15
Using PL/Scope

PL/Scope lets you develop powerful and effective PL/Scope source code tools that increase PL/SQL developer productivity by minimizing time spent browsing and understanding source code.

PL/Scope is intended for application developers, and is typically used in a development database environment.

> **Note:**
>
> PL/Scope cannot collect data for a PL/SQL unit whose source code is wrapped. For information about wrapping PL/SQL source code, see *Oracle Database PL/SQL Language Reference*.

Topics:

15.1 Overview of PL/Scope

PL/Scope is a compiler-driven tool that collects PL/SQL and SQL identifiers as well as SQL statements usage in PL/SQL source code.

PL/Scope collects PL/SQL identifiers, SQL identifiers, and SQL statements metadata at program-unit compilation time and makes it available in static data dictionary views. The collected data includes information about identifier types, usages (DECLARATION, DEFINITION, REFERENCE, CALL, ASSIGNMENT) and the location of each usage in the source code.

Starting with Oracle Database 12*c* Release 2 (12.2), PL/Scope has been enhanced to report on the occurrences of static SQL, and dynamic SQL call sites in PL/SQL units. The call site of the native dynamic SQL (EXECUTE IMMEDIATE, OPEN CURSOR FOR) and DBMS_SQL calls are collected. Dynamic SQL statements are generated at execution time, so only the call sites can be collected at compilation time. The collected data in the new DBA_STATEMENTS view can be queried along with the other data dictionary views to help answer questions about the scope of changes required for programming projects, and performing code analysis. It is also useful to identify the source of SQL statement not performing well. PL/Scope provides

insight into dependencies between tables, views and the PL/SQL units. This level of details can be used as a migration assessment tool to determine the extent of changes required.

PL/Scope can help you answer questions such as :

- Where and how a column x in table y is used in the PL/SQL code?
- Is the SQL in my application PL/SQL code compatible with TimesTen?
- What are the constants, variables and exceptions in my application that are declared but never used?
- Is my code at risk for SQL injection?
- What are the SQL statements with an optimizer hint coded in the application?
- Which SQL has a BULK COLLECT clause ? Where is the SQL called from ?

15.2 Privileges Required for Using PL/Scope

By default, PUBLIC has SELECT privileges on various system tables and views, and EXECUTE privileges on various PL/SQL objects.

The PL/Scope data is available in the DBA_IDENTIFIERS and DBA_STATEMENTS data dictionary views. The user must have the privileges to query data in these views.

The following privileges have been granted on these relevant views.

View Name	Privilege Granted to Role
USER_IDENTIFIERS	READ to PUBLIC
ALL_IDENTIFIERS	READ to PUBLIC
DBA_IDENTIFIERS	SELECT to SELECT_CATALOG_ROLE
USER_STATEMENTS	READ to PUBLIC
ALL_STATEMENTS	READ to PUBLIC
DBA_STATEMENTS	SELECT to SELECT_CATALOG_ROLE

A database administrator can verify the list of privileges on these views by using a query similar to the following:

```
SELECT *
FROM   SYS.DBA_TAB_PRIVS
WHERE  GRANTEE = 'PUBLIC'
AND TABLE_NAME IN
('ALL_IDENTIFIERS','USER_IDENTIFIERS','ALL_STATEMENTS','USER_STATEMENTS');
```

15.3 Specifying Identifier and Statement Collection

By default, PL/Scope does not collect data for identifiers and statements in the PL/SQL source program. To enable and control what is collected, set the PL/SQL compilation parameter PLSCOPE_SETTINGS.

Starting with Oracle Database 12*c* Release 2 (12.2), the `PLSCOPE_SETTINGS` has a new syntax that offers more controls and options to collect identifiers and SQL statements metadata. The metadata is collected in the static data dictionary views `DBA_IDENTIFIERS` and `DBA_STATEMENTS`.

To collect PL/Scope data for all identifiers in the PL/SQL source program, including identifiers in package bodies, set the PL/SQL compilation parameter `PLSCOPE_SETTINGS` to

'IDENTIFIERS:ALL'. The possible values for the IDENTIFIERS clause are : ALL, NONE (default), PUBLIC, SQL, and PLSQL. New SQL identifiers are introduced for : ALIAS, COLUMN, MATERIALIZED VIEW, OPERATOR, TABLE, and VIEW. The enhanced metadata collection enables the generation of reports useful for understanding the applications. PL/Scope can now be used as a tool to estimate the complexity of PL/SQL applications coding projects with a finer granularity than previously possible.

To collect PL/Scope data for all SQL statements used in PL/SQL source program, set the PL/SQL compilation parameter PLSCOPE_SETTINGS to 'STATEMENTS:ALL'. The default value is NONE.

> **Note:**
>
> Collecting all identifiers and statements might generate large amounts of data and slow compile time.

PL/Scope stores the data that it collects in the SYSAUX tablespace. If the PL/Scope collection is enabled and SYSAUX tablespace is unavailable during compilation of a program unit, PL/Scope does not collect data for the compiled object. The compiler does not issue a warning, but it saves a warning in USER_ERRORS.

