INTRODUCTION TO QUERY PERFORMANCE TUNING: A 12 STEP PROGRAM

JANIS GRIFFIN

SENIOR DBA / PERFORMANCE EVANGELIST

WHO AM I?

- Senior DBA / Performance Evangelist for Solarwinds
  - Janis.Griffin@solarwinds.com
  - Twitter - @DoBoutAnything
  - Current – 25+ Years in Oracle, Sybase, SQL Server
  - DBA and Developer
- Specialize in Performance Tuning
- Review Database Performance for Customers and Prospects
- Common Thread – Paralyzed by Tuning

© 2015 SOLARWINDS WORLDWIDE, LLC. ALL RIGHTS RESERVED.

AGENDA

- Challenges Of Tuning
  - Who should tune
  - Which SQLs to tune
- Utilize Response Time Analysis (RTA)
  - Wait Events / Wait Time
- 12 Steps To Follow
- Several Case Studies

© 2015 SOLARWINDS WORLDWIDE, LLC. ALL RIGHTS RESERVED.

CHALLENGES OF TUNING

- » SQL Tuning is Hard
  - Who should tune – DBA or Developer
  - Which SQL to tune
- » Requires Expertise in Many Areas
  - Technical – Plan, Data Access, SQL Design
  - Business – What is the Purpose of SQL?
- » Tuning Takes Time
  - Large Number of SQL Statements
  - Each Statement is Different
- » Low Priority in Some Companies
  - Vendor Applications
  - Focus on Hardware or System Issues
- » Never Ending

© 2015 SOLARWINDS WORLDWIDE, LLC. ALL RIGHTS RESERVED.

Image courtesy of Gentle-Stress-Relief.com

© 2015 SOLARWINDS WORLDWIDE, LLC. ALL RIGHTS RESERVED.

1. FIND WHICH SQL TO TUNE

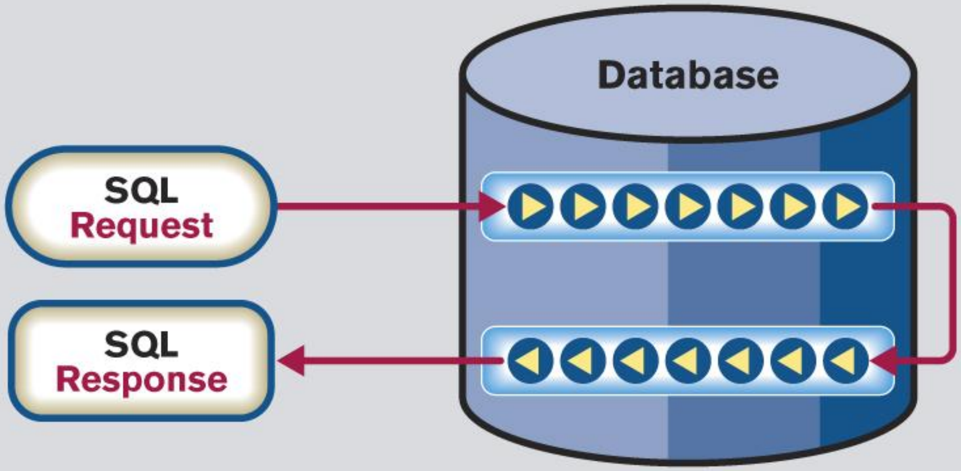
Methods for Identifying

- User / Batch Job Complaints
  - Known Poorly Performing SQL
  - Trace Session/Process
- Queries Performing Most I/O (Buffer Gets, Disk Reads)
  - Table or Index Scans
- Queries Consuming CPU
- Highest Response Times - DPA (formally Ignite)

© 2015 SOLARWINDS WORLDWIDE, LLC. ALL RIGHTS RESERVED.

RESPONSE TIME ANALYSIS (RTA)

Focus on Response Time



Identify Wait-Time at every step and rank bottlenecks by user impact.

- Understand the total time a Query spends in Database
- Measure time while Query executes
- Oracle helps by providing Wait Events

© 2015 SOLARWINDS WORLDWIDE, LLC. ALL RIGHTS RESERVED.

