Impact Analysis with PL/Scope

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
Resources for Oracle Database Developers

• Official home of PL/SQL - oracle.com/plsql

• SQL-PL/SQL discussion forum on OTN
  https://community.oracle.com/community/database/developer-tools/sql_and_pl_sql

• PL/SQL and EBR blog by Bryn Llewellyn - https://blogs.oracle.com/plsql-and-ebr

• Oracle Learning Library - oracle.com/oll

• Weekly PL/SQL and SQL quizzes, and more - plsqlchallenge.oracle.com

• Ask Tom - asktom.oracle.com – 'nuff said

• LiveSQL - 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
PL/Scope

• Added in 11.1, compiler-driven tool that collects information about identifiers and statements, 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.

• And with 12.2, you can now also analyze SQL statements in PL/SQL.
Life Without PL/Scope

• Prior to PL/Scope, analyzing impact mostly meant text searches through files, or queries against ALL_SOURCE and ALL_DEPENDENCIES views.

• ALL_DEPENDENCIES is fine for giving you dependency info at the database object level, but not below.
  – "Find all the packages that reference table X".

• With 11.1, Oracle now supports fine-grained dependencies for invalidation, but that information is not available via data dictionary views.

```
SELECT owner,
      name,
      type,
      referenced_owner || '.' ||
FROM all_dependencies
AND referenced_type IN ('TABLE', 'VIEW')
AND referenced_name = 'MY_TABLE'
ORDER BY name, referenced_owner, referenced_name
```

11g_fgd*.sql
Getting Started with PL/Scope

- PL/Scope must be enabled; it is off by default.
- When your program is compiled, information about all identifiers are written to the ALL_IDENTIFIERS view.
- You then query the contents of the view to get information about your code.
- Check the ALL_PLSQL_OBJECT_SETTINGS view for the PL/Scope setting of a particular program unit.

```sql
ALTER SESSION SET plscope_settings='IDENTIFIERS:ALL'
```
Key Columns in ALL_IDENTIFIERS

• **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

• **SIGNATURE**
  – Unique value for an identifier. Especially helpful when distinguishing between overloading of a subprogram or "connecting" subprogram declarations in package with definition in package body.

• **USAGE_ID and USAGE_CONTEXT_ID**
  – Reveal hierarchy of identifiers in a program unit
Start with some simple examples

• Show all the identifiers in a program unit
• Show all variables declared in a subprogram (not at package level)
• Show all variables declared in the package specifications
• Show the locations where a variable could be modified
More advanced examples

• Find exceptions that are defined but never raised
• Show the hierarchy of identifiers in a program unit
• Validate naming conventions with PL/Scope

plscope_unused_exceptions.sql
plscope_hierarchy.sql
plscope_naming_conventions.sql
PL/Scope Helper Utilities

• Clearly, "data mining" in ALL_IDENTIFIERS can get complicated.

• Suggestions for putting PL/Scope to use:
  – Build views to hide some of the complexity.
  – Build packages to provide high-level subprograms to perform specific actions.
12.2 Enhancements to PL/Scope

• Gathers data on SQL statements in PL/SQL program units
• You can now find:
  – where specific columns are referenced
  – all program units performing specific DML operations on table (and help you consolidate such statements)
  – all SQL statements containing hints
  – all dynamic SQL usages – ideal for getting rid of SQL injection vulnerabilities
  – locations in your code where you commit or rollback
  – multiple appearances of same SQL statement (same SQL_ID)

```
ALTER SESSION SET plscope_settings='IDENTIFIERS:ALL, STATEMENTS:ALL'
```
New ALL_STATEMENTS View

• The ALL_STATEMENTS view (along with USER_STATEMENTS) contains information about each SQL statement in program units compiled with PL/Scope enabled.
  – full_text – text of SQL statement
  – has_into_record – INTO plsql_record
  – has_current_of – Uses CURRENT OF syntax
  – has_for_update – Uses FOR UPDATE syntax
  – has_in_binds -
  – has_into_bulk – Uses BULK COLLECT INTO
  – usage_id – Same as with ALL_IDENTIFIERS – and unique across both tables!
  – sql_id – pointer to SQL statement in v$ views
Some Examples

Find SQL Statements with Hints

```
SELECT owner,
       object_name,
       line,
       full_text
FROM all_statements
WHERE has_hint = 'YES'
```

Find All DML Statements On Table

```
SELECT idt.line,
       idt.owner || '.' || idt.object_name
       code_unit,
       RTRIM(src.text, CHR(10)) text
FROM all_identifiers idt
     all_statements st
     all_source src
WHERE idt.usage = 'REFERENCE'
     AND idt.TYPE = 'TABLE'
     AND idt.name = table_in
     AND idt.owner = owner_in
     AND idt.line = src.line
     AND idt.object_name = src.name
     AND idt.owner = src.owner
     AND idt.usage_context_id = st.usage_id
```

• There's so much you can do!
More Examples!

Same SQL Statement Used > 1?

```
SELECT sql_id, text, COUNT(*)
    FROM all_statements
    WHERE sql_id IS NOT NULL
    GROUP BY sql_id, text
    HAVING COUNT(*) > 1
/
```

Same SQL_ID but different signature.

Uses BULK COLLECT INTO?

```
SELECT *
    FROM all_statements
    WHERE has_into_bulk = 'YES'
/
```
More Examples: Find dynamic SQL

Native Dynamic SQL – easy!

```
SELECT st.owner, st.object_name, st.line, s.text
  FROM all_statements st, all_source s
  WHERE st.TYPE IN ('EXECUTE IMMEDIATE', 'OPEN')
    AND st.owner = s.owner
    AND st.object_name = s.name
    AND st.line = s.line
UNION ALL

SELECT idnt.owner, idnt.object_name, idnt.line, src.text
  FROM all_identifiers idnt, all_source src
  WHERE idnt.owner <> 'SYS'
    AND idnt.signature IN (SELECT a.signature
                            FROM all_identifiers a
                            WHERE a.usage = 'DECLARATION'
                              AND a.owner = 'SYS'
                              AND a.object_name = 'DBMS_SQL'
                              AND a.object_type = 'PACKAGE')
    AND idnt.owner = src.owner
    AND idnt.name = src.name
    AND idnt.line = src.line
```

DBMS_SQL References: must recompile built-in with PL/Scope enabled!
Conclusions

• PL/Scope gives you a level of visibility into your code that was never before possible.
• With 12.2 enhancements adding analysis of SQL, you can now perform detailed analysis of the impact of changing your data structures.
• Check out my (and other) LiveSQL scripts demonstrating PL/Scope capabilities.

livesql.oracle.com

Doc: 12.2 Database Development Guide