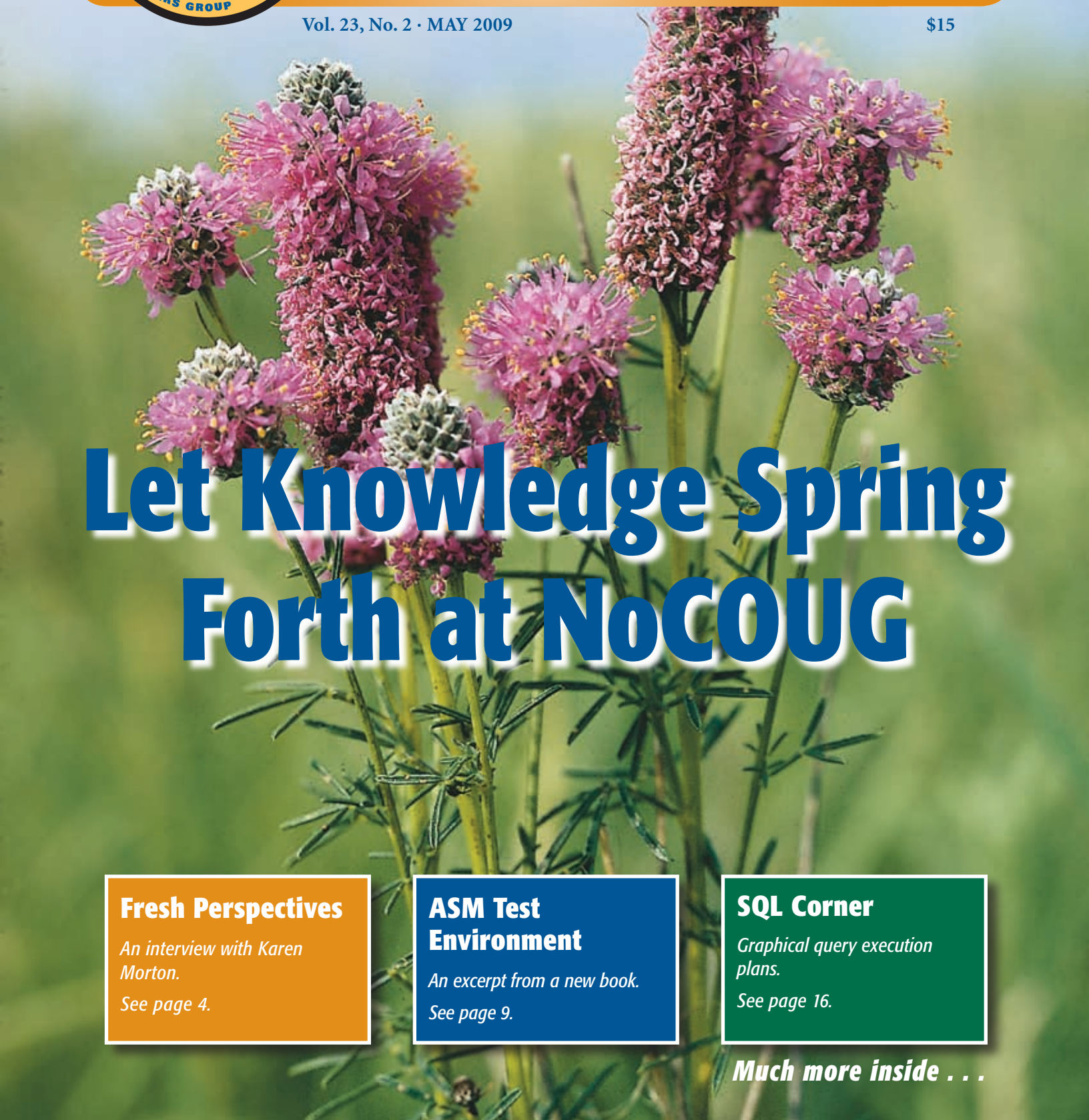
Official Publication of the Northern California Oracle Users Group

NoCOUG JOURNAL

Vol. 23, No. 2 · MAY 2009 $15



Let Knowledge Spring Forth at NoCOUG

Much more inside . . .

