Oracle SQL Plan Execution: How It Really Works and How to Troubleshoot It

Tanel Põder

http://blog.tanelpoder.com

http://www.e2sn.com

Introduction

About me:

Occupation:	DBA, researcher, consultant
Expertise:	Oracle internals geek, *End-to-end* performance & scalability, Oracle troubleshooting, Oracle capacity planning
Oracle experience:	13+ years as DBA
Certification:	OCM (2002) OCP (1999)
Professional affiliations:	OakTable Network Oracle ACE Director
Blog:	http://blog.tanelpoder.com
Company:	http://www.e2sn.com

What is an Oracle workload about?

A bunch of sessions

- You need to have a session for doing anything in an Oracle database
- Every session has entries in V$SESSION, V$SESSTAT, etc

Executing SQL (or PL/SQL) cursors

- Every SQL has a SQL_ID (or hash_value which essentially is the same thing)
- Even PL/SQL calls use a cursor for invoking the stored procedure
  - Since Oracle 10.2.0.3 Oracle reports the PLSQL_OBJECT/SUBPROGRAM ID in V$SESSION

Running on CPU - or not running on CPU

- Running on CPU
  - Wanting to run on CPU (but OS doesn't allow it)
- Not running on CPU
  - Sleeping, waiting for system call to complete

What is an execution plan?

For Oracle server:

- *Parsed*, optimized and compiled SQL code kept inside library cache

For DBAs and developers:

- Text or graphical representation of SQL execution flow

Often known as *explain plan*

- To be correct in terms, explain plan is just a tool, command in Oracle
- Explain plan outputs textual representation of execution plan into plan table
- DBAs/developers report human readable output from plan table

Viewing execution plans

DBMS_XPLAN

- *Explain plan for select ....*
  select * from table(dbms_xplan.display())
- select * from table(dbms_xplan.display_cursor(null,null, 'allstats')
- select * from table(dbms_xplan.display_cursor(<*sqlid*>, <*child*>, 'advanced')
- select * from table(dbms_xplan.display_awr(<sqlid>))
- @x.sql

V$SQL_PLAN (and V$SQL_PLAN_STATISTICS[_ALL])

- @xms
- @xmsh <hash_value> <child#>

V$SQL_MONITOR / V$SQL_PLAN_MONITOR (11g+)

- DBMS_SQLTUNE.REPORT_SQL_MONITOR
- @xp <SID> or @xph <SID>

Requires Oracle diagnostics and tuning pack license

event 10132 level 1

- Dumps execution plan to trace file every hard parse

Parse stages

Syntactic check

- Syntax, keywords, sanity

Semantic check

- Whether objects referenced exist, are accessible (by permissions) and are usable

soft parse

View merging

- Queries are written to reference base tables
- Can merge both stored views and inline views

Query transformation

- Transitivity, etc (example: if a=1 and a=b then b=1)

Optimization

Query execution plan (QEP) generation

Loading SQL and execution plan in library cache

hard parse

View merging

Optimizer merges subqueries, inline and stored views and runs queries directly on base tables

- Not always possible though due semantic reasons

```
SQL> create or replace view empview
  2  as
  3  select e.empno, e.ename, d.dname
  4  from emp e, dept d
  5  where e.deptno = d.deptno;

SQL> select * from empview
  2  where ename = 'KING';
```

Can be controlled using:
  Parameter: _complex_view_merging
                  _simple_view_merging

  Hints: MERGE, NO_MERGE

```
------------------------------------------------------------------------------
| Id  | Operation                    | Name       | Rows | Bytes | Cost (%CPU)|
------------------------------------------------------------------------------
|   0 | SELECT STATEMENT             |            |    7 |   210 |     5  (20)|
|*  1 |  HASH JOIN                   |            |    7 |   210 |     5  (20)|
|   2 |   TABLE ACCESS FULL          | DEPT       |    4 |    52 |     2   (0)|
|*  3 |   TABLE ACCESS BY INDEX ROWID| EMP        |    7 |   119 |     2   (0)|
|*  4 |    INDEX RANGE SCAN          | EMP_ENAME  |    8 |       |     1   (0)|
------------------------------------------------------------------------------
```

Subquery unnesting

Subqueries can be unnested, converted to anti- and semijoins

Can be controlled using:
Parameter: _unnest_subqueries
Hints: UNNEST, NO_UNNEST

```
SQL> select * from employees e
  2  where exists (
  3      select ename from bonus b
  4      where e.ename = b.ename
  5  );
--------------------------------------------------------------------------
| Id  | Operation                     | Name      | Rows  | Bytes | Cost (
--------------------------------------------------------------------------
|   0 | SELECT STATEMENT              |           |     1 |    37 |     5
|   1 |  NESTED LOOPS                 |           |     1 |    37 |     5
|   2 |   NESTED LOOPS                |           |     1 |    24 |     4
|   3 |    SORT UNIQUE                |           |     1 |     7 |     2
|   4 |     TABLE ACCESS FULL         | BONUS     |     1 |     7 |     2
|*  5 |    TABLE ACCESS BY INDEX ROWID| EMP       |     1 |    17 |     1
|*  6 |     INDEX RANGE SCAN          | EMP_ENAME |    37 |       |     1
|   7 |   TABLE ACCESS BY INDEX ROWID | DEPT      |     1 |    13 |     1
|*  8 |    INDEX UNIQUE SCAN          | PK_DEPT   |     1 |       |     0
--------------------------------------------------------------------------
Predicate Information (identified by operation id):
---------------------------------------------------
   5 - filter("E"."DEPTNO" IS NOT NULL)
   6 - access("E"."ENAME"="B"."ENAME")
   8 - access("E"."DEPTNO"="D"."DEPTNO")
```

