

SQL Lineage Made Easy with PL/Scope 12.2?

Philipp Salvisberg



@phsalvisberg

BASEL ■ BERN ■ BRUGG ■ DÜSSELDORF ■ FRANKFURT A.M. ■ FREIBURG I.B.R. ■ GENÈVE
HAMBURG ■ KOPENHAGEN ■ LAUSANNE ■ MÜNCHEN ■ STUTTGART ■ WIEN ■ ZÜRICH

trivadis
makes IT easier. ■ ■ ■

■ About Me

- Trivadian since April 2000
 - Senior Principal Consultant, Partner
 - Member of the Board of Directors
 - www.salvis.com/blog
 - [@phsalvisberg](https://twitter.com/phsalvisberg)
- Database centric development with Oracle database
- Fond of DSLs to build full stack solutions efficiently and keep them manageable
- Author of free SQL Developer Extensions PL/SQL Cop, PL/SQL Unwrapper, oddgen and Bitemp Remodeler



■ Agenda

1. Data Lineage
2. PL/Scope
3. Data Lineage with PL/Scope
4. Options
5. Core Messages

Data Lineage

■ What Is Data Lineage?

Data lineage is generally defined as a kind of data life cycle that includes the **data's origins and where it moves** over time.

This term can also describe what happens to data as it goes through diverse processes.

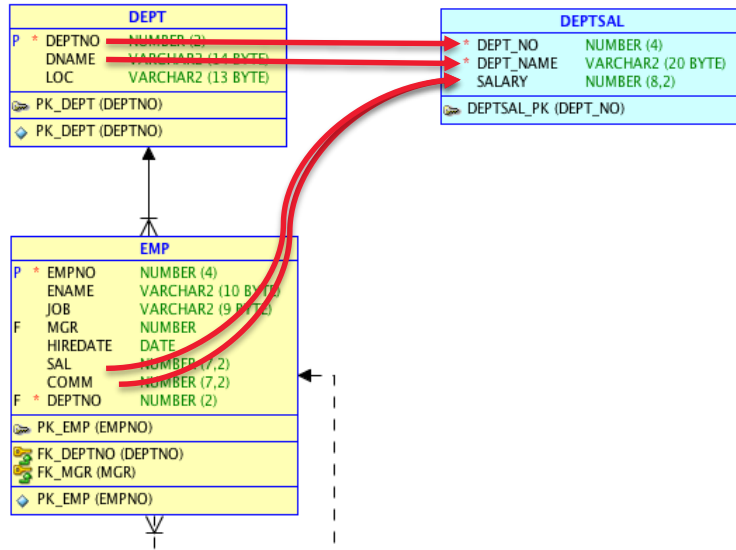
Source: <https://www.techopedia.com/definition/28040/data-lineage>

■ Example

```
INSERT INTO deptsal (dept_no, dept_name, salary)
SELECT d.deptno, d.dname, SUM(e.sal + NVL(e.comm, 0)) AS sal
FROM dept d
LEFT JOIN (SELECT *
            FROM emp
            WHERE hiredate > DATE '1980-01-01') e
ON e.deptno = d.deptno
GROUP BY d.deptno, d.dname;
```

Where Did the Output Come From?