WHAT ARE WAIT EVENTS

- Events have 0-3 parameters that provide more information
  - Example: db file sequential read – P1=file#, P2=block#, P3=blocks
- Knowing what a query waits on - gives a starting place
  - Locking issues may lead to a different solution
  - Than if it were waiting on disk reads
- Oracle 10g – 800+ wait events
- Oracle 11g – 1100+ wait events
- Oracle 12c – 1500+ wait events
- Good news: only need to know small (very small) subset of them
  - If you know the top 10 or so, it will take you a long way

© 2015 SOLARWINDS WORLDWIDE, LLC. ALL RIGHTS RESERVED.

WAIT EVENT INFORMATION

V$SESSION

SID
SERIAL#
USERNAME
MACHINE
PROGRAM
MODULE
ACTION
SQL_ID
PLAN_HASH_VALUE
EVENT
P1TEXT
P1
P2TEXT
P2
P3TEXT
P3
STATE (WAITING, WAITED)
BLOCKING_SESSION

V$SQL

SQL_ID
SQL_FULLTEXT

V$SQL_PLAN

SQL_ID
PLAN_HASH_VALUE
OPERATION
OBJECT_NAME

V$SQL_BIND_CAPTURE

SQL_ID
NAME
DATATYPE_STRING
VALUE_STRING

V$SQLAREA

SQL_ID
EXECUTIONS
PARSE_CALLS
BUFFER_GETS
DISK_READS

DBA_OBJECTS

OBJECT_ID
OBJECT_NAME
OBJECT_TYPE

© 2015 SOLARWINDS WORLDWIDE, LLC. ALL RIGHTS RESERVED.

BASE QUERY

```
INSERT INTO rta_data
SELECT
    sid, serial#, username, program, module, action,
    machine, osuser, sql_id, blocking_session,
    decode(state, ’WAITING’, event, ’CPU’) event,
    p1, p1text, p2, p2text,
    etc…,
    SYSDATE
FROM V$SESSION s
WHERE s.status = ’ACTIVE’
AND wait_class != ’Idle’
AND username != USER;
```

© 2015 SOLARWINDS WORLDWIDE, LLC. ALL RIGHTS RESERVED.

ACTIVE SESSION HISTORY (ASH)

- V$ACTIVE_SESSION_HISTORY
  - Data warehouse for session statistics
  - Oracle 10g and higher
  - Data is sampled every second
  - Holds at least one hour of history
  - Never bigger than:
    - 2% of SGA_TARGET
    - 5% of SHARED_POOL (if automatic sga sizing is turned off)

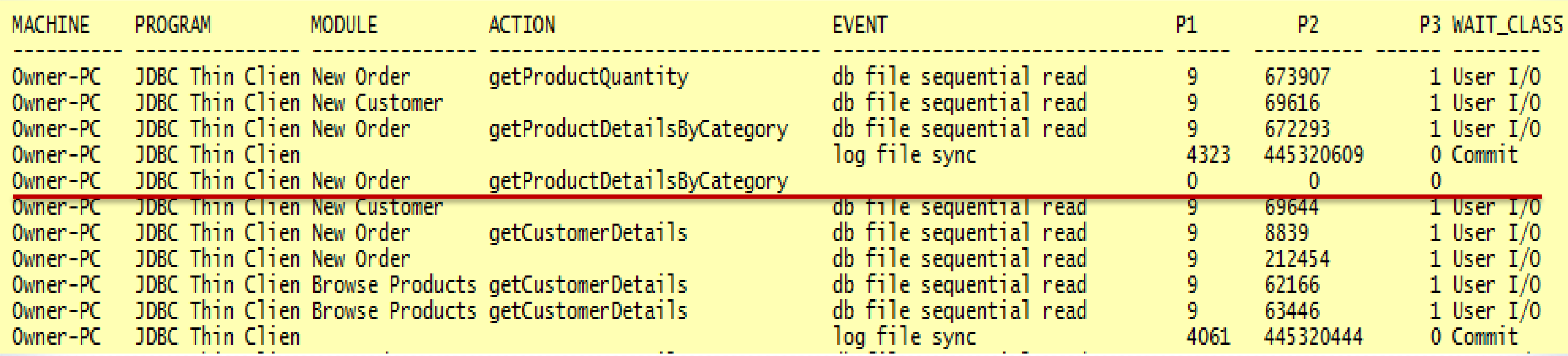
- WRH$_ACTIVE_SESSION_HISTORY
  - Above table gets flushed to this table

