

SQL Tuning with 10046 Trace Data and DBMS_XPLAN

Hotsos Enterprises, Ltd.
Grapevine, Texas

Oracle. Performance. Now.
Mahesh.Vallampati@hotsos.com

Agenda

- Goal of SQL Tuning or Optimization
- Approach for SQL Tuning
 - Historical
 - An alternative Approach
- 10046 Trace Data and STAT Lines with STATISTICS_LEVEL
- DBMS_XPLAN
- Method R
- Method R Applied to STAT Lines and Explain Plan
- A note of caution – LIO Versus PIO
- Real World Example
- Next Steps
- Q&A

Goals of SQL Tuning or Optimization

Goal of SQL Tuning or Optimization

- SQL Optimization Goals
 - Sustain or improve the response time for a SQL Statement
 - Consume the least amount of system and database resources for that SQL statement (usually Logical I/Os)
 - Meet or exceed the response time expectation of the user using the SQL statement
 - Know where most of time was spent in the processing of the SQL statement
 - Evaluate alternate options to reduce the time

Approach for SQL Tuning

A Historical Perspective

- SQL Tuning usually has involved the following steps
 - Trial and Error
 - Make changes until you hit upon the solution - usually indexes or statistics
 - Guessing
 - Guess which part of the SQL statement is slow
 - “Look at the Data Model”
 - Study the data model
 - Use Materialized Views or Temporary Tables or Rebuild Tables and Indexes
 - Pre-build or Rebuild the tables or views and query against them
 - Try to rewrite the SQL
 - Make efforts to re-write the SQL

An Alternative Approach for SQL Tuning

- What exactly happens when a SQL Statement is executed by the database?
 - The database breaks down the SQL Statements into several row source operations
- SQL Tuning Approach consists of four steps
 - Which row source operation of the SQL Statement is taking a long time?
 - What part of the SQL Statement is driving it?
 - Why is taking a long time?
 - What needs to be done to address it?
- This paper focuses on the first two steps
- The third and fourth steps are usually easy to figure out once the first two are known

Benefits of the Alternative Approach

- The Oracle Database has instrumentation and diagnostic capabilities built in that provide this information
- The Approach is repeatable and learnable

10046 Trace Data and STAT Lines

Event 10046 instructs the Oracle kernel to emit a sequential record of what it does with your time.

- Event 10046: enable SQL statement timing
 - More accurate: “enable database and system call timing”
- Sequential record logged to a trace file
 - Database calls
 - Oracle “timed events,” if you ask for them (waits - true)
 - Placeholder-value bindings, if you ask for them (binds – true)
 - **Execution Plans for SQL statements (called STAT lines)**

STAT Lines

- STAT lines in the 10046 trace file reveals the execution plan that was chosen by the Oracle query optimizer.
- STAT lines are **individual row source operations** that make up the execution plan
- STAT line examples
 - STAT #7 id=9 cnt=3 pid=4 pos=2 obj=19 op='TABLE ACCESS BY INDEX ROWID IND\$ (cr=8 pr=1 pw=0 time=9528 us)'
 - STAT #7 id=5 cnt=8 pid=4 pos=1 obj=0 op='NESTED LOOPS (cr=696 pr=0 pw=0 time=13664 us)'

STAT Lines and STATISTICS_LEVEL

- Using the timing statistics from the STAT lines in trace files is straightforward.
- Time values are available even if STATISTICS_LEVEL is not set to “all” but not accurate
- Prerequisites
 - STATISTICS_LEVEL=all
 - Will set "_rowsource_execution_statistics"=true
 - Or the time values don't roll up properly

STATISTICS_LEVEL

STATISTICS_LEVEL

- Introduced in 9i R2
- Controls all major statistics collections and advisories in the database
- Database Parameter – Instance wide or session level
- Valid values for STATISTICS_LEVEL
 - BASIC
 - TYPICAL
 - ALL

STATISTICS_LEVEL

- V\$STATISTICS_LEVEL (Described in this view)
- **STATISTICS_LEVEL should not be set to “all” at the instance level**
 - This will cause some stability issues
- Instead, set it at the session level for the SQL statement you are trying to tune

STATISTICS_LEVEL

- BASIC:
 - No advisories or statistics are collected.
- TYPICAL:
 - Buffer cache advisory
 - MTTR advisory
 - Shared pool sizing advisory
 - Segment level statistics
 - PGA target advisory
 - Timed statistics and more in 10g
- ALL:
 - All of TYPICAL, plus the following
 - Timed operating system statistics
 - **Row source execution statistics**

STAT Lines Continued...

