Inlining of local subprograms
  - 2 Most aggressive, maximum possible code transformations, biggest impact on compile time. [default]
  - 1 Smaller scale change, less impact on compile times
  - 0 Pre-10g compilation without optimization

```
ALTER SESSION SET PLSQL_OPTIMIZE_LEVEL =
                                     1;
```

10g_optimize_cfl.sql

Learn more about the PL/SQL optimizer

```
http://www.oracle.com/technology/tech/pl_sql/htdocs/new_in_10gr1.htm
```

- ***PL/SQL Just Got Faster***
  - Explains the workings of the PL/SQL compiler and runtime system and shows how major improvements on this scale are indeed possible.
- ***PL/SQL Performance Measurement Harness***
  - Describes a performance experiment whose conclusion is the large factors quoted above. We've provided a downloadable kit to enable you to repeat the experiment yourself.
- ***Freedom, Order, and PL/SQL Optimization***
  - Intended for professional PL/SQL programmers, explores the use and behavior of the new compiler.
- ***PL/SQL Performance — Debunking the Myths***
  - Re-examines some old notions about PL/SQL performance.

Optimizing compiler details

- **Oracle retains optimizer settings on a module-by-module basis.**
  - When you recompile a particular module with non-default settings, the settings will "stick," allowing you to recompile later using REUSE SETTINGS. For example:

```
ALTER PROCEDURE bigproc COMPILE PLSQL_OPTIMIZE_LEVEL = 1;
```

- **and then:**

```
ALTER PROCEDURE bigproc COMPILE REUSE SETTINGS;
```

Warnings help you build *better* code

- **Enable compiler warnings, which identify ways in which you can improve your code.**
  - These are not *errors*, but potential problems with code structure or performance.
- **To use compiler warnings, you must turn them on in your 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 warning 5005 as a "hard" compile
error:
ALTER SESSION SET plsql_warnings =
   'enable:06002', 'enable:performance', 'ERROR:05005';
```

Compiler time warnings - example

- **Check for “unreachable” code….**

```
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
```

plw*.sql

New compile-time warnings in Oracle11g

- **PLW-6009: Exception handler does not re-raise an exception.**
- **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**
- **More feedback on impact of optimization**
  - PLW-6007: Notification that entire subprograms were removed

plw*.sql files

Conditional Compilation

- **Compile selected parts of a program based on conditions you provide with various compiler *directives*.**
- **Conditional compilation will allow you to:**
  - 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.
- **Available in 10gR2 and patch sets of 10gR1, plus 9iR2 (with guidance from Oracle Support)**

Three types of compiler directives

- **Selection directives: $IF**
  - Use the $IF directive to evaluate expressions and determine which code should be included or avoided.
- **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.
- **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.**

```
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_parameters.sql
```

Access to post-processed code

- **You can display or retrieve post-processed code with the DBMS_PREPROCESSOR package.**

```
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_SOU
   RCE
   ('PROCEDURE', USER, 'POST_PROCESSED');
END;
/

PROCEDURE post_processed
IS
BEGIN




   -- Fast and modern and easy
   NULL;

END post_processed;
```

Notice the white space.

Error directive example

- **If my program has not been compiled with optimization level 1 (less aggressive) 0=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
IS
BEGIN
$IF $$plsql_optimize_level < 2
$THEN
   $error 'Program must be compiled with full optimization' $end
$END
   NULL;
END long_compilation;
/
```

cc_opt_level_check.sql

Using DBMS_DB_VERSION

- **Each version of Oracle from Oracle Database 10g Release 2 will contain package named DBMS_DB_VERSION containing 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

Compiler Improvements - Summary

- **Optimizer**
  - – Go with the default and enjoy the performance!
- **Compile-time warnings**
  - – Try them out, see how much value you can extract from it.
- **Conditional compilation**
  - – Lots of potential, mainly for use into the future
  - – Smart tool support needed to make it feasible and maintainable (one's code becomes very hard to read)
- **Automatic inlining (Oracle11g)**
  - – Useful, but probably in a relatively limited way