One Picture Is Worth Ten Thousand Words!

by Iggy Fernandez

Chris Lawson

I have to admit that I find tabular query execution plans—such as those produced by DBMS_XPLAN—not very easy to read, especially when many tables are involved. An example of a tabular plan is shown at the bottom of this page. The hierarchical relationship between the steps is expressed by varying the indentation level ever so slightly, but I find this hard to follow. (The last time I was trying to make sense of such a plan, a colleague suggested that I use a sheet of paper as a makeshift ruler.) I also find it hard to determine the order in which the steps are executed. Another problem is that the elapsed execution times that are listed in the plans are cumulative; this makes it difficult to identify the time-consuming steps.

A graphical query plan such as the one shown on the next page is much easier to read. The PL/SQL code that produced it is shown in the following pages. It produces commands—in the "dot" language—for a graphing tool called Graphviz that can be downloaded from **www.graphviz.org**. Here is an example; it shows abbreviated versions of the commands needed to produce the graph on the next page.

```
digraph a {
"5" [label="Step 1\nDIM_E",shape=plaintext]
"7" [label="Step 2\nDIM_D",shape=plaintext]
"12" [label="Step 3\nDIM_A",shape=plaintext]
"14" [label="Step 4\nIDX_DIM_B_1",shape=plaintext]
"13" [label="Step 5\nDIM_B",shape=plaintext]
"11" [label="Step 6\nNESTED LOOPS",shape=plaintext]
"15" [label="Step 7\nIDX_DIM_C_1",shape=plaintext]
"10" [label="Step 8\nNESTED LOOPS",shape=plaintext]
"9" [label="Step 9\nDIM_C",shape=plaintext]
"16" [label="Step 10\nFACT",shape=plaintext]
"8" [label="Step 11\nHASH JOIN",shape=plaintext]
"6" [label="Step 12\nHASH JOIN RIGHT OUTER",shape=plaintext]
"4" [label="Step 13\nHASH JOIN RIGHT OUTER",shape=plaintext]
"3" [label="Step 14\nFILTER",shape=plaintext]
"2" [label="Step 15\nHASH GROUP BY",shape=plaintext]
"1" [label="Step 16\nSORT ORDER BY",shape=plaintext]
```

```
"1"->"";
"2"->"1";
"3"->"2";
"4"->"3";
"5"->"4";
"6"->"4";
"7"->"6";
"8"->"6";
"9"->"8";
"10"->"9";
"11"->"10";
"12"->"11";
"13"->"11";
"14"->"13";
"15"->"10";
"16"->"8";
};
```

The source of the information shown is V$SQL_PLAN_STATISTICS_ALL which is the same as that used by DBMS_XPLAN. A PL/SQL function is called recursively in order to produce the information that is needed.

Assuming that you have installed Graphviz on your computer, you can use the following command to produce a graphical query plan from the output (spool.dot) of the code. Various output formats are available; the example shown below uses the PDF format.