Schema Level

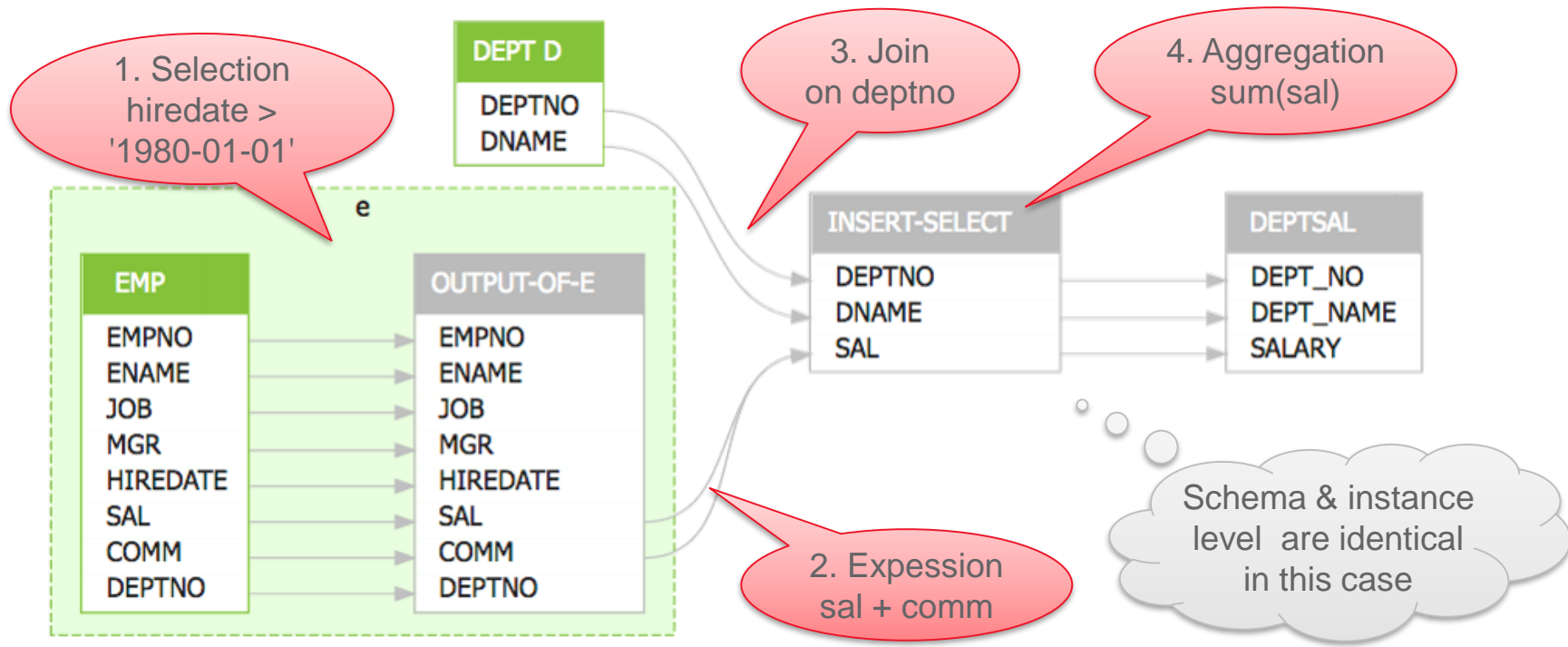


Instance Level

| DEPTNO | DNAME | DEPT... | DEPT_NAME | SALARY |
|--------|------------|---------|------------|--------|
| 10 | ACCOUNTING | 10 | ACCOUNTING | 8750 |
| 20 | RESEARCH | 20 | RESEARCH | 10875 |
| 30 | SALES | 30 | SALES | 11600 |
| 40 | OPERATIONS | 40 | OPERATIONS | (null) |

| DEPTNO | ENAME | SAL | COMM |
|--------|--------|------|--------|
| 10 | CLARK | 2450 | (null) |
| 10 | KING | 5000 | (null) |
| 10 | MILLER | 1300 | (null) |
| 20 | ADAMS | 1100 | (null) |
| 20 | FORD | 3000 | (null) |
| 20 | JONES | 2975 | (null) |
| 20 | SCOTT | 3000 | (null) |

■ How Were the Inputs Manipulated to Produce the Output?



PL/Scope

■ What Is 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 **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.