> **See Also:**
>
> - *Oracle Database Reference* for information about PLSCOPE_SETTINGS
> - *Oracle Database PL/SQL Language Reference* for information about PL/SQL compilation parameters

15.4 How Much Space is PL/Scope Data Using?

PL/Scope stores its data in the SYSAUX tablespace. If you are logged on as SYSDBA, you can use the query in Example 15-1 to display the amount of space that PL/Scope data is using.

Example 15-1 How Much Space is PL/Scope Data Using?

Query:

```
SELECT SPACE_USAGE_KBYTES
FROM V$SYSAUX_OCCUPANTS
WHERE OCCUPANT_NAME='PL/SCOPE';
```

Result:

```
SPACE_USAGE_KBYTES
------------------
              1920

1 row selected.
```

> **See Also:**
>
> *Oracle Database Administrator's Guide* for information about managing the `SYSAUX` tablespace

15.5 Viewing PL/Scope Data

To view the data that PL/Scope has collected, you can use either:

- Static Data Dictionary Views for PL/SQL and SQL Identifiers
- Static Data Dictionary Views for SQL Statements
- SQL Developer

15.5.1 Static Data Dictionary Views for PL/SQL and SQL Identifiers

The `DBA_IDENTIFIERS` static data dictionary view family display information about PL/Scope identifiers, including their types and usages.

Topics:

- PL/SQL and SQL Identifier Types that PL/Scope Collects
- About Identifiers Usages
- Identifiers Usage Unique Keys
- About Identifiers Usage Context
- About Identifiers Signature

> **See Also:**
>
> *Oracle Database Reference* for more information about the dictionary view `DBA_IDENTIFIERS` view

15.5.1.1 PL/SQL and SQL Identifier Types that PL/Scope Collects

Table 15-1 shows the identifier types that PL/Scope collects, in alphabetical order. The identifier types in Table 15-1 appear in the `TYPE` column of the `DBA_IDENTIFIERS` static data dictionary views family.

> **Note:**
>
> Identifiers declared in compilation units that were not compiled with `PLSCOPE_SETTINGS='IDENTIFIERS:ALL'` do not appear in `DBA_IDENTIFIERS` static data dictionary views family.
>
> Pseudocolumns, such as ROWNUM, are not supported since they are not user defined identifiers.
>
> PL/Scope ignores column names that are literal strings.

Table 15-1 Identifier Types that PL/Scope Collects

TYPE Column Value	Comment
`ALIAS`	New SQL identifier introduced in Oracle Database 12*c* Release 2 (12.2).
`ASSOCIATIVE ARRAY`	
`COLUMN`	New SQL identifier introduced in Oracle Database 12*c* Release 2 (12.2).
`CONSTANT`	
`CURSOR`	

`BFILE DATATYPE`
`BLOB DATATYPE`
`BOOLEAN DATATYPE`
`CHARACTER DATATYPE`
`CLOB DATATYPE`
`DATE DATATYPE`
`INTERVAL DATATYPE`
`NUMBER DATATYPE`
`TIME DATATYPE`
`TIMESTAMP DATATYPE`
	Each `DATATYPE` is a base type declared in package `STANDARD`.

`EXCEPTION`	

`FORMAL IN`
`FORMAL IN OUT`
`FORMAL OUT`	

`FUNCTION`	
`INDEX TABLE`	
`ITERATOR`	An iterator is the index of a `FOR` loop.
`LABEL`	A label declaration also acts as a context.
`LIBRARY`	
`MATERIALIZED VIEW`	New SQL identifier introduced in Oracle Database 12*c* Release 2 (12.2).
`NESTED TABLE`	
`OBJECT`	

Table 15-1 (Cont.) Identifier Types that PL/Scope Collects

TYPE Column Value	Comment
OPAQUE	Examples of internal opaque types are ANYDATA and XMLType.
OPERATOR	New SQL identifier introduced in Oracle Database 12*c* Release 2 (12.2).
PACKAGE	
PROCEDURE	
RECORD	
REFCURSOR	
SEQUENCE	New SQL identifier introduced in Oracle Database 12*c* Release 2 (12.2).
SUBTYPE	
SYNONYM	PL/Scope does not resolve the base object name of a synonym. To find the base object name of a synonym, query *_SYNONYMS.
TABLE	New SQL identifier introduced in Oracle Database 12*c* Release 2 (12.2).
TRIGGER	
UROWID	
VARIABLE	Can be object attribute, local variable, package variable, or record field.
VARRAY	
VIEW	This type is used for editioning views. New SQL identifier introduced in Oracle Database 12*c* Release 2 (12.2).

See Also:

Oracle Database Reference for more information about identifiers in the stored objects

15.5.1.2 About Identifiers Usages

PL/Scope usages are the verbs that describe actions performed on behalf of the identifier.

Table 15-2 shows the usages that PL/Scope reports, in alphabetical order. The identifier actions in Table 15-2 appear in the USAGE column of the DBA_IDENTIFIERS static data dictionary views family, which is described in *Oracle Database Reference*.