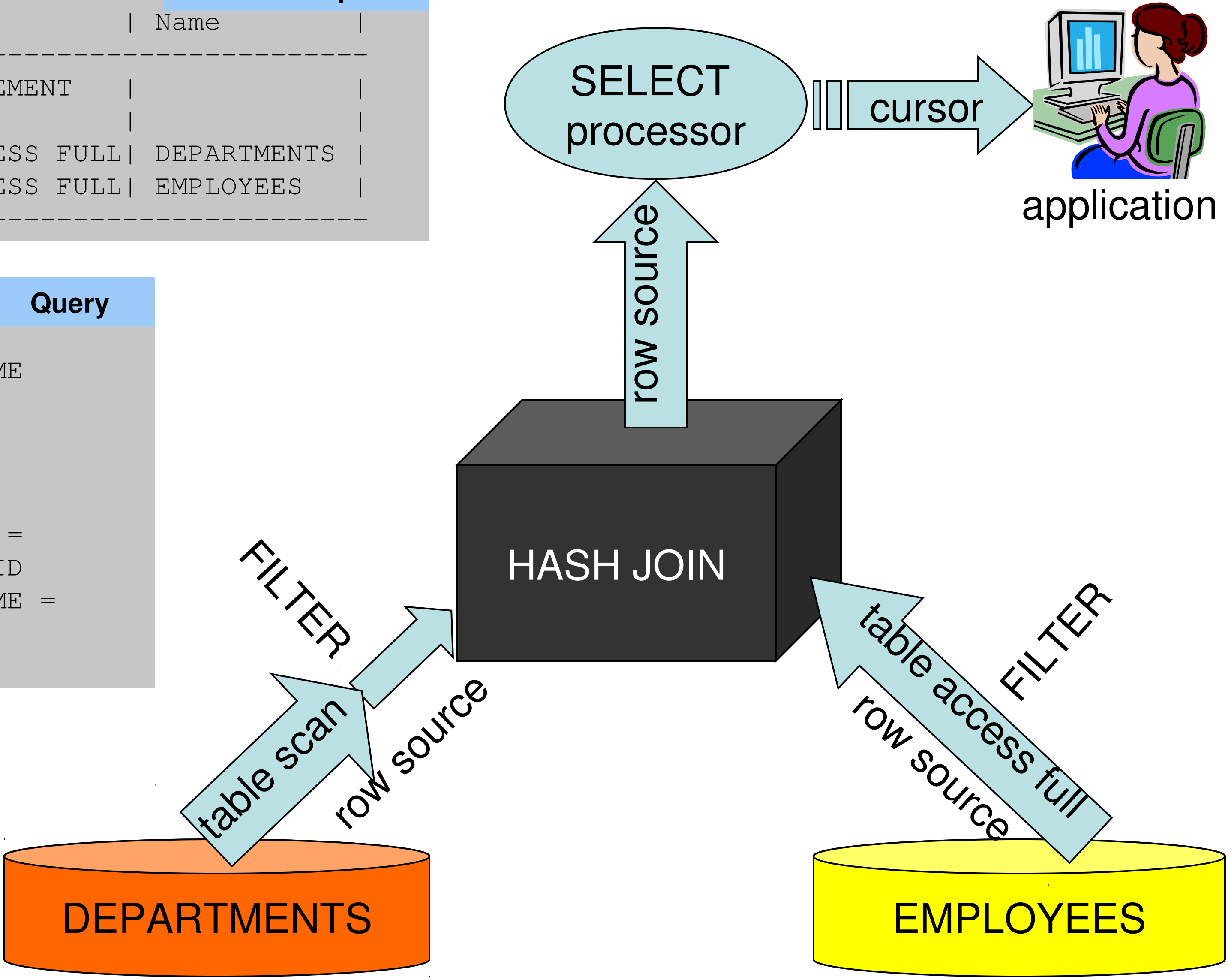
SQL execution basics

Execution plan

```
-------------------------------------------
| Id  | Operation          | Name        |
-------------------------------------------
|   0 | SELECT STATEMENT   |             |
|*  1 |  HASH JOIN         |             |
|*  2 |   TABLE ACCESS FULL| DEPARTMENTS |
|*  3 |   TABLE ACCESS FULL| EMPLOYEES   |
-------------------------------------------
```

Query

```
SELECT
    E.LAST_NAME,
    D.DEPARTMENT_NAME
FROM
    EMPLOYEES E,
    DEPARTMENTS D
WHERE
    E.DEPARTMENT_ID =
      D.DEPARTMENT_ID
AND D.DEPARTMENT_NAME =
      'Sales'
AND E.SALARY > 2000;
```



SQL execution basics - multitable joins

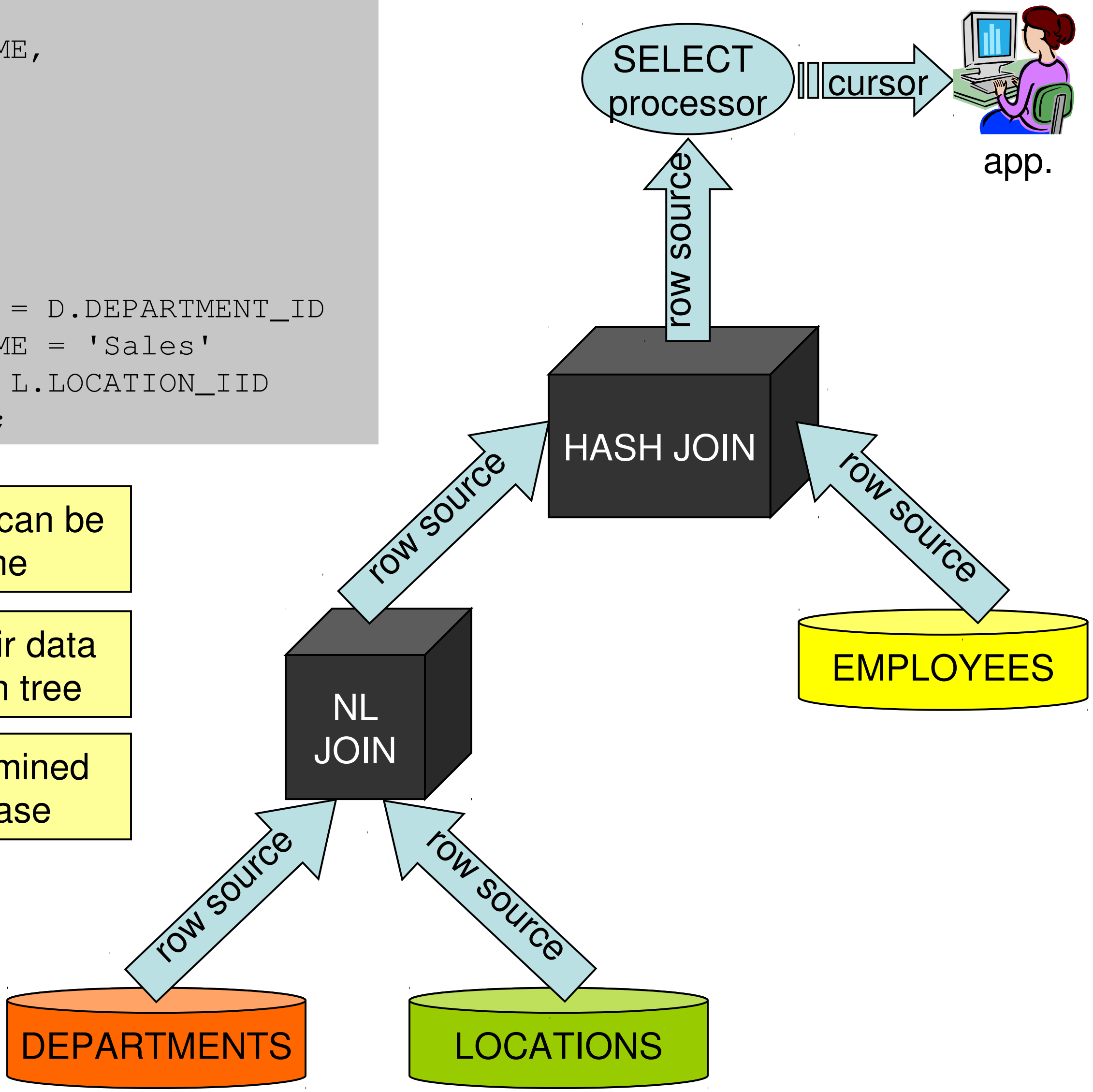
Multiple joins

```
SELECT
    E.LAST_NAME,
    D.DEPARTMENT_NAME,
    L.CITY
FROM
    EMPLOYEES E,
    DEPARTMENTS D,
    LOCATIONS L
WHERE
    E.DEPARTMENT_ID = D.DEPARTMENT_ID
AND D.DEPARTMENT_NAME = 'Sales'
AND D.LOCATION_ID = L.LOCATION_IID
AND E.SALARY > 2000;
```