Source: Oracle Database Developer's Guide 12c Release 2 (12.2) E49632-15, January 2017

■ Using PL/Scope

1. Enable (on session or system level, disabled by default, check IDE settings as well)

```
ALTER SESSION SET plscope_settings='identifiers:all, statements:all';
```

2. Compile schemas of interest (package, procedure, function, type, trigger, synonym)
3. Query views (uncompiled, referenced objects are suppressed)

```
SELECT * FROM all_identifiers;  
SELECT * FROM all_statements;
```

4. Build your code analysis utilities (views and packages)
see also <https://github.com/PhilippSalvisberg/plscope-utils>

■ Example – Identifiers & Statements



```
CREATE OR REPLACE PROCEDURE load_from_tab IS
BEGIN
  INSERT INTO deptsal (dept_no, dept_name, salary)
  SELECT /*+ordered */
        d.deptno, d.dname, SUM(e.sal + NVL(e.comm, 0)) AS sal
  FROM dept d
  LEFT JOIN (SELECT *
              FROM emp
              WHERE hiredate > DATE '1980-01-01') e
        ON e.deptno = d.deptno
  GROUP BY d.deptno, d.dname;
  COMMIT;
END load_from_tab;
```

```

1 SELECT line, col, name, name_path, type, usage, ref_owner, ref_object_type, ref_object_name
2 FROM plscope_identifiers
3 WHERE object_name = 'LOAD_FROM_TAB'
4 AND owner = USER
5 ORDER BY line, col;

```

Script Output x Query Result x

SQL | All Rows Fetched: 19 in 0.063 seconds

| LINE | COL | NAME | NAME_PATH | TYPE | USAGE | REF_OWNER | REF_OBJECT_TYPE | REF_OBJECT_NAME |
|------|-----|------------------|--|-----------|-------------|-----------|-----------------|-----------------|
| 1 | 1 | 11 LOAD_FROM_TAB | /LOAD_FROM_TAB | PROCEDURE | DECLARATION | PLSCOPE | PROCEDURE | LOAD_FROM_TAB |
| 2 | 1 | 11 LOAD_FROM_TAB | /LOAD_FROM_TAB/LOAD_FROM_TAB | PROCEDURE | DEFINITION | PLSCOPE | PROCEDURE | LOAD_FROM_TAB |
| 3 | 3 | 4 3nyyhcpmwxcgz | /LOAD_FROM_TAB/LOAD_FROM_TAB/3nyyhcpmwxcgz | INSERT | EXECUTE | (null) | (null) | (null) |
| 4 | 3 | 16 DEPTSAL | /LOAD_FROM_TAB/LOAD_FROM_TAB/3nyyhcpmwxcgz/DEPTSAL | TABLE | REFERENCE | PLSCOPE | TABLE | DEPTSAL |
| 5 | 3 | 25 DEPT_NO | /LOAD_FROM_TAB/LOAD_FROM_TAB/3nyyhcpmwxcgz/DEPT_NO | COLUMN | REFERENCE | PLSCOPE | TABLE | DEPTSAL |
| 6 | 3 | 34 DEPT_NAME | /LOAD_FROM_TAB/LOAD_FROM_TAB/3nyyhcpmwxcgz/DEPT_NAME | COLUMN | REFERENCE | PLSCOPE | TABLE | DEPTSAL |
| 7 | 3 | 45 SALARY | /LOAD_FROM_TAB/LOAD_FROM_TAB/3nyyhcpmwxcgz/SALARY | COLUMN | REFERENCE | PLSCOPE | TABLE | DEPTSAL |
| 8 | 5 | 13 DEPTNO | /LOAD_FROM_TAB/LOAD_FROM_TAB/3nyyhcpmwxcgz/DEPTNO | COLUMN | REFERENCE | PLSCOPE | TABLE | DEPT |
| 9 | 5 | 23 DNAME | /LOAD_FROM_TAB/LOAD_FROM_TAB/3nyyhcpmwxcgz/DNAME | COLUMN | REFERENCE | PLSCOPE | TABLE | DEPT |
| 10 | 5 | 36 SAL | /LOAD_FROM_TAB/LOAD_FROM_TAB/3nyyhcpmwxcgz/SAL | COLUMN | REFERENCE | PLSCOPE | TABLE | EMP |
| 11 | 5 | 48 COMM | /LOAD_FROM_TAB/LOAD_FROM_TAB/3nyyhcpmwxcgz/COMM | COLUMN | REFERENCE | PLSCOPE | TABLE | EMP |
| 12 | 6 | 11 DEPT | /LOAD_FROM_TAB/LOAD_FROM_TAB/3nyyhcpmwxcgz/DEPT | TABLE | REFERENCE | PLSCOPE | TABLE | DEPT |
| 13 | 8 | 24 EMP | /LOAD_FROM_TAB/LOAD_FROM_TAB/3nyyhcpmwxcgz/EMP | TABLE | REFERENCE | PLSCOPE | TABLE | EMP |
| 14 | 9 | 24 HIREDATE | /LOAD_FROM_TAB/LOAD_FROM_TAB/3nyyhcpmwxcgz/HIREDATE | COLUMN | REFERENCE | PLSCOPE | TABLE | EMP |
| 15 | 10 | 10 DEPTNO | /LOAD_FROM_TAB/LOAD_FROM_TAB/3nyyhcpmwxcgz/DEPTNO | COLUMN | REFERENCE | PLSCOPE | TABLE | EMP |
| 16 | 10 | 21 DEPTNO | /LOAD_FROM_TAB/LOAD_FROM_TAB/3nyyhcpmwxcgz/DEPTNO | COLUMN | REFERENCE | PLSCOPE | TABLE | DEPT |
| 17 | 11 | 16 DEPTNO | /LOAD_FROM_TAB/LOAD_FROM_TAB/3nyyhcpmwxcgz/DEPTNO | COLUMN | REFERENCE | PLSCOPE | TABLE | DEPT |
| 18 | 11 | 26 DNAME | /LOAD_FROM_TAB/LOAD_FROM_TAB/3nyyhcpmwxcgz/DNAME | COLUMN | REFERENCE | PLSCOPE | TABLE | DEPT |
| 19 | 12 | 4 COMMIT | /LOAD_FROM_TAB/LOAD_FROM_TAB/COMMIT | COMMIT | EXECUTE | (null) | (null) | (null) |