Table 15-2 Usages that PL/Scope Reports

USAGE Column Value	Description
ASSIGNMENT	An assignment can be made only to an identifier that can have a value, such as a VARIABLE. Examples of assignments are: • Using an identifier to the left of an assignment operator • Using an identifier in the INTO clause of a FETCH statement • Passing an identifier to a subprogram by reference (OUT mode) • Using an identifier as the bind variable in the USING clause of an EXECUTE IMMEDIATE statement in either OUT or IN OUT mode An identifier that is passed to a subprogram in IN OUT mode has both a REFERENCE usage (corresponding to IN) and an ASSIGNMENT usage (corresponding to OUT). Expressions and nested subqueries are not supported as assignment sources.
CALL	In the context of PL/Scope, a CALL is an operation that pushes a call onto the call stack; that is: • A call to a FUNCTION or PROCEDURE • Running or fetching a cursor identifier (a logical call to SQL) A GOTO statement, or a raise of an exception, is not a CALL, because neither pushes a call onto the call stack.
DECLARATION	A DECLARATION tells the compiler that an identifier exists, and each identifier has exactly one DECLARATION. Each DECLARATION can have an associated data type. For a loop index declaration, LINE and COL (in *_IDENTIFIERS views) are the line and column of the FOR clause that implicitly declares the loop index. For a label declaration, LINE and COL are the line and column on which the label appears (and is implicitly declared) within the delimiters << and >>.
DEFINITION	A DEFINITION tells the compiler how to implement or use a previously declared identifier. Each of these types of identifiers has a DEFINITION: • EXCEPTION (can have multiple definitions) • FUNCTION • OBJECT • PACKAGE • PROCEDURE • TRIGGER For a top-level identifier only, the DEFINITION and DECLARATION are in the same place.

Table 15-2 (Cont.) Usages that PL/Scope Reports

USAGE Column Value	Description
REFERENCE	A REFERENCE uses an identifier without changing its value. Examples of references are: • Raising an exception identifier • Using a type identifier in the declaration of a variable or formal parameter • Using a variable identifier whose type contains fields to access a field. For example, in myrecordvar.myfield := 1, a reference is made to myrecordvar, and an assignment is made to myfield. • Using a cursor for any purpose except FETCH • Passing an identifier to a subprogram by value (IN mode) • Using an identifier as the bind variable in the USING clause of an EXECUTE IMMEDIATE statement in either IN or IN OUT mode An identifier that is passed to a subprogram in IN OUT mode has both a REFERENCE usage (corresponding to IN) and an ASSIGNMENT usage (corresponding to OUT).

15.5.1.3 Identifiers Usage Unique Keys

Every identifier usage is given a numeric ID that is unique within the code unit. This is the USAGE_ID of the identifier.

Each row of a *_IDENTIFIERS view represents a unique usage of an identifier in the PL/SQL unit. In each of these views, these are equivalent unique keys within a compilation unit:

- LINE, COL, and USAGE
- USAGE_ID

> **Note:**
>
> An identifier that is passed to a subprogram in IN OUT mode has two rows in *_IDENTIFIERS: a REFERENCE usage (corresponding to IN) and an ASSIGNMENT usage (corresponding to OUT).

This example shows the USAGE_ID generated for PROCEDURE p1.

```
CREATE OR REPLACE PROCEDURE p1 (a OUT VARCHAR2)
IS
  b VARCHAR2(100) := 'hello FROM p1';
BEGIN
  a := b;
END;
/

SELECT USAGE_ID, USAGE, NAME
FROM ALL_IDENTIFIERS
```

```
WHERE OBJECT_NAME = 'P1'
ORDER BY USAGE_ID;


         1 DECLARATION P1
         2 DEFINITION  P1
         3 DECLARATION A
         4 REFERENCE   VARCHAR2
         5 DECLARATION B
         6 REFERENCE   VARCHAR2
         7 ASSIGNMENT  B
         8 ASSIGNMENT  A
         9 REFERENCE   B
```

> **See Also:**
>
> About Identifiers Usages for the usages in the `*_IDENTIFIERS` views

15.5.1.4 About Identifiers Usage Context

Identifier usages can be contexts for other identifier usages. This creates a one to many parent-child relationship among the usages. The `USAGE_ID` of the parent context usage is the `USAGE_CONTEXT_ID` for the child usages.

Context is useful for discovering relationships between usages. Except for top-level schema object declarations and definitions, every usage of an identifier happens within the context of another usage.

The default top-level context, which contains all top level objects, is identified by a `USAGE_CONTEXT_ID` of 0.

For example:

- A local variable declaration happens within the context of a top-level procedure declaration.
- If an identifier is declared as a variable, such as `x VARCHAR2(10)`, the `USAGE_CONTEXT_ID` of the `VARCHAR2` type reference contains the `USAGE_ID` of the `x` declaration, allowing you to associate the variable declaration with its type.

In other words, `USAGE_CONTEXT_ID` is a reflexive foreign key to `USAGE_ID`, as Example 15-2 shows.