Only two row sources can be joined together at a time

Row sources pass their data "up" the execution plan tree

The join order is determined during optimization phase



SQL execution terminology

ACCESS PATH

- A means to access physical data in database storage
- From tables, indexes, external tables, database links

ROW SOURCE

- A virtual stream of rows
- Can come through access paths from tables, indexes
- Or from other child row sources

FILTER *PREDICATE*

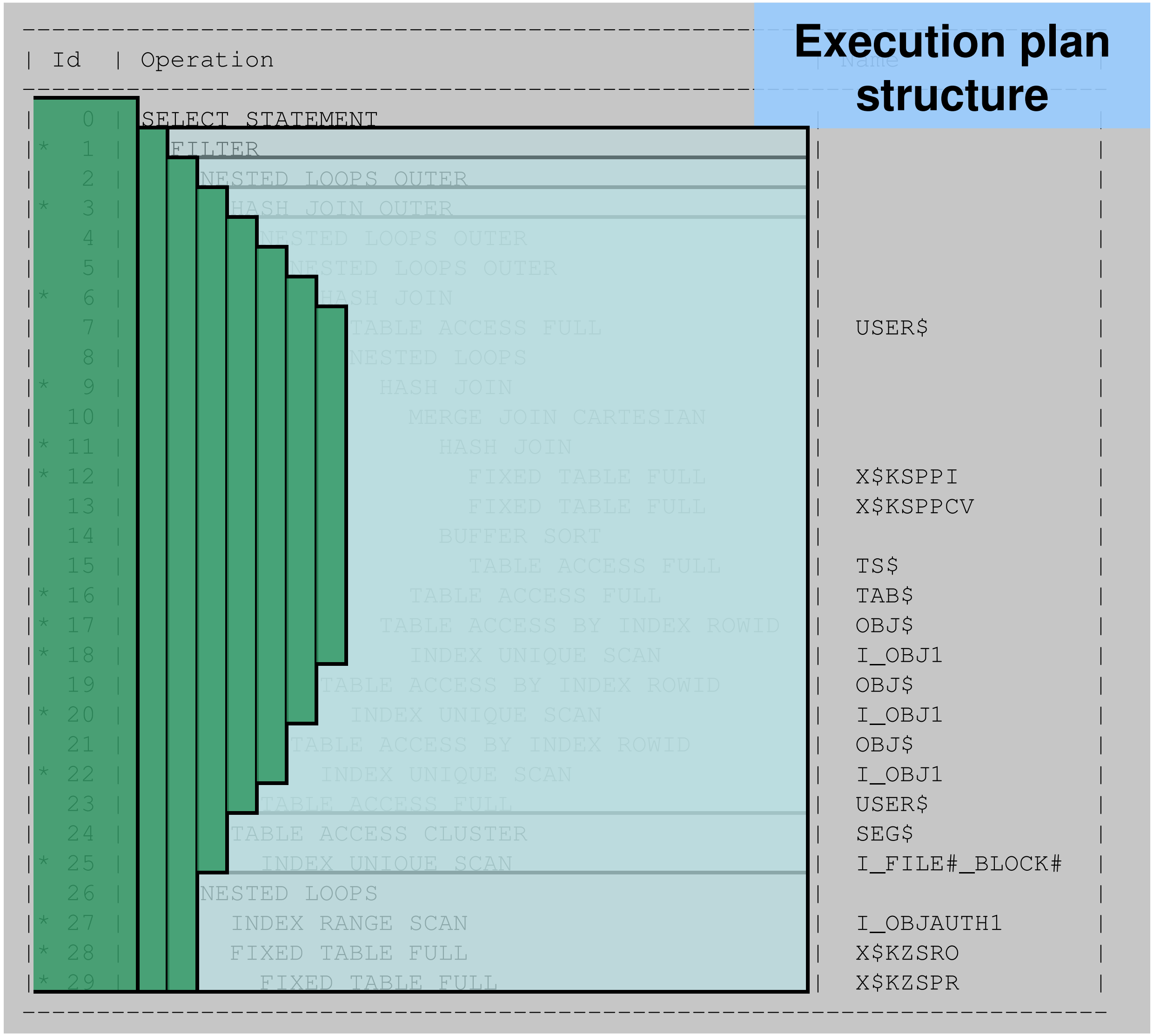
- A property of row source - can discard rows based on defined conditions - *filter predicates*

JOIN

- Filters and merges rows based on matching rows from child rowsources. Matching is defined by *join predicates*
- Any join operator can join only two inputs