Data Lineage With PL/Scope

■ Where-Lineage On Schema-Level of ...

```
CREATE OR REPLACE PROCEDURE load_from_tab IS
BEGIN
    INSERT INTO deptsal (dept_no, dept_name, salary)
    SELECT /*+ordered */
           d.deptno, d.dname, SUM(e.sal + NVL(e.comm, 0)) AS sal
    FROM dept d
    LEFT JOIN (SELECT *
               FROM emp
               WHERE hiredate > DATE '1980-01-01') e
              ON e.deptno = d.deptno
    GROUP BY d.deptno, d.dname;
COMMIT;
END load_from_tab;
```

■ ... Needs a Parser to Produce this Result

| Line | Col | From Owner | From Object Type | From Object Name | From Column Name | To Owner | To Object Type | To Object Name | To Column Name |
|------|-----|------------|------------------|------------------|------------------|----------|----------------|----------------|----------------|
| 3 | 4 | PLSCOPE | TABLE | DEPT | DEPTNO | PLSCOPE | TABLE | DEPTSAL | DEPT_NO |
| 3 | 4 | PLSCOPE | TABLE | DEPT | DNAME | PLSCOPE | TABLE | DEPTSAL | DEPT_NAME |
| 3 | 4 | PLSCOPE | TABLE | EMP | SAL | PLSCOPE | TABLE | DEPTSAL | SALARY |
| 3 | 4 | PLSCOPE | TABLE | EMP | COMM | PLSCOPE | TABLE | DEPTSAL | SALARY |

■ SYS.UTL_XML.ParseQuery to the Rescue

```
SELECT /*+ordered */
      d.deptno, d.dname,
      SUM(e.sal + NVL(e.comm, 0)) AS sal
FROM dept d
LEFT JOIN (SELECT *
           FROM emp
           WHERE hiredate >
             DATE '1980-01-01') e
ON e.deptno = d.deptno
GROUP BY d.deptno, d.dname;
```

produces

```
<QUERY> (...)
  <SELECT_LIST_ITEM>
    <SUM>
      <ADD>
        <COLUMN_REF>
          <TABLE_ALIAS>E</TABLE_ALIAS>
          <COLUMN>SAL</COLUMN>
        </COLUMN_REF>
        <NVL>
          <COLUMN_REF>
            <TABLE_ALIAS>E</TABLE_ALIAS>
            <COLUMN>COMM</COLUMN>
          </COLUMN_REF>
          <LITERAL>0</LITERAL>
        </NVL>
      </ADD>
    </SUM>
    <COLUMN_ALIAS>SAL</COLUMN_ALIAS>
  </SELECT_LIST_ITEM> (...)
</QUERY>
```