Example 15-2 USAGE_CONTEXT_ID and USAGE_ID

```
ALTER SESSION SET PLSCOPE_SETTINGS = 'IDENTIFIERS:ALL';

CREATE OR REPLACE PROCEDURE a (p1 IN BOOLEAN) AUTHID DEFINER IS
  v PLS_INTEGER;
BEGIN
  v := 42;
  DBMS_OUTPUT.PUT_LINE(v);
  RAISE_APPLICATION_ERROR (-20000, 'Bad');
EXCEPTION
  WHEN Program_Error THEN NULL;
END a;
/
```

```
CREATE or REPLACE PROCEDURE b (
  p2 OUT PLS_INTEGER,
  p3 IN OUT VARCHAR2
)  AUTHID DEFINER
IS
  n NUMBER;
  q BOOLEAN := TRUE;
BEGIN
  FOR j IN 1..5 LOOP
    a(q); a(TRUE); a(TRUE);
    IF j > 2 THEN
       GOTO z;
    END IF;
  END LOOP;
<<z>> DECLARE
  d CONSTANT CHAR(1) := 'X';
  BEGIN
    SELECT COUNT(*) INTO n FROM Dual WHERE Dummy = d;
  END z;
END b;
/
WITH v AS (
  SELECT    Line,
            Col,
            INITCAP(NAME) Name,
            LOWER(TYPE)   Type,
            LOWER(USAGE)  Usage,
            USAGE_ID,
            USAGE_CONTEXT_ID
    FROM USER_IDENTIFIERS
      WHERE Object_Name = 'B'
        AND Object_Type = 'PROCEDURE'
)
SELECT RPAD(LPAD(' ', 2*(Level-1)) ||
                Name, 20, '.')||' '||
                RPAD(Type, 20)||
                RPAD(Usage, 20)
                IDENTIFIER_USAGE_CONTEXTS
  FROM v
  START WITH USAGE_CONTEXT_ID = 0
  CONNECT BY PRIOR USAGE_ID = USAGE_CONTEXT_ID
  ORDER SIBLINGS BY Line, Col
/
```

Result:

```
IDENTIFIER_USAGE_CONTEXTS
-------------------------------------------------------------
B.................. procedure           declaration
  B................ procedure           definition
    P2.............. formal out          declaration
      Pls_Integer... subtype             reference
    P3.............. formal in out       declaration
      Varchar2...... character datatype  reference
    N............... variable            declaration
      Number........ number datatype     reference
    Q............... variable            declaration
```

```
    Q............. variable            assignment
    Boolean....... boolean datatype    reference
  J............... iterator            declaration
    A............. procedure           call
      Q........... variable            reference
    A............. procedure           call
    A............. procedure           call
    J............. iterator            reference
    Z............. label               reference
  Z............... label               declaration
    D............. constant            declaration
      D........... constant            assignment
      Char........ subtype             reference
```

15.5.1.5 About Identifiers Signature

The signature of an identifier is unique within and across program units. That is, the signature distinguishes the identifier from other identifiers with the same name, whether they are defined in the same program unit or different program units.

For the program unit in Example 15-3, which has two identifiers named `p5`, the static data dictionary view `USER_IDENTIFIERS` has several rows in which `NAME` is `p5`, but in these rows, `SIGNATURE` varies. The rows associated with the outer procedure `p5` have one signature, and the rows associated with the inner procedure `p5` have another signature. If program unit `q` calls procedure `p5`, the `USER_IDENTIFIERS` view for `q` has a row in which `NAME` is `p5` and `SIGNATURE` is the signature of the outer procedure `p5`.

Example 15-3 Program Unit with Two Identifiers Named p5

This example shows a program unit with two identifiers named p5 to demonstrate the uniqueness of the signature.

```
CREATE OR REPLACE PROCEDURE p5 AUTHID DEFINER IS
  PROCEDURE p5 IS
  BEGIN
    DBMS_OUTPUT.PUT_LINE('Inner p5');
  END p5;
BEGIN
  DBMS_OUTPUT.PUT_LINE('Outer p5');
  p5();
END p5;
/

SELECT LINE || ' > ' || TEXT
FROM   ALL_SOURCE
WHERE  NAME = 'P5'
   AND TYPE = 'PROCEDURE'
ORDER BY LINE;


1 > PROCEDURE p5 AUTHID DEFINER IS
2 >   PROCEDURE p5 IS
3 >   BEGIN
4 >     DBMS_OUTPUT.PUT_LINE('Inner p5');
5 >   END p5;
6 > BEGIN
7 >   DBMS_OUTPUT.PUT_LINE('Outer p5');
8 >   p5();
```

```
9 > END p5;
```