© 2015 SOLARWINDS WORLDWIDE, LLC. ALL RIGHTS RESERVED.

V$ACTIVE_SESSION_HISTORY

» Details of what happened between 12:07 and 12:18 am

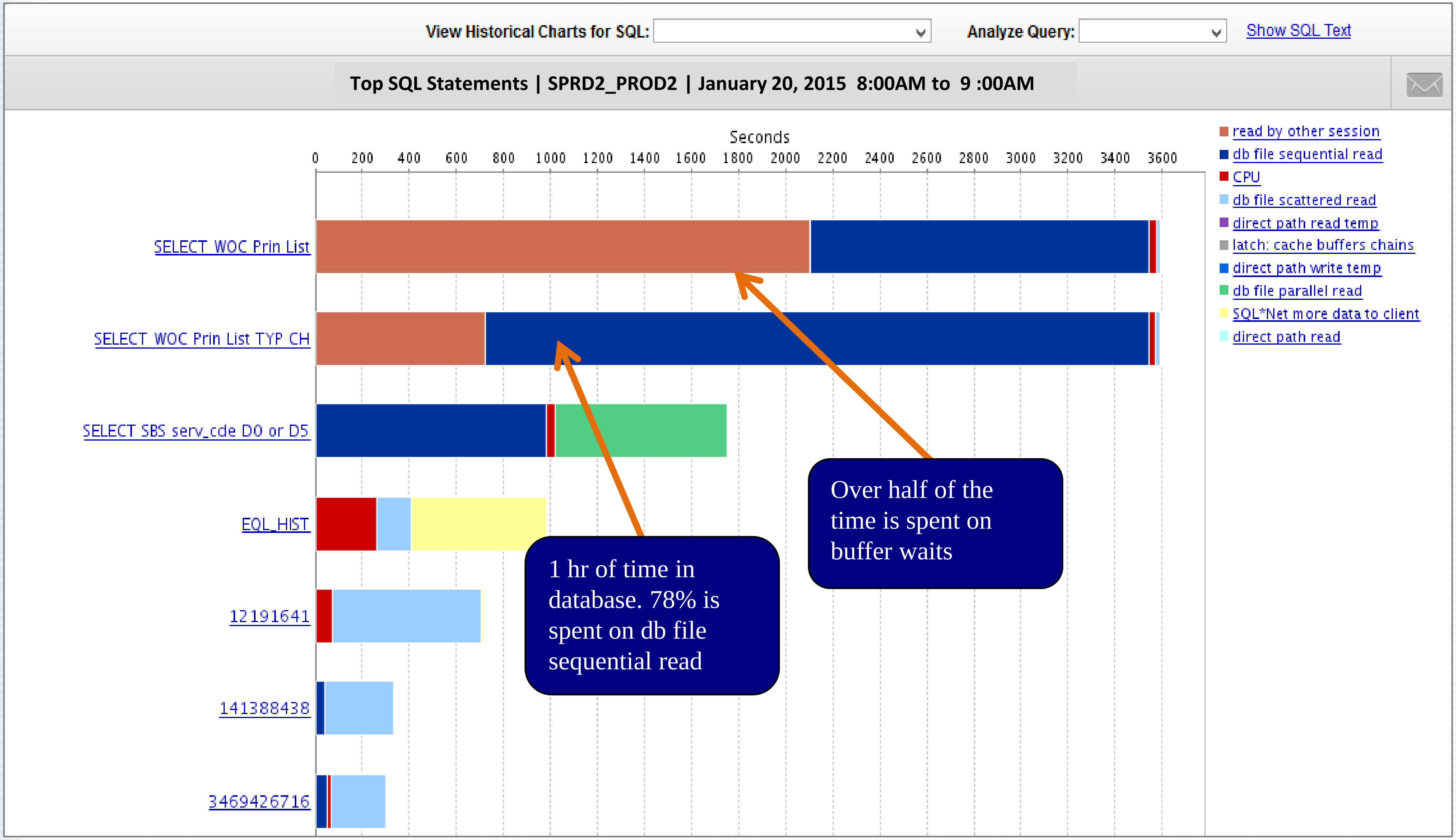
```
SELECT
        --s.sql_id, sql.sql_text, s.session_id, s.user_id,
          s.machine, s.program, s.module,
          s.action, s.blocking_session, s.event, s.p1, s.p2, s.p3, s.wait_class
FROM v$active_session_history s
LEFT OUTER JOIN v$sql sql  ON  s.sql_id   = sql.sql_id
                                   AND s.sql_child_number  =  sql.child_number
WHERE s.session_type <> 'BACKGROUND'
AND s.sample_time  BETWEEN TO_DATE(‘03/31/15 12:07', 'mm/dd/yy hh24:mi')
                              AND     TO_DATE(‘03/31/15 12:18','mm/dd/yy hh24:mi')
```



```
MACHINE    PROGRAM         MODULE          ACTION                         EVENT                          P1            P2     P3 WAIT_CLASS
---------- --------------- --------------- ------------------------------ ------------------------------ ----- ---------- ------ --------
Owner-PC   JDBC Thin Clien New Order       getProductQuantity             db file sequential read          9   673907          1 User I/O
Owner-PC   JDBC Thin Clien New Customer                                   db file sequential read          9   69616           1 User I/O
Owner-PC   JDBC Thin Clien New Order       getProductDetailsByCategory    db file sequential read          9   672293          1 User I/O
Owner-PC   JDBC Thin Clien                                                log file sync                 4323   445320609       0 Commit
Owner-PC   JDBC Thin Clien New Order       getProductDetailsByCategory                                     0           0       0
Owner-PC   JDBC Thin Clien New Customer                                   db file sequential read          9   69644           1 User I/O
Owner-PC   JDBC Thin Clien New Order       getCustomerDetails             db file sequential read          9   8839            1 User I/O
Owner-PC   JDBC Thin Clien New Order                                      db file sequential read          9   212454          1 User I/O
Owner-PC   JDBC Thin Clien Browse Products getCustomerDetails             db file sequential read          9   62166           1 User I/O
Owner-PC   JDBC Thin Clien Browse Products getCustomerDetails             db file sequential read          9   63446           1 User I/O
Owner-PC   JDBC Thin Clien                                                log file sync                 4061   445320444       0 Commit
```