First rule for reading an execution plan

Parent operations get input only from their children

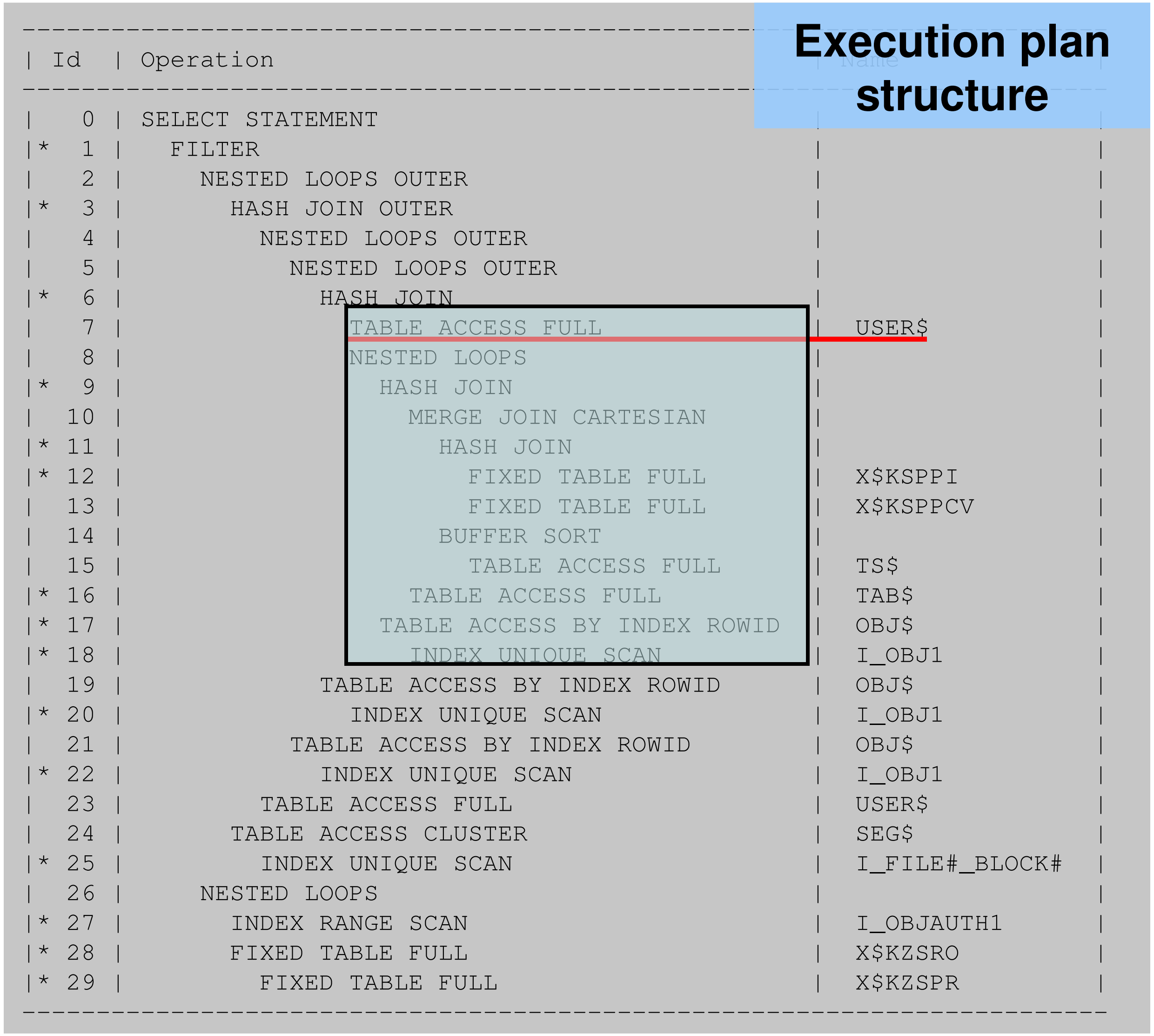


Execution plan structure

```
--------------------------------------------------------------------------
| Id  | Operation                                    |  Name             |
--------------------------------------------------------------------------
|   0 | SELECT STATEMENT                             |                   |
|*  1 |  FILTER                                      |                   |
|   2 |   NESTED LOOPS OUTER                         |                   |
|*  3 |    HASH JOIN OUTER                           |                   |
|   4 |     NESTED LOOPS OUTER                       |                   |
|   5 |      NESTED LOOPS OUTER                      |                   |
|*  6 |       HASH JOIN                              |                   |
|   7 |        TABLE ACCESS FULL                     |  USER$            |
|   8 |        NESTED LOOPS                          |                   |
|*  9 |         HASH JOIN                            |                   |
|  10 |          MERGE JOIN CARTESIAN                |                   |
|* 11 |           HASH JOIN                          |                   |
|* 12 |            FIXED TABLE FULL                  |  X$KSPPI          |
|  13 |            FIXED TABLE FULL                  |  X$KSPPCV         |
|  14 |           BUFFER SORT                        |                   |
|  15 |            TABLE ACCESS FULL                 |  TS$              |
|* 16 |          TABLE ACCESS FULL                   |  TAB$             |
|* 17 |         TABLE ACCESS BY INDEX ROWID          |  OBJ$             |
|* 18 |          INDEX UNIQUE SCAN                   |  I_OBJ1           |
|  19 |       TABLE ACCESS BY INDEX ROWID            |  OBJ$             |
|* 20 |        INDEX UNIQUE SCAN                     |  I_OBJ1           |
|  21 |      TABLE ACCESS BY INDEX ROWID             |  OBJ$             |
|* 22 |       INDEX UNIQUE SCAN                      |  I_OBJ1           |
|  23 |     TABLE ACCESS FULL                        |  USER$            |
|  24 |    TABLE ACCESS CLUSTER                      |  SEG$             |
|* 25 |     INDEX UNIQUE SCAN                        |  I_FILE#_BLOCK#   |
|  26 |   NESTED LOOPS                               |                   |
|* 27 |    INDEX RANGE SCAN                          |  I_OBJAUTH1       |
|* 28 |    FIXED TABLE FULL                          |  X$KZSRO          |
|* 29 |     FIXED TABLE FULL                         |  X$KZSPR          |
--------------------------------------------------------------------------
```

Second rule for reading an execution plan

Data access starts from the first line without children



```
--------------------------------------------------------------------------
| Id  | Operation                                     | Name             |
--------------------------------------------------------------------------
|   0 | SELECT STATEMENT                              |                  |
|*  1 |   FILTER                                      |                  |
|   2 |     NESTED LOOPS OUTER                        |                  |
|*  3 |       HASH JOIN OUTER                         |                  |
|   4 |         NESTED LOOPS OUTER                    |                  |
|   5 |           NESTED LOOPS OUTER                  |                  |
|*  6 |             HASH JOIN                         |                  |
|   7 |               TABLE ACCESS FULL               | USER$            |
|   8 |               NESTED LOOPS                    |                  |
|*  9 |                 HASH JOIN                     |                  |
|  10 |                   MERGE JOIN CARTESIAN        |                  |
|* 11 |                     HASH JOIN                 |                  |
|* 12 |                       FIXED TABLE FULL        | X$KSPPI          |
|  13 |                       FIXED TABLE FULL        | X$KSPPCV         |
|  14 |                     BUFFER SORT               |                  |
|  15 |                       TABLE ACCESS FULL       | TS$              |
|* 16 |                   TABLE ACCESS FULL           | TAB$             |
|* 17 |                 TABLE ACCESS BY INDEX ROWID   | OBJ$             |
|* 18 |                   INDEX UNIQUE SCAN           | I_OBJ1           |
|  19 |             TABLE ACCESS BY INDEX ROWID       | OBJ$             |
|* 20 |               INDEX UNIQUE SCAN               | I_OBJ1           |
|  21 |           TABLE ACCESS BY INDEX ROWID         | OBJ$             |
|* 22 |             INDEX UNIQUE SCAN                 | I_OBJ1           |
|  23 |         TABLE ACCESS FULL                     | USER$            |
|  24 |       TABLE ACCESS CLUSTER                    | SEG$             |
|* 25 |         INDEX UNIQUE SCAN                     | I_FILE#_BLOCK#   |
|  26 |     NESTED LOOPS                              |                  |
|* 27 |       INDEX RANGE SCAN                        | I_OBJAUTH1       |
|* 28 |       FIXED TABLE FULL                        | X$KZSRO          |
|* 29 |         FIXED TABLE FULL                      | X$KZSPR          |
--------------------------------------------------------------------------
```