Script to Gather Row Source Execution Statistics

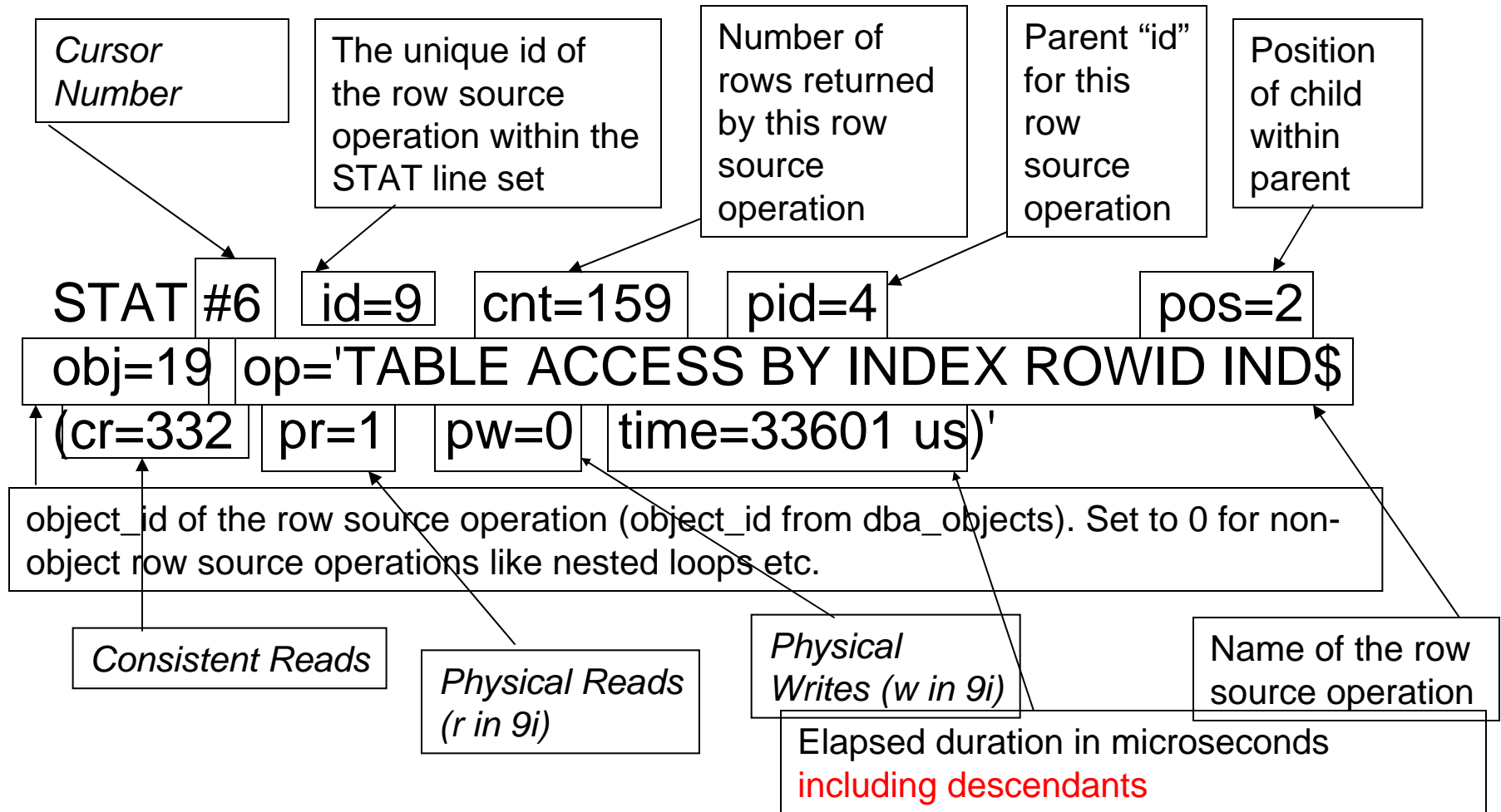
- alter session set timed_statistics = true;
- alter session set max_dump_file_size = unlimited;
- alter session set tracefile_identifier = 'Query1_10046';
- **alter session set statistics_level=all;**
- alter session set events '10046 trace name context forever, level 12'; or dbms_support in 9i or dbms_monitor in 10g
 - Plug SQL statement that needs to be tuned here
 - select object_type, count(*) from dba_objects
 - where owner='SCOTT'
 - group by object_type;
- **Disconnect** ←
- (instead of disabling trace)
- exit;

Instead of disabling trace, just disconnect
Turning off trace before cursor close withholds the
cursor's execution plan information