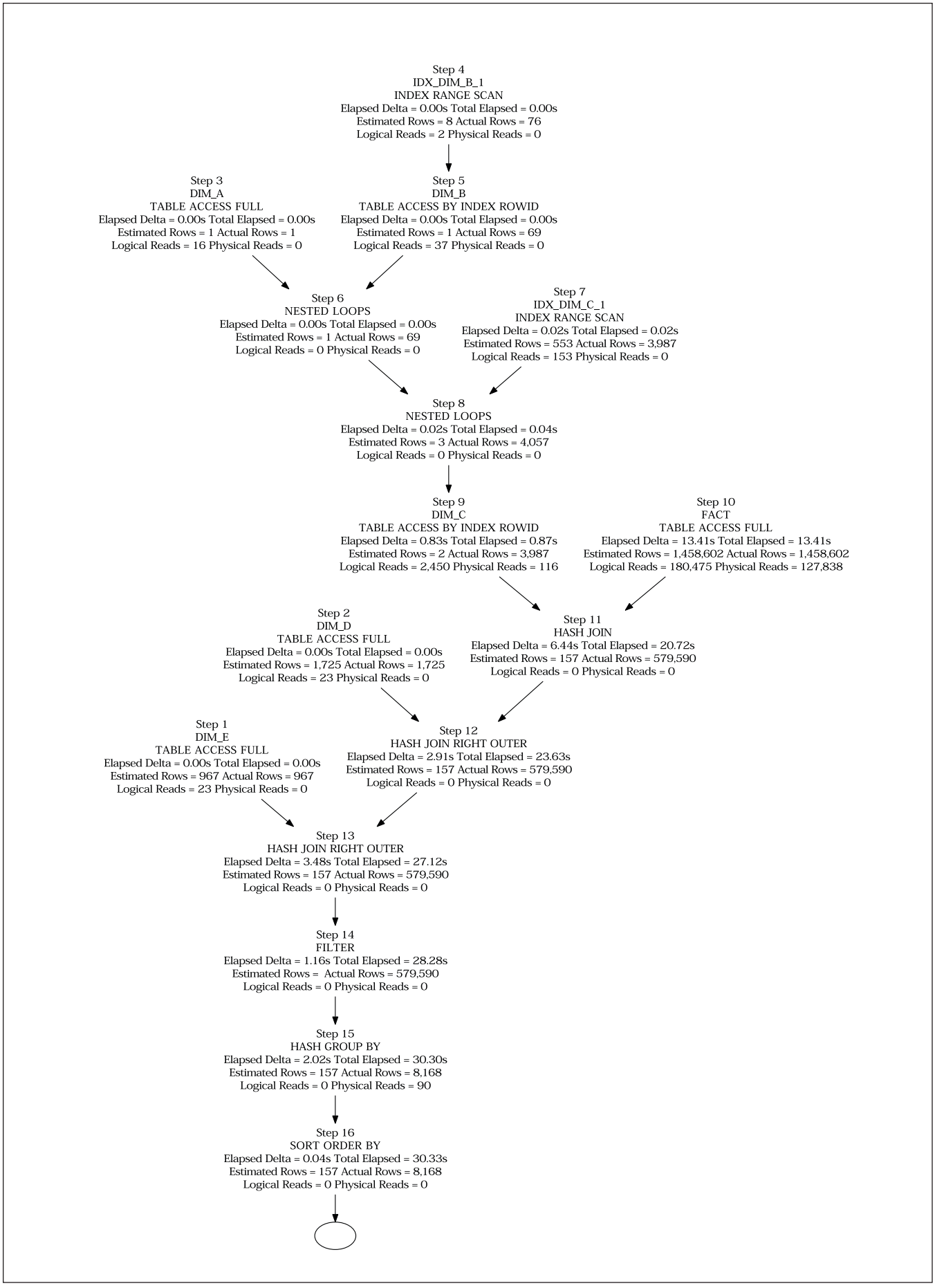
```
dot -Tpdf -oplan.pdf spool.dot
```

If you would like an electronic copy of all the code, please e-mail me at **iggy_fernandez@hotmail.com**. ▲

Iggy Fernandez is the editor of NoCOUG Journal and the author of Beginning Oracle Database 11g Administration (Apress, 2009). He can be reached at **iggy_fernandez@hotmail.com**.

Id	Operation	Name	Starts	E-Rows	E-Bytes	Cost (%CPU)	E-Time	A-Rows	A-Time	Buffers	Reads	Writes
1	SORT ORDER BY		1	157	24963	33884 (5)	00:06:47	8168	00:00:30.33	183K	128K	90
2	HASH GROUP BY		1	157	24963	33884 (5)	00:06:47	8168	00:00:30.30	183K	128K	90
* 3	FILTER		1					579K	00:00:28.28	183K	127K	0
* 4	HASH JOIN RIGHT OUTER		1	157	24963	33883 (5)	00:06:47	579K	00:00:27.12	183K	127K	0
5	TABLE ACCESS FULL	DIM_E	1	967	3868	6 (0)	00:00:01	967	00:00:00.01	23	0	0
* 6	HASH JOIN RIGHT OUTER		1	157	24335	33877 (5)	00:06:47	579K	00:00:23.63	183K	127K	0
7	TABLE ACCESS FULL	DIM_D	1	1725	15525	6 (0)	00:00:01	1725	00:00:00.01	23	0	0
* 8	HASH JOIN		1	157	22922	33870 (5)	00:06:47	579K	00:00:20.72	183K	127K	0
9	TABLE ACCESS BY INDEX ROWID	DIM_C	1	2	126	293 (0)	00:00:04	3987	00:00:00.87	2658	116	0
10	NESTED LOOPS		1	3	375	303 (0)	00:00:04	4057	00:00:00.04	208	0	0
11	NESTED LOOPS		1	1	62	10 (0)	00:00:01	69	00:00:00.01	55	0	0
* 12	TABLE ACCESS FULL	DIM_A	1	1	20	4 (0)	00:00:01	1	00:00:00.01	16	0	0
* 13	TABLE ACCESS BY INDEX ROWID	DIM_B	1	1	42	6 (0)	00:00:01	69	00:00:00.01	39	0	0
* 14	INDEX RANGE SCAN	IDX_DIM_B_1	1	8		1 (0)	00:00:01	76	00:00:00.01	2	0	0
* 15	INDEX RANGE SCAN	IDX_DIM_C_1	69	553		2 (0)	00:00:01	3987	00:00:00.02	153	0	0
* 16	TABLE ACCESS FULL	FACT	1	1458K	29M	33546 (5)	00:06:43	1458K	00:00:13.41	180K	127K	0