Execution plan structure

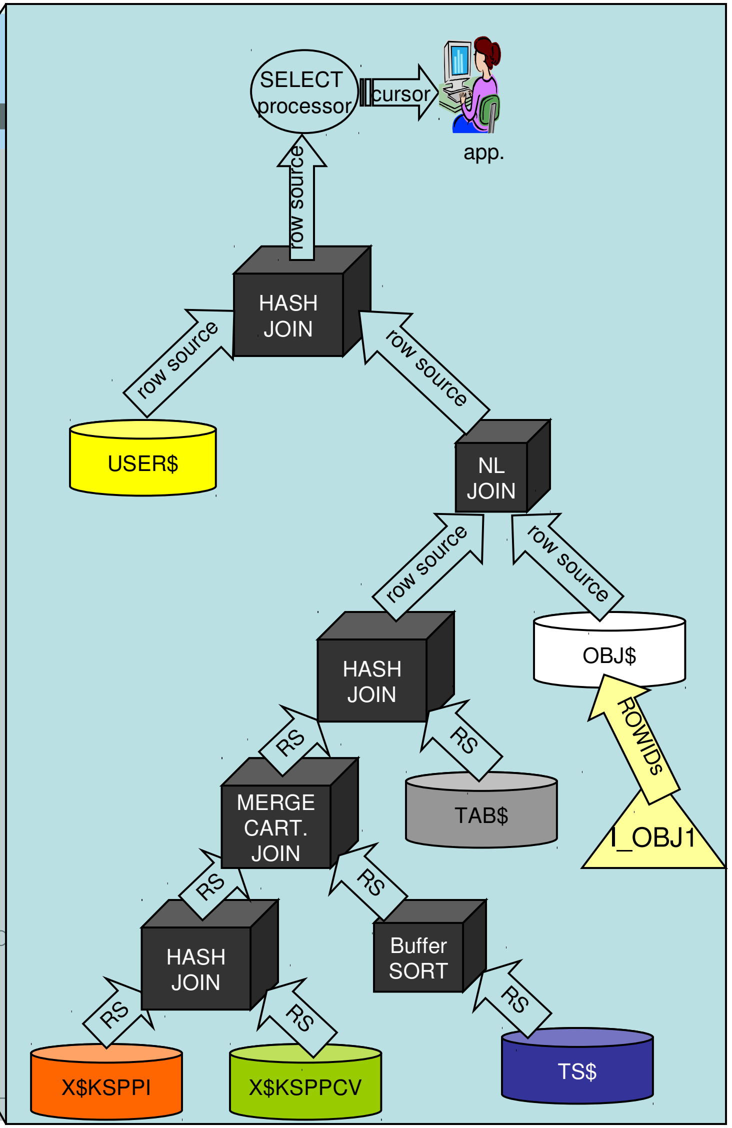
First operation with no children (leaf operation) **accesses** data

Cascading rowsources

Rows "flow" upwards to parent rowsources in cascading fashion

```
-------------------------------------------------
| Id  | Operation
-------------------------------------------------
|   0 | SELECT STATEMENT
|*  1 |   FILTER                                     |
|   2 |     NESTED LOOPS OUTER                       |
|*  3 |       HASH JOIN OUTER                        |
|   4 |         NESTED LOOPS OUTER                   |
|   5 |           NESTED LOOPS OUTER                 |
|*  6 |             HASH JOIN                        |
|   7 |               TABLE ACCESS FULL              | USER$
|   8 |               NESTED LOOPS                   |
|*  9 |                 HASH JOIN                    |
|  10 |                   MERGE JOIN CARTESIAN       |
|* 11 |                     HASH JOIN                |
|* 12 |                       FIXED TABLE FULL       | X$KSPPI
|  13 |                       FIXED TABLE FULL       | X$KSPPCV
|  14 |                     BUFFER SORT              |
|  15 |                       TABLE ACCESS FULL      | TS$
|* 16 |                   TABLE ACCESS FULL          | TAB$
|* 17 |                 TABLE ACCESS BY INDEX ROWID  | OBJ$
|* 18 |                   INDEX UNIQUE SCAN          | I_OBJ1
|  19 |             TABLE ACCESS BY INDEX ROWID      | OBJ$
|* 20 |               INDEX UNIQUE SCAN              | I_OBJ1
|  21 |           TABLE ACCESS BY INDEX ROWID        | OBJ$
|* 22 |             INDEX UNIQUE SCAN                | I_OBJ1
|  23 |         TABLE ACCESS FULL                    | USER$
|  24 |       TABLE ACCESS CLUSTER                   | SEG$
|* 25 |         INDEX UNIQUE SCAN                    | I_FILE#_BLO
|  26 |     NESTED LOOPS                             |
|* 27 |       INDEX RANGE SCAN                       | I_OBJAUTH1
|* 28 |       FIXED TABLE FULL                       | X$KZSRO
|* 29 |         FIXED TABLE FULL                     | X$KZSPR
-------------------------------------------------------------------
```

Execution structur



SQL execution plan recap

Execution plan lines are just Oracle kernel functions!

- In other words, each row source is a function

Data can only be accessed using *access path functions*

- Only access paths can access physical data
- Access paths process physical data, return *row sources*

Data processing starts from first line without children

- In other words the first leaf access path in execution plan

Row sources feed data to their parents

- Can be non-cascading, semi-cascading or cascading

A join operation can input only two row sources

- However, it is possible to combine result of more than 2 row sources for some operations (not for joins though)
- Index combine, bitmap merging, filter, union all, for example

SQL Plan profiling

SQL execution plan line level profiling available since Oracle 9.2

- Stats externalized in V$SQL_PLAN_STATISTICS[_ALL]

Statistics gathering is enabled by setting parameter:

- statistics_level=all
- ...or _rowsource_execution_statistics=true
- or via hint: /*+ gather_plan_statistics */ (Oracle 10.2+)

- Don't enable this at instance level as it can kill your performance

```
$ pstack 1780 | ./os_explain
   kpoal8
    SELECT FETCH:
     QUERY EXECUTION STATISTICS: Fetch
       GROUP BY SORT: Fetch
       QUERY EXECUTION STATISTICS: Fetch
         NESTED LOOP JOIN: Fetch
         QUERY EXECUTION STATISTICS: Fetch
           SORT: Fetch
            sorgetqbf
```

- Parameter introduced for reducing profiling overhead via reducing gettimeofday() syscalls
    - _rowsource_statistics_sampfreq = 128

Reading DBMS_XPLAN execution plan profile

```
SQL> select * from table(dbms_xplan.display_cursor(null,null,'ALLSTATS LAST'));
PLAN_TABLE_OUTPUT
-------------------------------------------------------------------------------------------
SQL_ID  56bs32ukywdsq, child number 0
--------------------------------------
select count(*) from dba_tables
Plan hash value: 736297560
-------------------------------------------------------------------------------------------
| Id  | Operation                    | Name       | Starts | E-Rows | A-Rows |   A-Time   |
-------------------------------------------------------------------------------------------
|   1 |  SORT AGGREGATE              |            |      1 |      1 |      1 |00:00:00.38 |
|*  2 |   HASH JOIN RIGHT OUTER      |            |      1 |   1690 |   1688 |00:00:00.37 |
|   3 |    TABLE ACCESS FULL         | USER$      |      1 |     68 |     68 |00:00:00.01 |
|*  4 |    HASH JOIN OUTER           |            |      1 |   1690 |   1688 |00:00:00.37 |
|*  5 |     HASH JOIN
|   6 |      TABLE ACCESS FU
|*  7 |      HASH JOIN
|   8 |       NESTED LOOPS O
|*  9 |        HASH JOIN RIG
|  10 |         TABLE ACCESS
|* 11 |         HASH JOIN
|  12 |          MERGE JOIN
|* 13 |           HASH JOIN
|* 14 |            FIXED TAB
|  15 |            FIXED TAB
|  16 |           BUFFER SOR
|  17 |            TABLE ACC
|* 18 |          TABLE ACCES
|* 19 |        INDEX UNIQUE
|* 20 |       TABLE ACCESS F
|  21 |     TABLE ACCESS FULL        | OBJ$       |      1 |  55517 |  55514 |00:00:00.01 |
-------------------------------------------------------------------------------------------
```