Real Life STAT Lines

- STAT #6 id=1 cnt=12 pid=0 pos=1 obj=0 op='HASH GROUP BY (cr=1049 pr=10 pw=0 time=141768 us)'
- STAT #6 id=2 cnt=435 pid=1 pos=1 obj=2371 op='VIEW DBA_OBJECTS (cr=1049 pr=10 pw=0 time=139205 us)'
- STAT #6 id=3 cnt=435 pid=2 pos=1 obj=0 op='UNION-ALL (cr=1049 pr=10 pw=0 time=136579 us)'
- STAT #6 id=4 cnt=435 pid=3 pos=1 obj=0 op='FILTER (cr=1038 pr=7 pw=0 time=83601 us)'
- STAT #6 id=5 cnt=465 pid=4 pos=1 obj=0 op='NESTED LOOPS (cr=706 pr=6 pw=0 time=45592 us)'
- STAT #6 id=6 cnt=1 pid=5 pos=1 obj=22 op='TABLE ACCESS BY INDEX ROWID USER\$ (cr=2 pr=0 pw=0 time=48 us)'
- STAT #6 id=7 cnt=1 pid=6 pos=1 obj=44 op='INDEX UNIQUE SCAN I_USER1 (cr=1 pr=0 pw=0 time=23 us)'
- STAT #6 id=8 cnt=465 pid=5 pos=2 obj=18 op='TABLE ACCESS FULL OBJ\$ (cr=704 pr=6 pw=0 time=42730 us)'
- STAT #6 id=9 cnt=159 pid=4 pos=2 obj=19 op='TABLE ACCESS BY INDEX ROWID IND\$ (cr=332 pr=1 pw=0 time=33601 us)'
- STAT #6 id=10 cnt=165 pid=9 pos=1 obj=39 op='INDEX UNIQUE SCAN I_IND1 (cr=167 pr=1 pw=0 time=29671 us)'
- STAT #6 id=11 cnt=0 pid=3 pos=2 obj=0 op='NESTED LOOPS (cr=3 pr=1 pw=0 time=12987 us)'
- STAT #6 id=12 cnt=1 pid=11 pos=1 obj=22 op='TABLE ACCESS BY INDEX ROWID USER\$ (cr=2 pr=0 pw=0 time=50 us)'
- STAT #6 id=13 cnt=1 pid=12 pos=1 obj=44 op='INDEX UNIQUE SCAN I_USER1 (cr=1 pr=0 pw=0 time=20 us)'
- STAT #6 id=14 cnt=0 pid=11 pos=2 obj=107 op='INDEX RANGE SCAN I_LINK1 (cr=1 pr=1 pw=0 time=12917 us)'

STAT Lines Description



11g STAT Lines Enhancement

- Following Added to the STAT lines
 - Cost → Cost Column in V\$SQL_PLAN
 - Card → Cardinality Column in V\$SQL_PLAN
 - Size → Bytes column in v\$SQL_PLAN

Hotsos way of Looking at STAT lines “Profile of Execution Plan”

op (obj)	(calculated)			(time)			(cnt)	(cr)	(pr)	(pw)
	Duration of self			Duration incl descendants			Rows returned	Oracle CR buffer gets (LIO blocks)	Oracle blocks manipulated by OS (PIO blocks)	
	seconds	%	% R	seconds	%	% R			read	written
HASH GROUP BY ()	0.000	1.5%	0.0%	0.027	100.0%	2.4%	2	0	0	0
VIEW DBA_OBJECTS (2371)	0.000	0.2%	0.0%	0.027	98.5%	2.4%	8	0	0	0
UNION-ALL ()	0.000	0.5%	0.0%	0.027	98.3%	2.4%	8	0	0	0
FILTER ()	0.000	0.4%	0.0%	0.023	85.1%	2.1%	8	0	0	0
NESTED LOOPS ()	0.000	0.2%	0.0%	0.014	49.9%	1.2%	8	0	0	0
TABLE ACCESS BY INDEX ROWID USER\$ (22)	0.000	0.1%	0.0%	0.000	0.1%	0.0%	1	1	0	0
INDEX UNIQUE SCAN I_USER1 (44)	0.000	0.1%	0.0%	0.000	0.1%	0.0%	1	1	0	0
TABLE ACCESS FULL OBJ\$ (18)	0.014	49.5%	1.2%	0.014	49.5%	1.2%	8	694	0	0
TABLE ACCESS BY INDEX ROWID IND\$ (19)	0.000	0.3%	0.0%	0.010	34.8%	0.8%	3	3	0	0
INDEX UNIQUE SCAN I_IND1 (39)	0.009	34.4%	0.8%	0.009	34.4%	0.8%	3	5	1	0
NESTED LOOPS ()	0.000	0.1%	0.0%	0.003	12.7%	0.3%	0	0	0	0
TABLE ACCESS BY INDEX ROWID USER\$ (22)	0.000	0.1%	0.0%	0.000	0.2%	0.0%	1	1	0	0
INDEX UNIQUE SCAN I_USER1 (44)	0.000	0.1%	0.0%	0.000	0.1%	0.0%	1	1	0	0
INDEX RANGE SCAN I_LINK1 (107)	0.003	12.5%	0.3%	0.003	12.5%	0.3%	0	1	1	0
Total	0.027	100.0%	2.4%					707	2	0