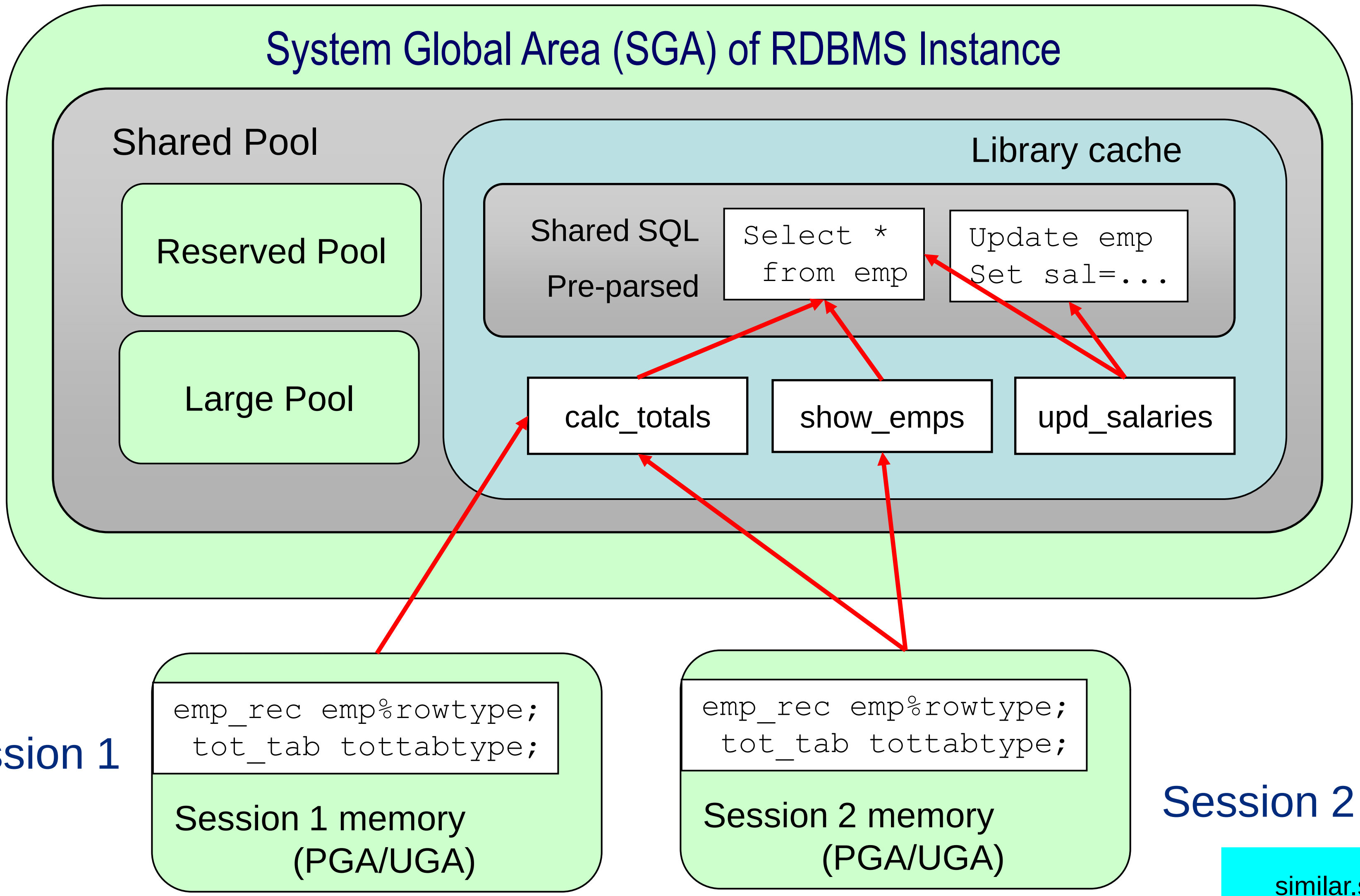
11g_inline.sql

Analyze memory usage of PL/SQL code

- **It is certainly possible to write PL/SQL code that consumes so much memory, it kills a user's session.**
  - It's quite easy to do, in fact.
- **As you work with more advanced features, like collections and FORALL, you will need to pay attention to memory, and make adjustments.**
- **First, let's review how Oracle manages memory at run-time.**

`memory_error.sql`

PL/SQL in Shared Memory



Session 1

Session 2

similar.sql

How PL/SQL uses the SGA, PGA and UGA

- **The SGA contains information that can be shared across sessions connected to the instance.**
  - From the PL/SQL perspective, this is limited to package static constants.

```
PACKAGE Pkg is                                             /* 11g feature! */
  Nonstatic_Constant CONSTANT PLS_INTEGER := My_Sequence.Nextval;
  Static_Constant    CONSTANT PLS_INTEGER := 42;
END Pkg;
```

- **The User Global Area contains session-specific data that persists across server call boundaries**
  - Package-level data
- **The Process Global Area contains session-specific data that is released when the current server call terminates.**
  - Local data

grantv$.sql
plsql_memory.pkg
plsql_memory_demo.sql

Tips for managing memory

- **Use LIMIT clause with BULK COLLECT.**
- **Use varrays with BULK COLLECT to declaratively guard against "memory creep."**
- **Use NOCOPY hint when passing IN OUT collections.**
- **Be very careful about defining collections at the package level.**
  - Memory will not be released when the block ends.
- **Use pipelined table functions.**

bulklimit.sql
varray_collection_limit.sql
nocopy*.tst
tabfunc_pipelined.sql

Code analysis with data dictionary views

- **Analyze objects defined in the database**
- **Analyze source code for contents and patterns**
- **Analyze program unit structure and header**
- **Check compile-time settings of program units**