Starts — number of times the rowsource was initialized

E-rows — CBO number estimated rows coming from rowsource

A-rows — actual *measured* number of rows during last execution

A-time — actual *measured (and extrapolated)* time spent inside a rowsource function or under its children (cumulative)

Buffer — number of buffer gets done within rowsource during last execution

Reading XMS/XMSH execution plan profile

```
SQL> @xms

SQL hash value:          2783852310    Cursor address:                00000003DCA9EF28    |    Statement firs

 Ch Pr   Op                                             Object          ms spent    Estimated Real #rows    Op. ite-
 ld ed   ID Operation                                   Name             in op.   output rows   returned     rations
--- -- ---- ---------------------------------------- ---------------- ------------ ----------- ---------- ----------
  0         0 SELECT STATEMENT
            1  SORT AGGRE
    A       2   HASH JOIN
            3    TABLE AC
    A       4    HASH JOI
    A       5     HASH JO
            6      TABLE
    A       7      HASH J
            8       NESTE
    A       9        HASH
           10         TAE
    A      11         HAS
           12          ME
    A      13           H
     F     14
           15
           16           E
           17
     F     18          TA
    A      19        INDE
     F     20       TABLE
           21     TABLE A

 Ch Op
 ld ID      Predicate In
--- ------ ------------
  0      2 - access("CX
         4 - access("T"
         5 - access("O"
         7 - access("O"
         9 - access("T"
```

ms spent in op.	milliseconds spent in rowsource function (cumulative)
Estimated rows	CBO rowcount estimate
Real # rows	Real *measured* rowcount from rowsource
Op. iterations	Number of times the rowsource was initialized
Logical reads	Consistent buffer gets
Logical writes	Current mode buffer gets (Note that some CUR gets may not always be due writing...)
Physical reads	Physial reads done by the rowsource function
Physical writes	Physical writes done by the rowsource function
Optimizer cost	Least significant thing for measuring the *real execution efficiency* of a statement

Real time SQL execution monitoring

Oracle 11g new feature

- Uses V$SQL_MONITOR and V$SQL_PLAN_MONITOR
- Always enabled for parallel execution queries
- Kicks in for serial queries after they've waited total 5 seconds for IO or have used CPU
  - *_sqlmon_threshold = 5*

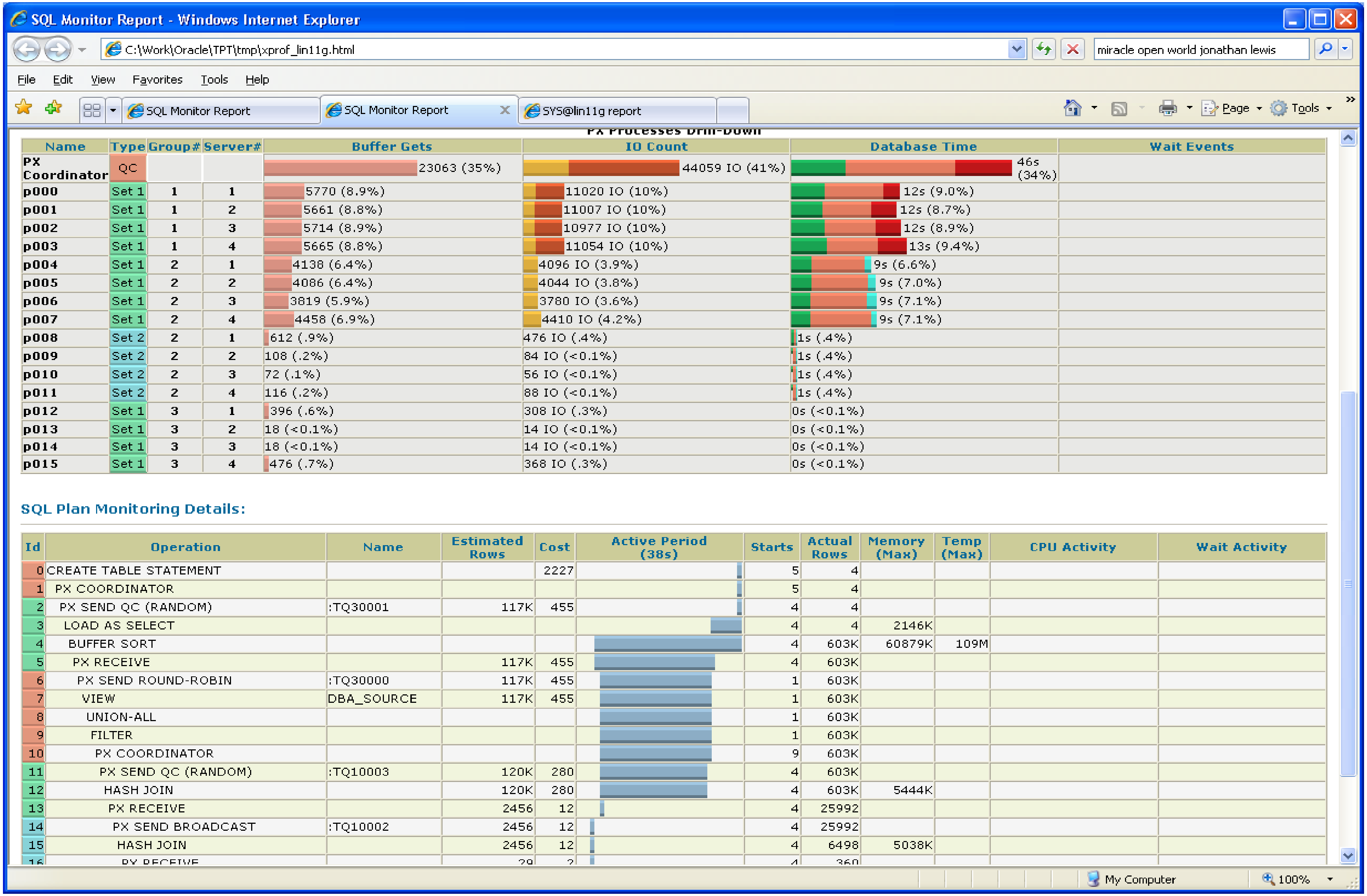
- You can also use MONITOR and NO_MONITOR hints for controlling the monitoring

```
Get execution statistics of last query executed in session:

SELECT
   DBMS_SQLTUNE.      REPORT_SQL_MONITOR        (
          session_id=>sys_context('userenv','sid'),
          report_level=>'ALL',
          type = 'TEXT'             — or HTML
```