■ PL/Scope Provides an INSERT Statement

```
INSERT INTO deptsal
    (dept_no, dept_name, salary)
SELECT /*+ordered */
    d.deptno, d.dname,
    SUM(e.sal + NVL(e.comm, 0)) AS sal
FROM dept d
LEFT JOIN (SELECT *
            FROM emp
            WHERE hiredate >
                DATE '1980-01-01') e
    ON e.deptno = d.deptno
GROUP BY d.deptno, d.dname;
```

produces

```
<QUERY>
  <SELECT>
    <SELECT_LIST/>
  </SELECT>
  <FROM>
    <FROM_ITEM>
      <TABLE>DEPTSAL</TABLE>
    </FROM_ITEM>
  </FROM>
</QUERY>
```

■ That's All We Need



Don't talk. Just act.
Don't say. Just show.
Don't promise. Just prove.



Where-Lineage in SQL Developer

Worksheet Query Builder

```
1 SELECT line, col,  
2     from_owner, from_object_type, from_object_name, from_column_name,  
3     to_owner, to_object_type, to_object_name, to_column_name  
4 FROM plscope_ins_lineage  
5 WHERE object_name = 'LOAD_FROM_TAB'  
6 ORDER BY to_column_name;
```

Query Result x

SQL | All Rows Fetched: 4 in 0.842 seconds

| | LINE | COL | FROM_OWNER | FROM_OBJEC... | FROM_OBJE... | FROM_CO... | TO_OWNER | TO_OBJECT_TYPE | TO_OBJECT_NAME | TO_COLUMN_NAME |
|---|------|-----|------------|---------------|--------------|------------|----------|----------------|----------------|----------------|
| 1 | 3 | 4 | PLSCOPE | TABLE | DEPT | DNAME | PLSCOPE | TABLE | DEPTSAL | DEPT_NAME |
| 2 | 3 | 4 | PLSCOPE | TABLE | DEPT | DEPTNO | PLSCOPE | TABLE | DEPTSAL | DEPT_NO |
| 3 | 3 | 4 | PLSCOPE | TABLE | EMP | COMM | PLSCOPE | TABLE | DEPTSAL | SALARY |
| 4 | 3 | 4 | PLSCOPE | TABLE | EMP | SAL | PLSCOPE | TABLE | DEPTSAL | SALARY |

■ Today's Where-Lineage by plsscope-utils

In scope

- INSERT ... SELECT statements
 - single_table_insert
 - multi_table_insert
 - with_clause
 - error_logging_clause
 - Views as source or target
 - Synonyms as source or target
 - Insert without column-list
 - Qualified all-column wildcard

Out of scope

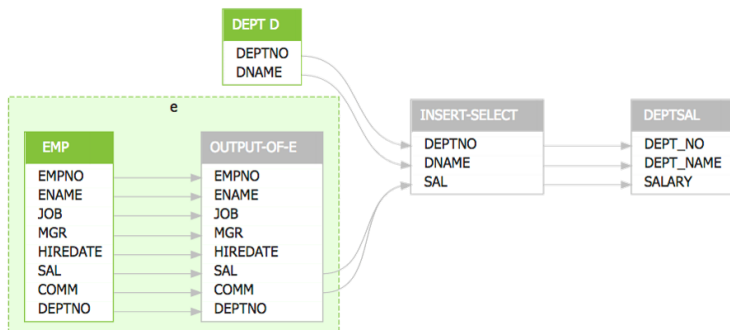
- UPDATE
- MERGE
- INSERT
 - values_clause
 - Explicit or implicit cursors
 - PL/SQL transformations (functions, procedures, triggers)
 - Database links
- Dynamic SQL
- Specialities (pivot, unpivot, XML, ...)