Blue Rectangle – Cumulative Duration (time) in STAT lines

Green Rectangle – Duration of Individual STAT lines calculated by profiler software

Recap

- 10046 trace files contain STAT Lines
- STAT lines contain execution plans for SQL statements
- STAT lines are nothing but row source operations of real execution plans
- STAT lines contain **cumulative duration** of row source operations including descendants
- For these durations to roll up properly, STATISTICS_LEVEL should be set to all
- Disabling trace or exiting from a session may suppress STAT lines, instead just disconnect
- Timing of individual row source operations should be carefully calculated to avoid over or undercounting
- **The Hotsos Profiler displays the STAT lines in an easy to read and understand format and calculates the individual durations of row source operations**

DBMS_XPLAN

DBMS_XPLAN

- Introduced in 9iR2
- Easy way of viewing the output of the EXPLAIN PLAN command in several, predefined formats
- 10g has some enhanced functionality
- We will use the DISPLAY function of DBMS_XPLAN available in 9iR2 and 10g

DBMS_XPLAN

- Ensure that you are using the right version of the
 - PLAN_TABLE
 - DBMS_XPLAN
- Version incompatibilities may cause some issues
- Where is plan table and DBMS_XPLAN?
 - Plan Table - \$ORACLE_HOME/rdbms/admin/utlxplan.sql
 - DBMS_XPLAN - \$ORACLE_HOME/rdbms/admin/dbmsxplan.sql

Using DBMS_XPLAN

```
EXPLAIN PLAN  
SET STATEMENT_ID = 'abc'  
FOR  
select object_type, count(1)  
from dba_objects  
where owner= 'SCOTT'  
group by object_type;
```

DBMS_XPLAN.DISPLAY Options

Format parameter for this function choices are

- BASIC - Just displays the minimum information in the plan
- Typical - Displays the relevant information in the plan and predicate information. (Parallel Execution information if applicable)
- Serial – Like typical, but no parallel execution information even if applicable
- All – All of typical including projections, alias, etc.

DBMS_XPLAN. DISPLAY Examples

- `SELECT * FROM
TABLE(dbms_xplan.display('PLAN_TABLE','abc','BASIC'));`
- `SELECT * FROM
TABLE(dbms_xplan.display('PLAN_TABLE','abc','TYPICAL'));`
- `SELECT * FROM
TABLE(dbms_xplan.display('PLAN_TABLE','abc','ALL'));`

Note this is different from the STATISTICS_LEVEL parameter
These are display formats for DBMS_XPLAN

DBMS_XPLAN.TYPICAL Output

Id	Operation	Name	Rows	Bytes	Cost (%CPU)	Time
0	SELECT STATEMENT		1788	50064	163 (5)	00:00:02
1	HASH GROUP BY		1788	50064	163 (5)	00:00:02
2	VIEW	DBA_OBJECTS	1788	50064	162 (5)	00:00:02
3	UNION-ALL					
* 4	FILTER					
5	NESTED LOOPS		1943	176K	160 (5)	00:00:02
6	TABLE ACCESS BY INDEX ROWID	USER\$	1	14	1 (0)	00:00:01
* 7	INDEX UNIQUE SCAN	I_USER1	1		0 (0)	00:00:01
* 8	TABLE ACCESS FULL	OBJ\$	1943	149K	159 (5)	00:00:02
* 9	TABLE ACCESS BY INDEX ROWID	IND\$	1	8	2 (0)	00:00:01
* 10	INDEX UNIQUE SCAN	I_IND1	1		1 (0)	00:00:01
11	NESTED LOOPS		2	34	2 (0)	00:00:01
12	TABLE ACCESS BY INDEX ROWID	USER\$	1	14	1 (0)	00:00:01
* 13	INDEX UNIQUE SCAN	I_USER1	1		0 (0)	00:00:01
* 14	INDEX RANGE SCAN	I_LINK1	2	6	1 (0)	00:00:01