Execution Profile (dbms_sqltune.report_sql_monitor)

```
SQL> @xp 128

REPORT
----------------------------------------------------------------------------------------------------------------
SQL Monitoring Report

SQL Text
----------------------------------------------------------------------------------------------------------------
select /*+ ordered use_nl(b) full(a) full(b) */ count(*) from sys.obj$ a, sys.obj$ b where a.name = b.name and r
----------------------------------------------------------------------------------------------------------------

Global Information
 Status              :  EXECUTING
 Instance ID         :  1
 Session ID          :  128
 SQL ID              :  1vm188y2gv75n
 SQL Execution ID    :  16777217
 Plan Hash Value     :  2119813036
 Execution Started   :  08/14/2008 18:12:52
 First Refresh Time  :  08/14/2008 18:13:00
 Last Refresh Time   :  08/14/2008 18:13:20


SQL Plan Monitoring Details
================================================================================================================
| Id   |       Operation        | Name |  Rows   |  Cost  |   Time    | Start  | Starts |   Rows   | Activity  |
|      |                        |      | (Estim) |        | Active(s) | Active |        | (Actual) | (percent) |
================================================================================================================
|    0 | SELECT STATEMENT       |      |         | 16502K |           |        |      1 |          |           |
|    1 |   SORT AGGREGATE       |      |       1 |        |           |        |      1 |          |           |
| -> 2 |    COUNT STOPKEY       |      |         |        |        21 |     +8 |      1 |     3006 |           |
| -> 3 |     NESTED LOOPS       |      |    116K | 16502K |        21 |     +8 |      1 |     3006 |           |
| -> 4 |      TABLE ACCESS FULL | OBJ$ |   69996 |    238 |        21 |     +8 |      1 |     2925 |           |
| -> 5 |      TABLE ACCESS FULL | OBJ$ |       2 |    236 |        28 |     +1 |   2926 |     3006 |    100.00 |
================================================================================================================
```

Execution Profile HTML output (type=>'HTML')


SQL Monitor Report - Windows Internet Explorer
C:\Work\Oracle\TPT\tmp\xprof_lin11g.html
miracle open world jonathan lewis
File
Edit
View
Favorites
Tools
Help
SQL Monitor Report
SQL Monitor Report
SYS@lin11g report
Page
Tools
My Computer
100%

PX Processes Drill-Down

Name	Type	Group#	Server#	Buffer Gets	IO Count	Database Time	Wait Events
PX Coordinator	QC			23063 (35%)	44059 IO (41%)	46s (34%)	
p000	Set 1	1	1	5770 (8.9%)	11020 IO (10%)	12s (9.0%)	
p001	Set 1	1	2	5661 (8.8%)	11007 IO (10%)	12s (8.7%)	
p002	Set 1	1	3	5714 (8.9%)	10977 IO (10%)	12s (8.9%)	
p003	Set 1	1	4	5665 (8.8%)	11054 IO (10%)	13s (9.4%)	
p004	Set 1	2	1	4138 (6.4%)	4096 IO (3.9%)	9s (6.6%)	
p005	Set 1	2	2	4086 (6.4%)	4044 IO (3.8%)	9s (7.0%)	
p006	Set 1	2	3	3819 (5.9%)	3780 IO (3.6%)	9s (7.1%)	
p007	Set 1	2	4	4458 (6.9%)	4410 IO (4.2%)	9s (7.1%)	
p008	Set 2	2	1	612 (.9%)	476 IO (.4%)	1s (.4%)	
p009	Set 2	2	2	108 (.2%)	84 IO (<0.1%)	1s (.4%)	
p010	Set 2	2	3	72 (.1%)	56 IO (<0.1%)	1s (.4%)	
p011	Set 2	2	4	116 (.2%)	88 IO (<0.1%)	1s (.4%)	
p012	Set 1	3	1	396 (.6%)	308 IO (.3%)	0s (<0.1%)	
p013	Set 1	3	2	18 (<0.1%)	14 IO (<0.1%)	0s (<0.1%)	
p014	Set 1	3	3	18 (<0.1%)	14 IO (<0.1%)	0s (<0.1%)	
p015	Set 1	3	4	476 (.7%)	368 IO (.3%)	0s (<0.1%)	

SQL Plan Monitoring Details:

Id	Operation	Name	Estimated Rows	Cost	Active Period (38s)	Starts	Actual Rows	Memory (Max)	Temp (Max)	CPU Activity	Wait Activity
0	CREATE TABLE STATEMENT			2227		5	4				
1	PX COORDINATOR					5	4				
2	PX SEND QC (RANDOM)	:TQ30001	117K	455		4	4				
3	LOAD AS SELECT					4	4	2146K			
4	BUFFER SORT					4	603K	60879K	109M		
5	PX RECEIVE		117K	455		4	603K				
6	PX SEND ROUND-ROBIN	:TQ30000	117K	455		1	603K				
7	VIEW	DBA_SOURCE	117K	455		1	603K				
8	UNION-ALL					1	603K				
9	FILTER					1	603K				
10	PX COORDINATOR					9	603K				
11	PX SEND QC (RANDOM)	:TQ10003	120K	280		4	603K				
12	HASH JOIN		120K	280		4	603K	5444K			
13	PX RECEIVE		2456	12		4	25992				
14	PX SEND BROADCAST	:TQ10002	2456	12		4	25992				
15	HASH JOIN		2456	12		4	6498	5038K			
16	PX RECEIVE		29	2		4	360				

Simple full table scan

Full table scan scans all the rows in the table

- All table blocks are scanned up to the HWM
- Even if all rows have been deleted from table
- Oracle uses multiblock reads where it can
- Most efficient way when querying majority of rows
  - And majority of columns

```
SQL> select * from emp;

PLAN_TABLE_OUTPUT
-----------------------------------------------------------------------------
Plan hash value: 4080710170

---------------------------------------------------------------------------
| Id  | Operation         | Name | Rows  | Bytes | Cost (%CPU)| Time     |
---------------------------------------------------------------------------
|   0 | SELECT STATEMENT  |      |    14 |   518 |     3   (0)| 00:00:01 |
|   1 |  TABLE ACCESS FULL| EMP  |    14 |   518 |     3   (0)| 00:00:01 |
---------------------------------------------------------------------------
```

Full table scan with a filter predicate

Filter operation throws away non-matching rows

- By definition, not the most efficient operation
- Filter conditions can be seen in predicate section

```
SQL> select * from emp where ename = 'KING';
PLAN_TABLE_OUTPUT
--------------------------------------------------------------------------
Plan hash value: 4080710170

--------------------------------------------------------------------------
| Id  | Operation         | Name | Rows  | Bytes | Cost (%CPU)| Time     |
--------------------------------------------------------------------------
|   0 | SELECT STATEMENT  |      |     1 |    37 |     3   (0)| 00:00:01 |
|*  1 |  TABLE ACCESS FULL| EMP  |     1 |    37 |     3   (0)| 00:00:01 |
--------------------------------------------------------------------------

Predicate Information (identified by operation id):
---------------------------------------------------
   1 - filter("ENAME"='KING')
```

Simple B*tree index+table access

Index tree is walked from root to leaf

- Key values and ROWIDs are gotten from index
- Table rows are gotten using ROWIDs
- *Access* operator fetches only matching rows
  - As opposed to *filter* which filters through the whole child rowsource

```
SQL> select * from emp where empno = 10;

---------------------------------------------------------------------------
| Id  | Operation                   | Name   | Rows  | Bytes | Cost (%CPU)|
---------------------------------------------------------------------------
|   0 | SELECT STATEMENT            |        |     1 |    37 |     1   (0)|
|   1 |  TABLE ACCESS BY INDEX ROWID| EMP    |     1 |    37 |     1   (0)|
|*  2 |   INDEX UNIQUE SCAN         | PK_EMP |     1 |       |     0   (0)|
---------------------------------------------------------------------------

Predicate Information (identified by operation id):
---------------------------------------------------

   2 - access("EMPNO"=10)
```