The following query shows the SIGNATURE for the PL/SQL unit is the same for its DECLARATION and DEFINITION of the inner and outer p5.

```
SELECT SIGNATURE, USAGE, LINE, COL, USAGE_ID, USAGE_CONTEXT_ID
FROM   ALL_IDENTIFIERS
WHERE OBJECT_NAME = 'P5'
ORDER BY LINE, COL, USAGE_ID;


75CD5986BA2EE5C61ACEED8C7162528F DECLARATION         1         11          1       0
75CD5986BA2EE5C61ACEED8C7162528F DEFINITION          1         11          2       1
33FB9F948F526C4B0634C0F35DFA91F6 DECLARATION         2         13          3       2
33FB9F948F526C4B0634C0F35DFA91F6 DEFINITION          2         13          4       3
33FB9F948F526C4B0634C0F35DFA91F6 CALL                8          3          7       2


CREATE OR REPLACE PROCEDURE q AUTHID DEFINER IS
BEGIN
  p5();
END q;
/


EXEC q;
Outer p5
Inner p5


SELECT SIGNATURE, USAGE, LINE, COL, USAGE_ID, USAGE_CONTEXT_ID
FROM   ALL_IDENTIFIERS
WHERE OBJECT_NAME = 'Q'
AND NAME = 'P5'
ORDER BY LINE, COL, USAGE_ID;


75CD5986BA2EE5C61ACEED8C7162528F CALL                3         3          3       2
```

Example 15-4 Find All Usages of VARCHAR2

Identifier signatures are globally unique. This is useful to find all usages of an identifier in all units in the database. This example shows a query to find all usages of VARCHAR2.

```
SELECT UNIQUE OBJECT_NAME uses_varchar2
FROM   ALL_IDENTIFIERS
WHERE  SIGNATURE = (SELECT SIGNATURE
                    FROM   ALL_IDENTIFIERS
                    WHERE  OBJECT_NAME = 'STANDARD'
                           AND OWNER = 'SYS'
                           AND USAGE = 'DECLARATION'
                           AND NAME = 'VARCHAR2')
ORDER  BY OBJECT_NAME;
```

15.5.2 Static Data Dictionary Views for SQL Statements

The DBA_STATEMENTS static dictionary views family describes the SQL statements collected by PL/Scope.

Starting with Oracle Database 12*c* Release 2 (12.2.0.1), a new view, DBA_STATEMENTS, reports on the occurrences of static SQL in PL/SQL units. It provides information about the SQL_ID, the canonical statement text, the statement type, useful statement usage attributes, its signature, and location in the PL/SQL code. Each row represents a SQL statement instance in the PL/SQL code.

Topics:

- SQL Statement Types that PL/Scope Collects
- Statements Location Unique Keys
- About SQL Statement Usage Context
- About SQL Statements Signature

> **See Also:**
>
> *Oracle Database Reference* for more information about the DBA_STATEMENTS view

15.5.2.1 SQL Statement Types that PL/Scope Collects

PL/Scope statement types represent the SQL statements used in PL/SQL.

SQL Statement types correspond to statements that can be used in PL/SQL to execute or otherwise interact with SQL. The statement type appear in the TYPE column of the DBA_STATEMENTS static data dictionary views family.

You must compile the PL/SQL units with the PLSCOPE_SETTINGS='STATEMENTS:ALL' to collect this metadata.

SQL statement types that PL/Scope collects:

- SELECT
- UPDATE
- INSERT
- DELETE
- MERGE
- EXECUTE IMMEDIATE
- SET TRANSACTION
- LOCK TABLE
- COMMIT
- SAVEPOINT
- ROLLBACK
- OPEN
- CLOSE
- FETCH

15.5.2.2 Statements Location Unique Keys

Each row in the DBA_STATEMENTS view represents a unique location of a SQL statement in the PL/SQL unit. This is equivalent to unique keys within a compilation unit.

These following columns are used to determine the location of a statement in the PL/SQL code:

- `OWNER, OBJECT_NAME, OBJECT_TYPE, LINE, COL`
- `USAGE_ID`

The `USAGE_ID` is uniquely defined within a PL/SQL unit. Unlike identifiers, SQL statements do not have different usages, such as `DECLARATION`, `ASSIGNMENT`, or `REFERENCE`. All statements are considered an implicit `CALL` to the sql engine, therefore, the `DBA_STATEMENTS` view does not have the `USAGE` column, but it does use the `USAGE_ID`.

Example 15-5 Using the USAGE_ID Column to Query SQL Identifiers and Statements

```
PROCEDURE p1 (p_cust_id   NUMBER,
              p_cust_name OUT VARCHAR2)
IS
BEGIN
    SELECT (SELECT CUST_FIRST_NAME
            FROM   CUSTOMERS)
    INTO   p_cust_name
    FROM   CUSTOMERS
    WHERE  CUSTOMER_ID = p_cust_id;
END;

SELECT USAGE_ID, TYPE, NAME, USAGE, LINE, COL
FROM ( SELECT USAGE_ID, TYPE, NAME, USAGE, LINE, COL
       FROM ALL_IDENTIFIERS
       WHERE OBJECT_NAME = 'P1'
       UNION
       SELECT USAGE_ID, TYPE, 'SQL STATEMENT', " ", LINE, COL
       FROM ALL_STATEMENTS
       WHERE OBJECT_NAME = 'P1')
ORDER BY USAGE_ID;

USAGE_ID TYPE               NAME            USAGE        LINE   COL
---------- ----------------- --------------- ----------- ----- -----
         1 PROCEDURE         P1              DECLARATION     1    11
         2 PROCEDURE         P1              DEFINITION      1    11
         3 FORMAL IN         P_CUST_ID       DECLARATION     1    15
         4 NUMBER DATATYPE   NUMBER          REFERENCE       1    25
         5 FORMAL OUT        P_CUST_NAME     DECLARATION     1    33
         6 CHARACTER DATATYPE VARCHAR2       REFERENCE       1    49
         7 SQL STATEMENT     SELECT                          3     3
         8 TABLE             CUSTOMERS       REFERENCE       4    10
         9 FORMAL IN         P_CUST_ID       REFERENCE       4    38
        10 COLUMN            CUSTOMER_ID     REFERENCE       4    26
        11 FORMAL OUT        P_CUST_NAME     ASSIGNMENT      3    31
        12 COLUMN            CUST_FIRST_NAME REFERENCE       3    10
```