Predicate Information (identified by operation id):

```

4 - filter("O"."TYPE#"<>1 AND "O"."TYPE#"<>10 OR "O"."TYPE#"=1 AND (SELECT 1 FROM
      "SYS"."IND$" "I" WHERE "I"."OBJ#"=:B1 AND ("I"."TYPE#"=1 OR "I"."TYPE#"=2 OR
      "I"."TYPE#"=3 OR "I"."TYPE#"=4 OR "I"."TYPE#"=6 OR "I"."TYPE#"=7 OR "I"."TYPE#"=9))=1)
7 - access("U"."NAME"='SCOTT')
8 - filter("O"."NAME"<>'_NEXT_OBJECT' AND "O"."NAME"<>'_default_auditing_options_'
      AND "O"."LINKNAME" IS NULL AND "O"."OWNER#"="U"."USER#")
9 - filter("I"."TYPE#"=1 OR "I"."TYPE#"=2 OR "I"."TYPE#"=3 OR "I"."TYPE#"=4 OR
      "I"."TYPE#"=6 OR "I"."TYPE#"=7 OR "I"."TYPE#"=9)
10 - access("I"."OBJ#"=:B1)
13 - access("U"."NAME"='SCOTT')
14 - access("L"."OWNER#"="U"."USER#")

```

Recap

- DBMS_XPLAN has been around since 9.2
- It is an elegant way of looking at Explain Plan outputs
- The “typical” option of DBMS_XPLAN.DISPLAY is the most useful as it ties predicate information (access and filters) back to the predicted row source operation lines in the explained plan

Method R

Method R

- Method R
 1. Target the tasks for which the business needs improvement.
 2. Collect properly scoped diagnostic data while the tasks are being slow.
 3. React with the candidate repair that will have the greatest net payoff to the business.
 - Stop if the cost of the repair exceeds the cost of the problem.
 4. Go to step 1.

Method R

- Tells you where to start
- Tells you when to stop
 - So you can avoid “Compulsive Tuning Disorder”

Method R Applied to STAT lines and Explain Plan

EXPLAIN PLAN and STAT Lines (Execution Plan) Side by Side

Explain Plan					Execution Plan					
Id	Operation	Name	Rows	Cost	Row Source Operation	Seconds	% of Duration	Cumulative	Rows	CR (LIO) Blocks
0	SELECT STATEMENT		1788	163	Select Statement	0	100%	0.027	2	707
1	HASH GROUP BY		1788	163	HASH GROUP BY	0	1.50%	0.027	2	0
2	VIEW	DBA_OBJECTS	1788	162	VIEW DBA_OBJECTS (2371)	0	0.20%	0.027	8	0
3	UNION-ALL				UNION-ALL	0	0.50%	0.023	8	0
4	FILTER				FILTER	0	0.40%	0.014	8	0
5	NESTED LOOPS		1943	160	NESTED LOOPS	0	0.20%	0	8	0
6	TABLE ACCESS BY INDEX ROWID	USER\$	1	1	TABLE ACCESS BY INDEX ROWID USER\$ (22)	0	0.10%	0	1	1
7	INDEX UNIQUE SCAN	I_USER1	1	0	INDEX UNIQUE SCAN I_USER1 (44)	0	0.10%	0.014	1	1
8	TABLE ACCESS FULL	OBJ\$	1943	159	TABLE ACCESS FULL OBJ\$ (18)	0.014	49.50%	0.01	8	694
9	TABLE ACCESS BY INDEX ROWID	IND\$	1	2	TABLE ACCESS BY INDEX ROWID IND\$ (19)	0	0.30%	0.009	3	3
10	INDEX UNIQUE SCAN	I_IND1	1	1	INDEX UNIQUE SCAN I_IND1 (39)	0.009	34.40%	0.003	3	5
11	NESTED LOOPS		2	2	NESTED LOOPS	0	0.10%	0	0	0
12	TABLE ACCESS BY INDEX ROWID	USER\$	1	1	TABLE ACCESS BY INDEX ROWID USER\$ (22)	0	0.10%	0	1	1
13	INDEX UNIQUE SCAN	I_USER1	1	0	INDEX UNIQUE SCAN I_USER1 (44)	0	0.10%	0	1	1
14	INDEX RANGE SCAN	I_LINK1	2	1	INDEX RANGE SCAN I_LINK1 (107)	0.003	12.50%	0.003	0	1

First Question

- Do the execution plan row source operation steps in the STAT lines match the Explain Plan row source operation steps?
- If not, why?
 - Skewed data
 - Statistics issue
 - Optimizer bug
 - Bind variable mismatch
 - Explain Plan session optimizer settings different from the settings of the session which generated the trace file