Predicate attributes

Predicate = access

- A means to avoid processing (some) unneeded data at all

Predicate = filter

- Everything from child row source is processed / filtered
- The non-matching rows are *thrown away*

```
SQL> select * from emp
  2  where empno > 7000
  3  and ename like 'KING%';
-------------------------------------------------------------------
| Id  | Operation                   | Name   | Rows  | Bytes | Cost (%CPU)|
-------------------------------------------------------------------
|   0 | SELECT STATEMENT            |        |     1 |    27 |     3   (0)|
|*  1 |  TABLE ACCESS BY INDEX ROWID| EMP    |     1 |    27 |     3   (0)|
|*  2 |   INDEX RANGE SCAN          | PK_EMP |     9 |       |     2   (0)|
-------------------------------------------------------------------
Predicate Information (identified by operation id):

PLAN_TABLE_OUTPUT
-------------------------------------------------------------------
   1 - filter("ENAME" LIKE 'KING%')
   2 - access("EMPNO">7000)
```

Index fast full scan

Doesn't necessarily return keys in order

- The whole index segment is just scanned as Oracle finds its blocks on disk (in contrast to tree walking)
- Multiblock reads are used
- As indexes don't usually contain all columns that tables do, FFS is more efficient if all used columns are in index
- Used mainly for aggregate functions, min/avg/sum,etc
- Optimizer must know that all table rows are represented in index! (null values and count example)

```
SQL> select min(empno), max(empno) from emp;

---------------------------------------------------------------------
| Id  | Operation             | Name   | Rows  | Bytes | Cost (%CPU)|
---------------------------------------------------------------------
|   0 | SELECT STATEMENT      |        |     1 |     5 |    25   (0)|
|   1 |  SORT AGGREGATE       |        |     1 |     5 |            |
|   2 |   INDEX FAST FULL SCAN| PK_EMP | 54121 |   264K|    25   (0)|
---------------------------------------------------------------------
```

Nested Loop Join

Nested loop join

- Read data from outer row source (upper one)
- *Probe* for a match in inner row source for each outer row

```
SQL> select d.dname, d.loc, e.empno, e.ename
  2  from emp e, dept d
  3  where e.deptno = d.deptno
  4  and d.dname = 'SALES'
  5  and e.ename like 'K%';

------------------------------------------------------------------------
| Id  | Operation                    | Name    | Rows  | Bytes | Cost  |
------------------------------------------------------------------------
|   0 | SELECT STATEMENT             |         |     1 |    37 |     4 |
|   1 |  NESTED LOOPS                |         |     1 |    37 |     4 |
|*  2 |   TABLE ACCESS FULL          | EMP     |     1 |    17 |     3 |
|*  3 |   TABLE ACCESS BY INDEX ROWID| DEPT    |     1 |    20 |     1 |
|*  4 |    INDEX UNIQUE SCAN         | PK_DEPT |     1 |       |       |
------------------------------------------------------------------------
Predicate Information (identified by operation id):
---------------------------------------------------
   2 - filter("E"."DEPTNO" IS NOT NULL AND "E"."ENAME" LIKE 'K%')
   3 - filter("D"."DNAME"='SALES')
   4 - access("E"."DEPTNO"="D"."DEPTNO")
```

Hash Join

Only for equijoins/non-equijoins (outer joins in 10g)

- Builds an array with hashed key values from smaller row source
- Scans the bigger row source, builds and compares hashed key values on the fly, returns matching ones

```
SQL> select d.dname, d.loc, e.empno, e.ename
  2  from emp e, dept d
  3  where e.deptno = d.deptno
  4  and d.dname = 'SALES'
  5  and e.ename between 'A%' and 'M%';
----------------------------------------------------------------
| Id  | Operation          | Name | Rows  | Bytes | Cost (%CPU)|
----------------------------------------------------------------
|   0 | SELECT STATEMENT   |      |     1 |    37 |     9  (12)|
|*  1 |  HASH JOIN         |      |     1 |    37 |     9  (12)|
|*  2 |   TABLE ACCESS FULL| DEPT |     1 |    20 |     2   (0)|
|*  3 |   TABLE ACCESS FULL| EMP  |     4 |    68 |     6   (0)|
----------------------------------------------------------------
Predicate Information (identified by operation id):
---------------------------------------------------
   1 - access("E"."DEPTNO"="D"."DEPTNO")
   2 - filter("D"."DNAME"='SALES')
   3 - filter("E"."DEPTNO" IS NOT NULL AND "E"."ENAME"<='M%'
               AND"E"."ENAME">='A%')
```

Sort-Merge Join

Requires both rowsources to be sorted

- Either by a sort operation
- Or sorted by access path (index range and full scan)

Cannot return any rows before both rowsources are sorted (non-cascading)

NL and Hash join should be normally preferred

```
SQL> select /*+ USE_MERGE(d,e) */ d.dname, d.loc, e.empno, e.ename
  2  from emp e, dept d
  3  where e.deptno = d.deptno
  4  and d.dname = 'SALES'
  5  and e.ename between 'A%' and 'X%'
  6  order by e.deptno;
-----------------------------------------------------------------------------
| Id  | Operation                    | Name    | Rows  | Bytes | Cost (%CPU)|
-----------------------------------------------------------------------------
|   0 | SELECT STATEMENT             |         |  1245 | 46065 |    64  (10)|
|   1 |  MERGE JOIN                  |         |  1245 | 46065 |    64  (10)|
|*  2 |   TABLE ACCESS BY INDEX ROWID| DEPT    |     1 |    20 |     2   (0)|
|   3 |    INDEX FULL SCAN           | PK_DEPT |     4 |       |     1   (0)|
|*  4 |   SORT JOIN                  |         |  3735 | 63495 |    62  (10)|
|*  5 |    TABLE ACCESS FULL         | EMP     |  3735 | 63495 |    61   (9)|
-----------------------------------------------------------------------------
```

Conclusion

Identifying problem SQL in the database

- Measure, don’t guess!
- As easy as just querying V$SESSION
  - Remember, a database workload is just a bunch of sessions, running SQL, waiting or working
- @a.sql
- @snapper_v3
- Perfsheet

Identifying the problem inside a SQL

- Measure, don’t guess!
- Don’t use just *explain plan*
- As it may show a wrong plan it doesn’t show the real execution statistics

- *Run* the statement and gather actual *execution statistics*
- Report with DBMS_XPLAN.DISPLAY_CURSOR or @xmsh.sql
- This allows profiling of where most of the response time has been spent
- Also compare real row counts vs estimated row counts

Training and Seminars by Tanel

Advanced Oracle SQL Tuning

- *10-12. May, Singapore (3 days)*
- How to get systematic about Oracle SQL tuning

Parallel Execution and Partitioning for Performance

- *13. May, Singapore*
- How to get the best performance out of partitioning and parallel execution

Advanced Oracle Troubleshooting

- July 2010…
- 3 days of *intensive* database troubleshooting
- How to troubleshoot hangs, crashes, deadlocks, latch,lock contention, bugs and bad performance

http://tech.e2sn.com/oracle-training-seminars

Questions?

Tanel Põder

http://blog.tanelpoder.com

http://www.e2sn.com

OakTable.net

ORACLE CERTIFIED MASTER