15.5.2.3 About SQL Statement Usage Context

Statements can act as a context for other statements or identifiers. Statements can also be in the context of other statements or identifiers.

The `USAGE_CONTEXT_ID` column is used to determine the context of the statement. All identifiers appearing within a statement will be in the context of that statement.

Expressions and nested subqueries are not supported as assignment sources.

Example 15-6 Using DBA_STATEMENTS USAGE_CONTEXT_ID to Query Identifiers

This example shows how to retrieve the identifiers in the context of the `SELECT` statement using the `USAGE_CONTEXT_ID` column.

```
PROCEDURE p1 (p_cust_id   NUMBER,
              p_cust_name OUT VARCHAR2)
IS
BEGIN
    SELECT (SELECT CUST_FIRST_NAME
            FROM   CUSTOMERS)
    INTO   p_cust_name
    FROM   CUSTOMERS
    WHERE  CUSTOMER_ID = p_cust_id;
END;

SELECT USAGE_ID, LPAD(' ', 2*(level-1)) || TO_CHAR(USAGE) || ' ' || NAME
usages, LINE, COL
FROM ( SELECT OBJECT_NAME, USAGE, USAGE_ID, USAGE_CONTEXT_ID, NAME, LINE, COL
       FROM ALL_IDENTIFIERS
       WHERE OBJECT_NAME = 'P1'
       UNION
       SELECT OBJECT_NAME, TYPE usage, USAGE_ID, USAGE_CONTEXT_ID,
'Statement' name, LINE, COL
       FROM ALL_STATEMENTS
       WHERE OBJECT_NAME = 'P1'
      )
      START WITH USAGE_CONTEXT_ID = 0 AND OBJECT_NAME = 'P1'
      CONNECT BY PRIOR USAGE_ID = USAGE_CONTEXT_ID AND OBJECT_NAME = 'P1';


  USAGE_ID USAGES                                    LINE        COL
---------- ---------------------------------- ---------- ----------
         1 DECLARATION P1                              1         11
         2   DEFINITION P1                             1         11
         3     DECLARATION P_CUST_ID                   1         15
         4       REFERENCE NUMBER                      1         27
         5     DECLARATION P_CUST_NAME                 2         33
         6       REFERENCE VARCHAR2                    2         49
         7     SELECT STATEMENT                        5          5
         8       REFERENCE CUSTOMERS                   8         12
         9       REFERENCE P_CUST_ID                   9         26
        10       REFERENCE CUSTOMER_ID                 9         12
        11       REFERENCE CUSTOMERS                   6         20
        12       REFERENCE CUST_FIRST_NAME             5         20
        13       ASSIGNMENT P_CUST_NAME                7         12
```

15.5.2.4 About SQL Statements Signature

Every SQL statement has a unique PL/Scope signature that identifies that instance of the statement in all the PL/SQL units.

The SQL statement signature distinguishes the call from a PL/SQL unit for the SQL with the same SQL_ID from another call from a different PL/SQL unit.

Nested subqueries are not individual SQL statements in ALL_STATEMENTS.

Example 15-7 Distinct SQL Signatures for the Same SQL Statement when Called from Different PL/SQL Units

This example shows two distinct signatures for the same SQL statement when it is called from PROCEDURE p1 and p2. You can observe the nested subquery is not assigned a distinct SQL_ID, therefore is not an individual SQL statements in ALL_STATEMENTS.

```
CREATE OR REPLACE PROCEDURE p1 (p_cust_id   NUMBER,
                                p_cust_name OUT VARCHAR2)
IS
BEGIN
    SELECT (SELECT CUST_FIRST_NAME
            FROM   CUSTOMERS)
    INTO   p_cust_name
    FROM   CUSTOMERS
    WHERE  CUSTOMER_ID = p_cust_id;
END;
/
CREATE OR REPLACE PROCEDURE P2 (p_cust_id   NUMBER,
                                p_cust_name OUT VARCHAR2)
IS
BEGIN
    SELECT (SELECT CUST_FIRST_NAME
            FROM   CUSTOMERS)
    INTO   p_cust_name
    FROM   CUSTOMERS
    WHERE  CUSTOMER_ID = p_cust_id;
END;
/

ACCEPT nam CHAR PROMPT "Enter OBJECT_NAME : "

SELECT *
FROM   ALL_STATEMENTS
WHERE OBJECT_NAME = '&&nam'
ORDER BY LINE, COL;
```