Tabular Depiction of a Query Execution Plan



Graphical Depiction of a Query Execution Plan

```
Copyright 2009 Iggy Fernandez

This program is free software: you can redistribute it and/or modify it
under the terms of the GNU General Public License as published by
the Free Software Foundation, either version 3 of the License, or (at
your option) any later version.

This program is distributed in the hope that it will be useful, but
WITHOUT ANY WARRANTY; without even the implied warranty of
MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE.
See the GNU General Public License for more details.

You should have received a copy of the GNU General Public License
along with this program.  If not, see <http://www.gnu.org/licenses/>.

CREATE OR REPLACE TYPE enhanced_plan_type
AS OBJECT
(
  execution_id NUMBER,
  operation VARCHAR2 (120),
  options VARCHAR2 (120),
  object_owner VARCHAR2 (30),
  object_name VARCHAR2 (30),
  id NUMBER,
  parent_id NUMBER,
  cardinality NUMBER,
  last_output_rows NUMBER,
  last_logical_reads NUMBER,
  last_disk_reads NUMBER,
  last_elapsed_time NUMBER,
  delta_elapsed_time NUMBER
);
/

CREATE OR REPLACE TYPE enhanced_plan_table
AS TABLE OF enhanced_plan_type
/

CREATE OR REPLACE PACKAGE enhanced_plan
AS
  FUNCTION plan
  (
    sql_id_in VARCHAR2,
    child_number_in NUMBER,
    parent_id_in NUMBER DEFAULT 0
  )
  RETURN enhanced_plan_table PIPELINED;
END enhanced_plan;
/

CREATE OR REPLACE PACKAGE BODY enhanced_plan AS

  FUNCTION PLAN
  (
    sql_id_in VARCHAR2,
    child_number_in NUMBER,
    parent_id_in NUMBER DEFAULT 0
  )
  RETURN enhanced_plan_table PIPELINED
  IS

    parent_row enhanced_plan_type := enhanced_plan_type
    (
      NULL, NULL, NULL, NULL, NULL, NULL, NULL,
      NULL, NULL, NULL, NULL, NULL, NULL
    );

    child_row enhanced_plan_type := enhanced_plan_type
    (
      NULL, NULL, NULL, NULL, NULL, NULL, NULL,
      NULL, NULL, NULL, NULL, NULL, NULL
    );

    execution_id NUMBER := 1;

    CURSOR parent_cursor IS
    WITH
    parent_statistics AS
    (
      SELECT
        operation,
        options,
        object_owner,
        object_name,
        id,
        parent_id,
        cardinality,
        last_output_rows,
        last_cr_buffer_gets + last_cu_buffer_gets
          AS last_logical_reads,
        last_disk_reads,
        last_elapsed_time / 1000000
          AS last_elapsed_time
      FROM
        v$sql_plan_statistics_all
       WHERE
        sql_id = sql_id_in
        AND child_number = child_number_in
        AND parent_id = parent_id_in
    ),
    child_statistics AS
    (
      SELECT
        parent_id,
        SUM (last_cr_buffer_gets + last_cu_buffer_gets)
          AS last_logical_reads,
        SUM (last_disk_reads) AS last_disk_reads,
        SUM (last_elapsed_time) / 1000000
          AS last_elapsed_time
      FROM
        v$sql_plan_statistics_all
      WHERE sql_id = sql_id_in
      AND child_number = child_number_in
      GROUP BY parent_id
    )
    SELECT
      p.operation,
      p.options,
      p.object_owner,
      p.object_name,
      p.ID,
      p.parent_id,
      p.cardinality,
      p.last_output_rows,
      p.last_logical_reads - NVL (c.last_logical_reads, 0)
        AS last_logical_reads,
      p.last_disk_reads - NVL (c.last_disk_reads, 0)
        AS last_disk_reads,
      p.last_elapsed_time AS last_elapsed_time,
      (p.last_elapsed_time - NVL (c.last_elapsed_time, 0))
        AS delta_elapsed_time