Analyzing source code

- **ALL_SOURCE**
  - Write queries against source code to identify violations of coding standards.
  - Which programs contain/exclude particular strings?
- **Use with other data dictionary views and utilities that reference source code.**
  - DBMS_UTILITY.FORMAT_CALL_STACK
  - Profiler data
- **ALL_IDENTIFIERS (Oracle11g) –PL/Scope**
  - Analyze all references to identifiers (named elements)

valstds.pks/pkb
package_analyzer.pks/pkb
notrun.sql

11*g*

PL/Scope: powerful code analysis tool

- **A compiler-driven tool that collects information about identifiers and stores it in data dictionary views.**
- **Use PL/Scope to answer questions like:**
  - Where is a variable assigned a value in a program?
  - What variables are declared inside a given program?
  - Which programs call another program (that is, you can get down to a subprogram in a package)?
  - Find the type of a variable from its declaration.
- **PL/Scope must be enabled; it is off by default.**

```
ALTER SESSION SET plscope_settings='IDENTIFIERS:ALL'
```

Working with PL/Scope

- **Key columns in view:**
  - TYPE - the type of identifier (VARIABLE, CONSTANT, etc.)
  - USAGE – the way the identifier is used (DECLARATION, ASSIGNMENT, etc.)
  - LINE and COL – line and column within line in which the identifier is found
- **Good to know**
  - Parameters have types FORMAL IN, FORMAL OUT, FORMAL IN OUT.

11g_plscope.sql
11g_plscope_amis.sql
plscope_helper.pkg
plscope_helper.sql

Analyzing program unit structure/header

- **Source code is handy, but also "freeform" text.**
  - The more structured the data, the better.
- **ALL_PROCEDURES**
  - Information about every subprogram you can execute
  - Missing some information (the type of subprogram)
- **ALL_ARGUMENTS**
  - Information about every argument of every subprogram you can execute
  - Rich resource of information, poorly designed.
  - Can figure out type of subprogram
  - DBMS_DESCRIBE offers another access path to more or less the same data

show_authid.sql
show_deterministic.sql

all_arguments.sql
show_all_arguments*.*
show_procs_with_parm_types.sql
is_function.sf

Compile time settings for program units

- **ALL_PLSQL_OBJECT_SETTINGS**
- **Stores information about compile-time characteristics of program units.**
  - Optimization level
  - Code type: NATIVE or INTERPRETED
  - Debug settings
  - Compile-time warnings
  - Conditional compilation flags
  - PL/Scope settings

whats_not_optimal.sql
show_non_default_object_settings.sql

Managing dependencies and invalidations

- **Review of dependency model**
  – Before Oracle11g
  – Oracle11g and higher: fine grained dependencies
- **Minimizing program unit invalidations**
- **Recompiling invalid code**

Pre-Oracle11g Dependency Model

- **Dependencies tracked at object level**
  - Which tables is a program dependent on?
  - Which program units is a program dependent on?
- **So if any change is made to a referenced object, all dependent objects' status are set to INVALID.**
  - Even if the change doesn't affect the dependent object.
- **Use ALL_DEPENDENCIES to analyze.**
  - REFERENCED* columns show the objects on which an object depends.

analyzedep*.*
code_referencing_tables.sql
layer_validator*.*

Oracle11g Dependency Model

- **Now dependencies are tracked down to the sub-object level: "fine-grained dependencies" or FGD.**
  - Columns within tables
  - Parameters within program units.
- **Impact of change:**
  - You can minimize invalidation of program units.
  - You *cannot* obtain this fine-grained dependency information through any data dictionary views – yet.

11g_fgd*.sql

Recompiling invalid code

- **Code goes invalid, must be recompiled.**
  - You have lots of options.
- **Automatic recompilation by Oracle**
  - Encounters invalid program unit, will recompile (sometimes).
- **DBMS_DDL.ALTER_COMPILE**
  - Recompile a single unit.
- **DBMS_UTILITY.RECOMPILE_SCHEMA**
  - Recompile all invalid units in schema
- **UTL_RECOMP**
  - Restricted authority; parallelization of recompilation
- **Solomon Yacobson's recompile utility**
  - For all versions, clean recompile of invalid program units

alter_compile.sql
recompile.sql
recompile_comparison.sql

Managing PL/SQL Applications

- **With an understanding of how PL/SQL works...**
- **With an awareness of the many data dictionary views available for your use...**
- **With a good set of scripts....**
- **You can very effectively manage your PL/QL code, making it easier to:**
  - Analyze impact of change
  - Apply changes more easily across the code base.