Select `ALL_STATEMENTS` for P1 and P2 to observe the different SQL signatures for the same `SQL_ID`.

```
new   3: WHERE OBJECT_NAME = 'P1'
OE
138835D3A2EBBA76A7A064E4DC14B466 SELECT
P1
PROCEDURE            7          5          5               2 c02b6yppqb46p NO  NO  NO
```

```
NO  NO  NO
YES SELECT (SELECT CUST_FIRST_NAME FROM CUSTOMERS) FROM CUSTOMERS WHERE CUSTOMER_ID
= :B1
SELECT (SELECT CUST_FIRST_NAME FROM CUSTOMERS) FROM CUSTOMERS WHERE CUSTOMER_ID      0


new   3: WHERE OBJECT_NAME = 'P2'
OE
E6A5E27E5E90997A169C5C25393FAB35 SELECT
P2
PROCEDURE             7          5          5                2 c02b6yppqb46p NO  NO  NO
NO  NO  NO
YES SELECT (SELECT CUST_FIRST_NAME FROM CUSTOMERS) FROM CUSTOMERS WHERE CUSTOMER_ID
= :B1
SELECT (SELECT CUST_FIRST_NAME FROM CUSTOMERS) FROM CUSTOMERS WHERE CUSTOMER_ID      0
```

15.5.3 SQL Developer

PL/Scope is a feature of SQL Developer. For information about using PL/Scope from SQL Developer, see SQL Developer online help

> **See Also:**
>
> *Oracle SQL Developer User's Guide*

15.6 Overview of Data Dictionary Views Useful to Manage PL/SQL Code

In addition to the PL/Scope data dictionary views, the following static dictionary views are most useful for PL/SQL programmers and are most often referenced in queries related to PL/SQL code management reports. This is not an exhaustive list of all static data dictionary views.

Summary of the Data Dictionary Views Useful to Manage PL/SQL Code

View Name	Description
ALL_ARGUMENTS	Lists the arguments of the functions and procedures that are accessible to the current user
ALL_DEPENDENCIES	Describes dependencies between procedures, packages, functions, package bodies, and triggers accessible to the current user
ALL_ERRORS	Describes the current errors on the stored objects accessible to the current user
ALL_IDENTIFIERS	Displays information about the identifiers in the stored objects accessible to the current user
USER_OBJECT_SIZE	Describes the size, in bytes, of PL/SQL objects owned by the current user. Although this information is meant to be used by the compiler and runtime engine, you can use it to identify the large programs in your environment.
ALL_OBJECTS	Describes all objects accessible to the current user
ALL_PLSQL_OBJECT_SETTINGS	Displays information about the compiler settings for the stored objects accessible to the current user

View Name	Description
ALL_PROCEDURES	Describes all PL/SQL functions and procedures, along with associated properties, that are accessible to the current user
ALL_SEQUENCES	Describes the sequences accessible to the current user
ALL_SOURCE	Describes the text source of the stored objects accessible to the current user
ALL_STATEMENTS	Describes all SQL statements in stored PL/SQL objects accessible to the user
ALL_STORED_SETTINGS	Describes the persistent parameter settings for stored PL/SQL units for which the current user has execute privileges
ALL_SYNONYMS	Describes the synonyms accessible to the current user
ALL_TAB_COLUMNS	Describes the columns of the tables, views, and clusters accessible to the current user
ALL_TABLES	Describes the relational tables accessible to the current user
ALL_TRIGGERS	Describes the triggers on tables accessible to the current user
ALL_VIEWS	Describes the views accessible to the current user

15.7 Sample PL/Scope Session

In this sample session, assume that you are logged in as `HR`.

1. Set the session parameter:

```
ALTER SESSION SET PLSCOPE_SETTINGS='IDENTIFIERS:ALL';
```

2. Create this package:

```
CREATE OR REPLACE PACKAGE pack1 AUTHID DEFINER IS
  TYPE r1 is RECORD (rf1 VARCHAR2(10));
  FUNCTION f1(fp1 NUMBER) RETURN NUMBER;
  PROCEDURE p1(pp1 VARCHAR2);
END PACK1;
/
CREATE OR REPLACE PACKAGE BODY pack1 IS
  FUNCTION f1(fp1 NUMBER) RETURN NUMBER IS
    a NUMBER := 10;
  BEGIN
    RETURN a;
  END f1;
  PROCEDURE p1(pp1 VARCHAR2) IS
    pr1 r1;
  BEGIN
    pr1.rf1 := pp1;
  END;
END pack1;
/
```

3. Verify that PL/Scope collected all identifiers for the package body:

```
SELECT PLSCOPE_SETTINGS
FROM USER_PLSQL_OBJECT_SETTINGS
WHERE NAME='PACK1' AND TYPE='PACKAGE BODY'
```

Result:

```
PLSCOPE_SETTINGS
------------------------------------------------------------------------
IDENTIFIERS:ALL
```