Options

SQL Lineage Tools

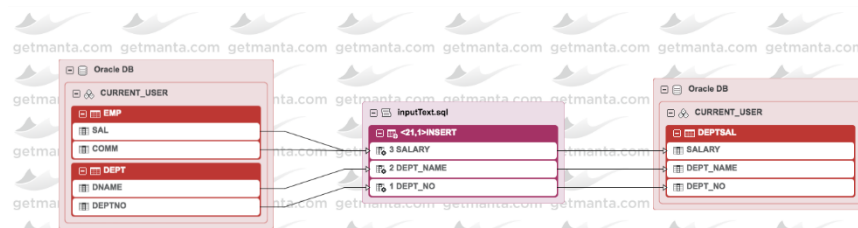
SQLDep

■ <https://sqldep.com/>



MANTA FLOW

■ <https://getmanta.com/>



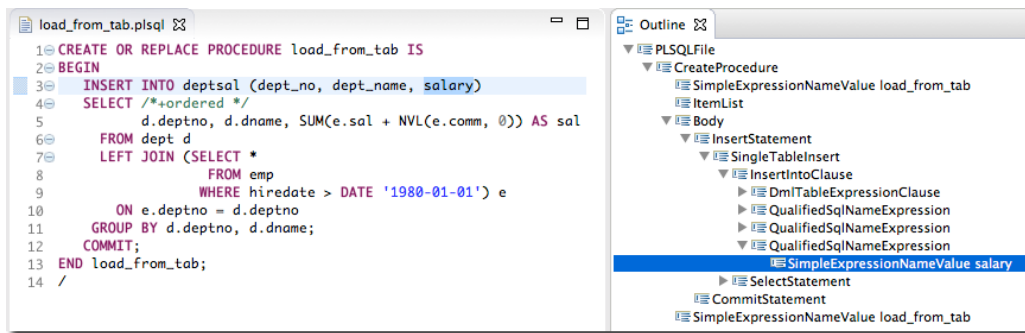
■ Third Party SQL & PL/SQL Parsers

General SQL Parser

- <http://www.sqlparser.com/>
- <http://www.sqlparser.com/determining-impact-data-lineage.php>

PL/SQL Cop or PL/SQL Analyzer

- <https://www.trivadis.com/en/plsql-cop>
- <https://www.salvis.com/blog/plsql-cop/>
- <https://www.salvis.com/blog/tvdca-trivadis-plsql-sql-codeanalyzer/>



■ Semantic Analysis by 3rd Party Products

```
INSERT INTO deptsal
    (dept_no, dept_name, salary)
SELECT /*+ordered */
    d.deptno, dname,
    SUM(sal + NVL(comm, 0)) AS sal
FROM dept d
LEFT JOIN (SELECT *
            FROM emp
            WHERE hiredate >
                DATE '1980-01-01') e
    ON e.deptno = d.deptno
GROUP BY d.deptno, dname;
```

- Source of unqualified columns?
(dname, sal, comm)
- Details about data model required
- Challenges
 - Multi-schema applications
 - Parsing user
 - Definer/invoker rights
 - Synonyms (private, public)
 - Session settings (container, schema)

Core Messages

■ Data Lineage with Oracle DB 12.2 – Possible

- SQL statements collected by PL/Scope 12.2 are the key for various analysis
- You need a parser for data lineage
- Parse-tree produced by SYS.UTL_XML.ParseQuery is helpful, but incomplete
 - Perfect for SELECT
 - 12.2 grammar support
 - Subset for INSERT, UPDATE, DELETE and MERGE
 - No PL/SQL (nowhere)
- Requires 3rd party tools



Questions and Answers

Philipp Salvisberg
Senior Principal Consultant

Tel. +41 58 459 52 31
philipp.salvisberg@trivadis.com