    FROM parent_statistics p, child_statistics c
    WHERE p.ID = c.parent_id(+)
    ORDER BY p.ID;

    CURSOR child_cursor IS
    SELECT
      operation,
      options,
      object_owner,
      object_name,
      ID,
      parent_id,
      cardinality,
      last_output_rows,
      last_logical_reads,
      last_disk_reads,
      last_elapsed_time,
      delta_elapsed_time
    FROM TABLE (enhanced_plan.plan (
      sql_id_in,
      child_number_in,
      parent_row.ID
    ));
```

```
  BEGIN

    OPEN parent_cursor;
    LOOP
      FETCH parent_cursor
      INTO
        parent_row.operation,
        parent_row.options,
        parent_row.object_owner,
        parent_row.object_name,
        parent_row.ID,
        parent_row.parent_id,
        parent_row.cardinality,
        parent_row.last_output_rows,
        parent_row.last_logical_reads,
        parent_row.last_disk_reads,
        parent_row.last_elapsed_time,
        parent_row.delta_elapsed_time;
      EXIT WHEN parent_cursor%NOTFOUND;
      OPEN child_cursor;
      LOOP
        FETCH child_cursor
        INTO
          child_row.operation,
          child_row.options,
          child_row.object_owner,
          child_row.object_name,
          child_row.ID,
          child_row.parent_id,
          child_row.cardinality,
          child_row.last_output_rows,
          child_row.last_logical_reads,
          child_row.last_disk_reads,
          child_row.last_elapsed_time,
          child_row.delta_elapsed_time;
        EXIT WHEN child_cursor%NOTFOUND;
        child_row.execution_id := execution_id;
        execution_id := execution_id + 1;
        PIPE ROW (child_row);
      END LOOP;
      CLOSE child_cursor;
      parent_row.execution_id := execution_id;
      execution_id := execution_id + 1;
      PIPE ROW (parent_row);
    END LOOP;
    CLOSE parent_cursor;

  END plan;

END enhanced_plan;
/

SET linesize 1000
SET trimspool on
SET pagesize 0
SET echo off
SET heading off
SET feedback off
SET verify off
SET time off
SET timing off
SET sqlblanklines on

DEFINE sql_id = &sql_id
DEFINE child_number = &child_number

SPOOL plan.dot

WITH

plan_table AS
(
  SELECT
    *
  FROM
    TABLE (enhanced_plan.plan (
      '&sql_id',
      &child_number
    ))
)

SELECT
  'digraph a {'
FROM
  DUAL

UNION ALL

SELECT
  '"'
  || id
  || '" [label="Step '
  || execution_id
  || '\n'
  || CASE WHEN object_name IS NULL
    THEN ('')
    ELSE (object_name || '\n')
    END
  || CASE WHEN options IS NULL
    THEN (operation || '\n')
    ELSE (operation || ' ' || options || '\n')
    END
  || 'Elapsed Delta = '
  || TRIM (TO_CHAR (delta_elapsed_time, '999,999,990.00'))
  || 's'
  || ' Total Elapsed = '
  || TRIM (TO_CHAR (last_elapsed_time, '999,999,990.00'))
  || 's\n'
  || 'Estimated Rows = '
  || TRIM (TO_CHAR (cardinality, '999,999,999,999,990'))
  || ' Actual Rows = '
  || TRIM (TO_CHAR (last_output_rows, '999,999,999,999,990'))
  || '\n'
  || 'Logical Reads = '
  || TRIM (TO_CHAR (last_logical_reads, '999,999,999,999,990'))
  || ' Physical Reads = '
  || TRIM (TO_CHAR (last_disk_reads, '999,999,999,999,990'))
  || '",shape=plaintext]' op
FROM
  plan_table

UNION ALL

SELECT
  edge
FROM
  (
    SELECT
      parent_id,
      '"' || id || '"' || '->' || '"' || PRIOR id || '"' || ';'
        AS edge
    FROM
      plan_table
      START WITH parent_id = 0
      CONNECT BY parent_id = PRIOR id
  )
  WHERE
    parent_id IS NOT NULL

UNION ALL

SELECT
    '};'
FROM
  DUAL;

SPOOL off
```