4. Display unique identifiers in `HR` by querying for all `DECLARATION` usages. For example, to see all unique identifiers with name like `%1`, use these SQL*Plus formatting commands and this query:

```
COLUMN NAME FORMAT A6
COLUMN SIGNATURE FORMAT A32
COLUMN TYPE FORMAT A9

SELECT NAME, SIGNATURE, TYPE
FROM USER_IDENTIFIERS
WHERE NAME LIKE '%1' AND USAGE='DECLARATION'
ORDER BY OBJECT_TYPE, USAGE_ID;
```

Result is similar to:

```
NAME   SIGNATURE                        TYPE
------ -------------------------------- ---------
PACK1  41820FA4D5EF6BE707895178D0C5C4EF PACKAGE
R1     EEBB6849DEE31BC77BF186EBAE5D4E2D RECORD
RF1    41D70040337349634A7F547BC83517C7 VARIABLE
F1     D51E825FF334817C977174423E3D0765 FUNCTION
FP1    CAC3474C112DBEC67AB926978D9A16C1 FORMAL IN
P1     B7C0576BA4D00C33A65CC0C64CADAB89 PROCEDURE
PP1    6B74CF95A5B7377A735925DFAA280266 FORMAL IN
FP1    98EB63B8A4AFEB5EF94D50A20165D6CC FORMAL IN
PP1    62D8463A314BE1F996794723402278CF FORMAL IN
PR1    BDB1CB26C97562CCC20CD1F32D341D7C VARIABLE

10 rows selected.
```

The `*_IDENTIFIERS` static data dictionary views display only basic type names; for example, the `TYPE` of a local variable or record field is `VARIABLE`. To determine the exact type of a `VARIABLE`, you must use its `USAGE_CONTEXT_ID`.

5. Find all local variables:

```
COLUMN VARIABLE_NAME FORMAT A13
COLUMN CONTEXT_NAME FORMAT A12

SELECT a.NAME variable_name,
       b.NAME context_name,
       a.SIGNATURE
FROM USER_IDENTIFIERS a, USER_IDENTIFIERS b
WHERE a.USAGE_CONTEXT_ID = b.USAGE_ID
AND a.TYPE = 'VARIABLE'
AND a.USAGE = 'DECLARATION'
AND a.OBJECT_NAME = 'PACK1'
AND a.OBJECT_NAME = b.OBJECT_NAME
AND a.OBJECT_TYPE =  b.OBJECT_TYPE
AND (b.TYPE = 'FUNCTION' or b.TYPE = 'PROCEDURE')
ORDER BY a.OBJECT_TYPE, a.USAGE_ID;
```

Result is similar to:

```
VARIABLE_NAME CONTEXT_NAME SIGNATURE
------------- ------------ --------------------------------
A             F1           1691C6B3C951FCAA2CBEEB47F85CF128
PR1           P1           BDB1CB26C97562CCC20CD1F32D341D7C

2 rows selected.
```

6. Find all usages performed on the local variable `A`:

```
COLUMN USAGE FORMAT A11
COLUMN USAGE_ID FORMAT 999
COLUMN OBJECT_NAME FORMAT A11
COLUMN OBJECT_TYPE FORMAT A12

SELECT USAGE, USAGE_ID, OBJECT_NAME, OBJECT_TYPE
FROM USER_IDENTIFIERS
WHERE SIGNATURE='1691C6B3C951FCAA2CBEEB47F85CF128'  -- signature of A
ORDER BY OBJECT_TYPE, USAGE_ID;
```

Result:

```
USAGE       USAGE_ID OBJECT_NAME OBJECT_TYPE
----------- -------- ----------- ------------
DECLARATION        6 PACK1       PACKAGE BODY
ASSIGNMENT         8 PACK1       PACKAGE BODY
REFERENCE          9 PACK1       PACKAGE BODY

3 rows selected.
```

The usages performed on the local identifier `A` are the identifier declaration (`USAGE_ID 6`), an assignment (`USAGE_ID 8`), and a reference (`USAGE_ID 9`).

7. From the declaration of the local identifier `A`, find its type:

```
COLUMN NAME FORMAT A6
COLUMN TYPE FORMAT A15

SELECT a.NAME, a.TYPE
FROM USER_IDENTIFIERS a, USER_IDENTIFIERS b
WHERE a.USAGE = 'REFERENCE'
AND a.USAGE_CONTEXT_ID = b.USAGE_ID
AND b.USAGE = 'DECLARATION'
AND b.SIGNATURE = 'D51E825FF334817C977174423E3D0765'  -- signature of F1
AND a.OBJECT_TYPE = b.OBJECT_TYPE
AND a.OBJECT_NAME = b.OBJECT_NAME;
```

Result:

```
NAME   TYPE
------ ---------------
NUMBER NUMBER DATATYPE

1 row selected.
```

8. Find out where the assignment to local identifier `A` occurred:

```
SELECT LINE, COL, OBJECT_NAME, OBJECT_TYPE
FROM USER_IDENTIFIERS
WHERE SIGNATURE='1691C6B3C951FCAA2CBEEB47F85CF128'  -- signature of A
AND USAGE='ASSIGNMENT';
```

Result:

```
      LINE        COL OBJECT_NAME OBJECT_TYPE
---------- ---------- ----------- ------------
         3          5 PACK1       PACKAGE BODY

1 row selected.
```