Apply Method R to this Profiler View of STAT Lines

- Which row source operations are consuming most of the response time?
- **What access predicate is driving it?**
- Why is it consuming most of the response time?
- Is that the right row source operation?
 - e.g. nested loops versus hash joins or vice versa
- Why are so many blocks being scanned to get a small subset of rows?
- If the index is being used and if it is doing a range scan over a large set of blocks, is that the right index to be used?
- Can an alternate index help or are we better off doing a full table scan?
- **Is there a big difference in the actual rows in the execution plan versus the expected rows from the explain plan?**

Method R view of STAT lines and explain plan

Explain Plan					Execution Plan					
Id	Operation	Name	Rows	Cost	Row Source Operation	Seconds	% of Duration	Cumulative	Rows	CR (LIO) Blocks
0	SELECT STATEMENT		1788	163	Select Statement	0	100%	0.027	2	707
1	HASH GROUP BY		1788	163	HASH GROUP BY	0	1.50%	0.027	2	0
2	VIEW	DBA_OBJECTS	1788	162	VIEW DBA_OBJECTS (2371)	0	0.20%	0.027	8	0
3	UNION-ALL				UNION-ALL	0	0.50%	0.023	8	0
4	FILTER				FILTER	0	0.40%	0.014	8	0
5	NESTED LOOPS		1943	160	NESTED LOOPS	0	0.20%	0	8	0
6	TABLE ACCESS BY INDEX ROWID	USER\$	1	1	TABLE ACCESS BY INDEX ROWID USER\$ (22)	0	0.10%	0	1	1
7	INDEX UNIQUE SCAN	I_USER1	1	0	INDEX UNIQUE SCAN I_USER1 (44)	0	0.10%	0.014	1	1
8	TABLE ACCESS FULL	OBJ\$	1943	159	TABLE ACCESS FULL OBJ\$ (18)	0.014	49.50%	0.01	8	604
9	TABLE ACCESS BY INDEX ROWID	IND\$	1	2	TABLE ACCESS BY INDEX ROWID IND\$ (19)	0	0.30%	0.009	3	3
10	INDEX UNIQUE SCAN	I_IND1	1	1	INDEX UNIQUE SCAN I_IND1 (39)	0.009	34.40%	0.003	3	5
11	NESTED LOOPS		2	2	NESTED LOOPS	0	0.10%	0	0	0
12	TABLE ACCESS BY INDEX ROWID	USER\$	1	1	TABLE ACCESS BY INDEX ROWID USER\$ (22)	0	0.10%	0	1	1
13	INDEX UNIQUE SCAN	I_USER1	1	0	INDEX UNIQUE SCAN I_USER1 (44)	0	0.10%	0	1	1
14	INDEX RANGE SCAN	I_LINK1	2	1	INDEX RANGE SCAN I_LINK1 (107)	0.003	12.50%	0.003	0	1

Predicate Information (identified by operation id):

```

4 - filter("o"."TYPE#"<>1 AND "o"."TYPE#"<>10 OR "o"."TYPE# "=1 AND (SELECT 1 FROM
      "SYS"."IND$" "I" WHERE "I"."OBJ#"=:B1 AND ("I"."TYPE#"=1 OR "I"."TYPE#"=2 OR
      "I"."TYPE#"=3 OR "I"."TYPE#"=4 OR "I"."TYPE#"=6 OR "I"."TYPE#"=7 OR "I"."TYPE#"=9))=1)
7 - access("u"."NAME"='SCOTT')
8 - filter("o"."NAME" <> '_NEXT_OBJECT' AND "o"."NAME" <> '_default_auditing_options_'
      AND "o"."LINKNAME" IS NULL AND "o"."OWNER#"="u"."USER#")
9 - filter("I"."TYPE#"=1 OR "I"."TYPE#"=2 OR "I"."TYPE#"=3 OR "I"."TYPE#"=4 OR
      "I"."TYPE#"=6 OR "I"."TYPE#"=7 OR "I"."TYPE#"=9)
10 - access("I"."OBJ#"=:B1)
13 - access("u"."NAME"='SCOTT')
14 - access("L"."OWNER#"="u"."USER#")
  
```

Notice wide difference in row count estimated by the plan and the actual

Recap

- Tying back STAT lines (real row source operations) in trace files (execution plan) to the explain plan output (predicted row source operations) using DBMS_XPLAN should be the first step in SQL optimization
- Applying Method R to this view helps us find out which row source operations are driving most of the response time and the corresponding predicates driving the response time can be obtained from DBMS_XPLAN
- Large discrepancies in the actual rows returned in the execution plan and the estimated rows in the execution plan will point to either gaps in the statistics gathering process or bugs in the optimizer

A note of caution – LIO Versus PIO

A Note of Caution (LIO Versus PIO)

Row source operation (object id)	Duration of self			Duration incl descendants			Rows returned	Oracle CR buffer gets (LIO blocks)	Oracle blocks manipulated by OS (PIO blocks)	
	seconds	%	% R	seconds	%	% R			read	written
HASH GROUP BY ()	0.000	1.5%	0.0%	0.027	100.0%	2.4%	2	0	0	0
VIEW DBA_OBJECTS (2371)	0.000	0.2%	0.0%	0.027	98.5%	2.4%	8	0	0	0
UNION-ALL ()	0.000	0.5%	0.0%	0.027	98.3%	2.4%	8	0	0	0
FILTER ()	0.000	0.4%	0.0%	0.023	85.1%	2.1%	8	0	0	0
NESTED LOOPS ()	0.000	0.2%	0.0%	0.014	49.9%	1.2%	8	0	0	0
TABLE ACCESS BY INDEX ROWID USER\$ (22)	0.000	0.1%	0.0%	0.000	0.1%	0.0%	1	1	0	0
INDEX UNIQUE SCAN I_USER1 (44)	0.000	0.1%	0.0%	0.000	0.1%	0.0%	1	1	0	0
TABLE ACCESS FULL OBJ\$ (18)	0.014	49.5%	1.2%	0.014	49.5%	1.2%	8	694	0	0
TABLE ACCESS BY INDEX ROWID IND\$ (19)	0.000	0.3%	0.0%	0.010	34.8%	0.8%	3	3	0	0
INDEX UNIQUE SCAN I_IND1 (39)	0.009	34.4%	0.8%	0.009	34.4%	0.8%	3	5	1	0
NESTED LOOPS ()	0.000	0.1%	0.0%	0.003	12.7%	0.3%	0	0	0	0
TABLE ACCESS BY INDEX ROWID USER\$ (22)	0.000	0.1%	0.0%	0.000	0.2%	0.0%	1	1	0	0
INDEX UNIQUE SCAN I_USER1 (44)	0.000	0.1%	0.0%	0.000	0.1%	0.0%	1	1	0	0
INDEX RANGE SCAN I_LINK1 (107)	0.003	12.5%	0.3%	0.003	12.5%	0.3%	0	1	1	0
Total	0.027	100.0%	2.4%					707	2	0

In the lines that are highlighted, what % of the time was due to physical I/O and not logical I/Os?

LIO Versus PIO

- White paper by Cary Millsap
 - [Why You Should Focus on LIOs Instead of PIOs](#)
 - Available on hotsos.com Library
- Cary's white paper clearly documents why
 - You need to focus on response time keeping logical I/O's in mind
 - Even after eliminating physical I/O's there is still an opportunity to make the SQL statement run faster
 - A focus on LIO reduction automatically drives PIO reduction
 - <http://www.hotsos.com/e-library/abstract.php?id=7>

LIO Versus PIO

- SQL statements consume logical I/Os which drives physical I/O consumption
- Physical I/O is a secondary derived effect
- When the query is run the first time, depending on the data cached in the database, the physical I/O may dominate the components of the response time
- When you run the query the second time, the data should be cached and you will see a better picture of the inefficient row source operations triggered by Logical I/O's.

LIO Versus PIO

- To get the right profile of the execution plan from a Logical I/O perspective, we need to do the following.
 - Execute the query once
 - This will parse the query
 - The buffer pool will cache the data
 - Execute the query again
 - This time the STAT lines show the effect of Logical I/Os on the SQL execution response time

Real World Example

Real World Example – SQL Statement

```
SELECT iu.ship_product_id
AS part_number, iu.merge_id AS fc_code,
oa.country_code, COUNT (1) quantity
FROM shipping.shippable sh,
shipping.order_address oa,
shipping.item_unit iu
WHERE
sh.manifest_date >
TRUNC (SYSTIMESTAMP AT TIME ZONE 'GMT')- (7 * 13)
AND oa.order_id = sh.order_id
AND oa.addr_type_code = 'S'
AND iu.order_id = oa.order_id
AND iu.merge_id IS NOT NULL
GROUP BY iu.ship_product_id, iu.merge_id, oa.country_code
```

Profiler View of Execution Plan

Row	source operation (object id)	Duration of self			Duration incl descendants			Rows returned	Oracle CR buffer gets (LIO blocks)	Oracle blocks manipulated by OS (PIO blocks)	
		seconds	%	% R	seconds	%	% R			read	written
1.	SORT GROUP BY ()	0.210	0.0%	0.0%	1,968.896	100.0%	98.9%	409	0	0	0
2.	HASH JOIN ()	877.500	44.6%	44.1%	1,968.685	100.0%	98.9%	24,886	0	0	0
3.	HASH JOIN ()	326.902	16.6%	16.4%	516.963	26.3%	26.0%	496	0	0	0
4.	PARTITION RANGE ALL PARTITION: 1 16 ()	0.010	0.0%	0.0%	0.296	0.0%	0.0%	496	0	0	0
5.	TABLE ACCESS BY LOCAL INDEX ROWID OBJ#(6873) PARTITION: 1 16 (6873)	0.189	0.0%	0.0%	0.286	0.0%	0.0%	496	407	35	0
6.	INDEX RANGE SCAN OBJ#(6959) PARTITION: 1 16 (6959)	0.097	0.0%	0.0%	0.097	0.0%	0.0%	496	21	3	0
7.	INDEX FAST FULL SCAN OBJ#(171583) (171583)	189.765	9.6%	9.5%	189.765	9.6%	9.5%	16,672,887	132,946	132,882	0
8.	INDEX FAST FULL SCAN OBJ#(171589) (171589)	574.222	29.2%	28.9%	574.222	29.2%	28.9%	44,875,302	377,551	377,134	0
9.	Total	1,968.896	100.0%	98.9%					510,925	510,054	0

- *Line 2 and Line 8 are the expensive row source operations*
- *Notice the calculation of the individual duration of the row source operations*

DBMS_XPLAN

PLAN_TABLE_OUTPUT

```
-----  
-----  
| Id | Operation | Name | R |  
-----  
| 0 | SELECT STATEMENT | | |  
| 1 | SORT GROUP BY | | |  
|* 2 | HASH JOIN | | |  
|* 3 | HASH JOIN | | |  
| 4 | PARTITION RANGE ALL | | |  
| 5 | TABLE ACCESS BY LOCAL INDEX ROWID | SHIPPABLE | |  
|* 6 | INDEX RANGE SCAN | SHIPPABLE_MANIFEST_DATE_IDX | |  
|* 7 | INDEX FAST FULL SCAN | OA_COUNT_ADDR_IDX | |  
|* 8 | INDEX FAST FULL SCAN | IU_PI_MI_OI_IDX | |  
-----
```

Predicate Information (identified by operation id):

```
-----  
2 - access("IU"."ORDER_ID"="OA"."ORDER_ID")  
3 - access("OA"."ORDER_ID"="SH"."ORDER_ID")  
-----
```

PLAN_TABLE_OUTPUT

```
-----  
6 - access("SH"."MANIFEST_DATE">TRUNC(SYSTIMESTAMP(6) AT TIME ZONE 'GMT ') -91  
7 - filter("OA"."ADDR_TYPE_CODE"='S')  
8 - filter("IU"."MERGE_ID" IS NOT NULL)  
-----
```

Note: cpu costing is off

SQL Statement – Access Predicates Highlighted

```
SELECT iu.ship_product_id
AS part_number, iu.merge_id AS fc_code,
oa.country_code, COUNT (1) quantity
FROM shipping.shippable sh,
shipping.order_address oa,
shipping.item_unit iu
WHERE
sh.manifest_date > TRUNC (SYSTIMESTAMP AT TIME ZONE 'GMT')- (7 * 13)
AND oa.order_id = sh.order_id
AND oa.addr_type_code = 'S'
AND iu.order_id = oa.order_id
AND iu.merge_id IS NOT NULL
GROUP BY iu.ship_product_id, iu.merge_id, oa.country_code
```

Recap

- Run the query once to parse the query and warm up the buffer pool with the data blocks
- Trace it the second time with `STATISTICS_LEVEL=all`
- STAT Lines in trace files contain timing information of row source operations. These timings includes descendants of the row source operations.
- Calculate the individual row source operations timings using the Profiler software
- Obtain the explain plan for the SQL statement using `DBMS_XPLAN`
- Put them side by side and see which access predicates are driving the row source operations that are taking the most time

Next Steps

- If you have a SQL statement that you think can be made to run faster
 - Upload a zip file containing the following
 - 10046 Trace file for SQL statement
 - Output of DBMS_XPLAN
 - To the Profiler Portal at
 - <https://secure.hotsos.com/secure/profiler/fman.php>
 - Send an e-mail to webinar@hotsos.com
 - We will schedule a time to review it with you
- Education
 - We recommend the Optimizing Oracle SQL Intensive Class
 - <https://portal.hotsos.com/education/OPINT>

A large, stylized graphic featuring a black 'Q' and a red 'A' that overlap. The 'Q' is positioned above the 'A', and they are both rendered in a thick, rounded font style.

Q U E S T I O N S
A N S W E R S