© 2015 SOLARWINDS WORLDWIDE, LLC. ALL RIGHTS RESERVED.

RTA - WAIT TIME & EVENTS



View Historical Charts for SQL:

Analyze Query:

Show SQL Text

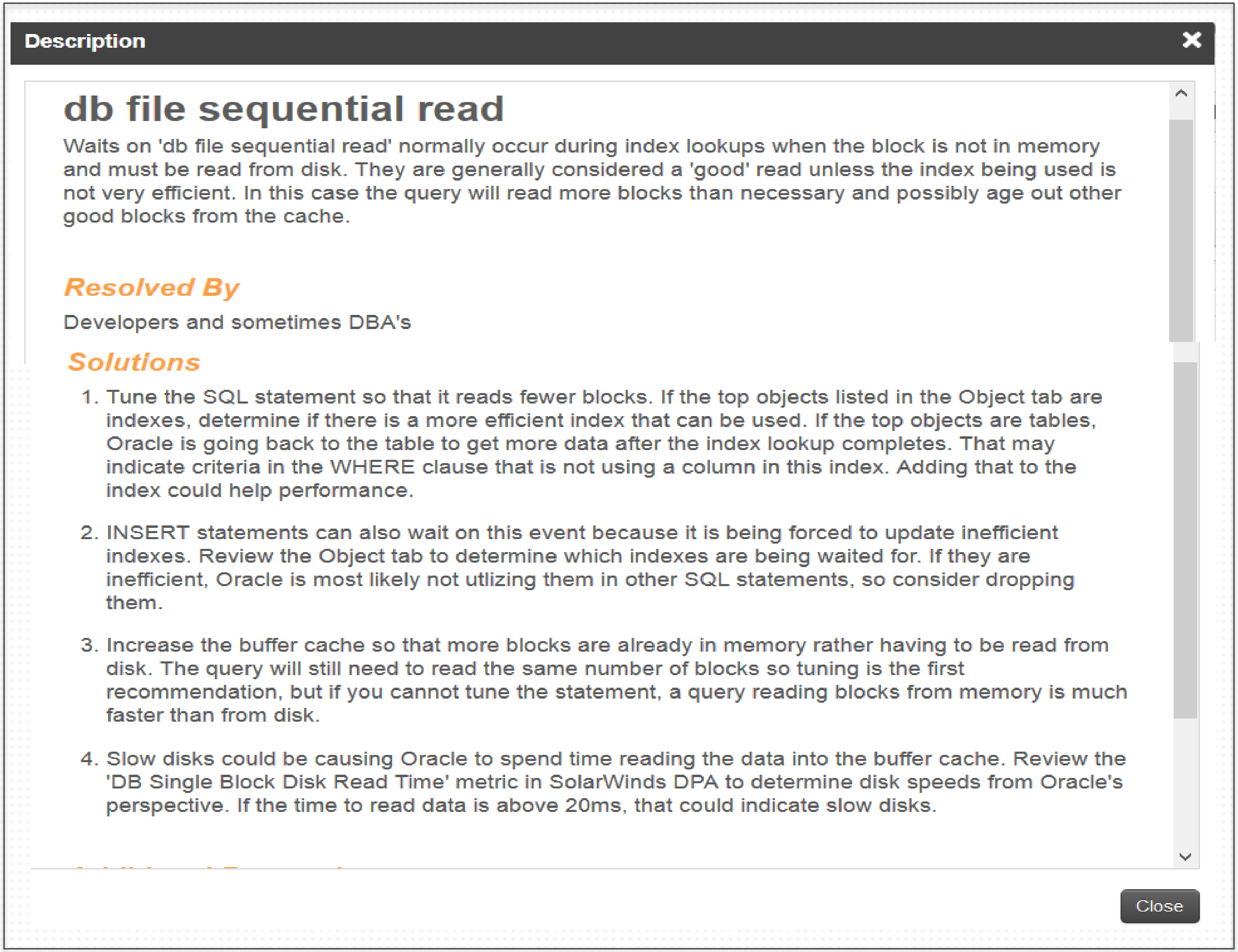
Top SQL Statements | SPRD2_PROD2 | January 20, 2015 8:00AM to 9 :00AM

1 hr of time in database. 78% is spent on db file sequential read

Over half of the time is spent on buffer waits

© 2015 SOLARWINDS WORLDWIDE, LLC. ALL RIGHTS RESERVED.

RTA BENEFITS



db file sequential read

Waits on 'db file sequential read' normally occur during index lookups when the block is not in memory and must be read from disk. They are generally considered a 'good' read unless the index being used is not very efficient. In this case the query will read more blocks than necessary and possibly age out other good blocks from the cache.

Resolved By

Developers and sometimes DBA's

Solutions

1. Tune the SQL statement so that it reads fewer blocks. If the top objects listed in the Object tab are indexes, determine if there is a more efficient index that can be used. If the top objects are tables, Oracle is going back to the table to get more data after the index lookup completes. That may indicate criteria in the WHERE clause that is not using a column in this index. Adding that to the index could help performance.

2. INSERT statements can also wait on this event because it is being forced to update inefficient indexes. Review the Object tab to determine which indexes are being waited for. If they are inefficient, Oracle is most likely not utlizing them in other SQL statements, so consider dropping them.

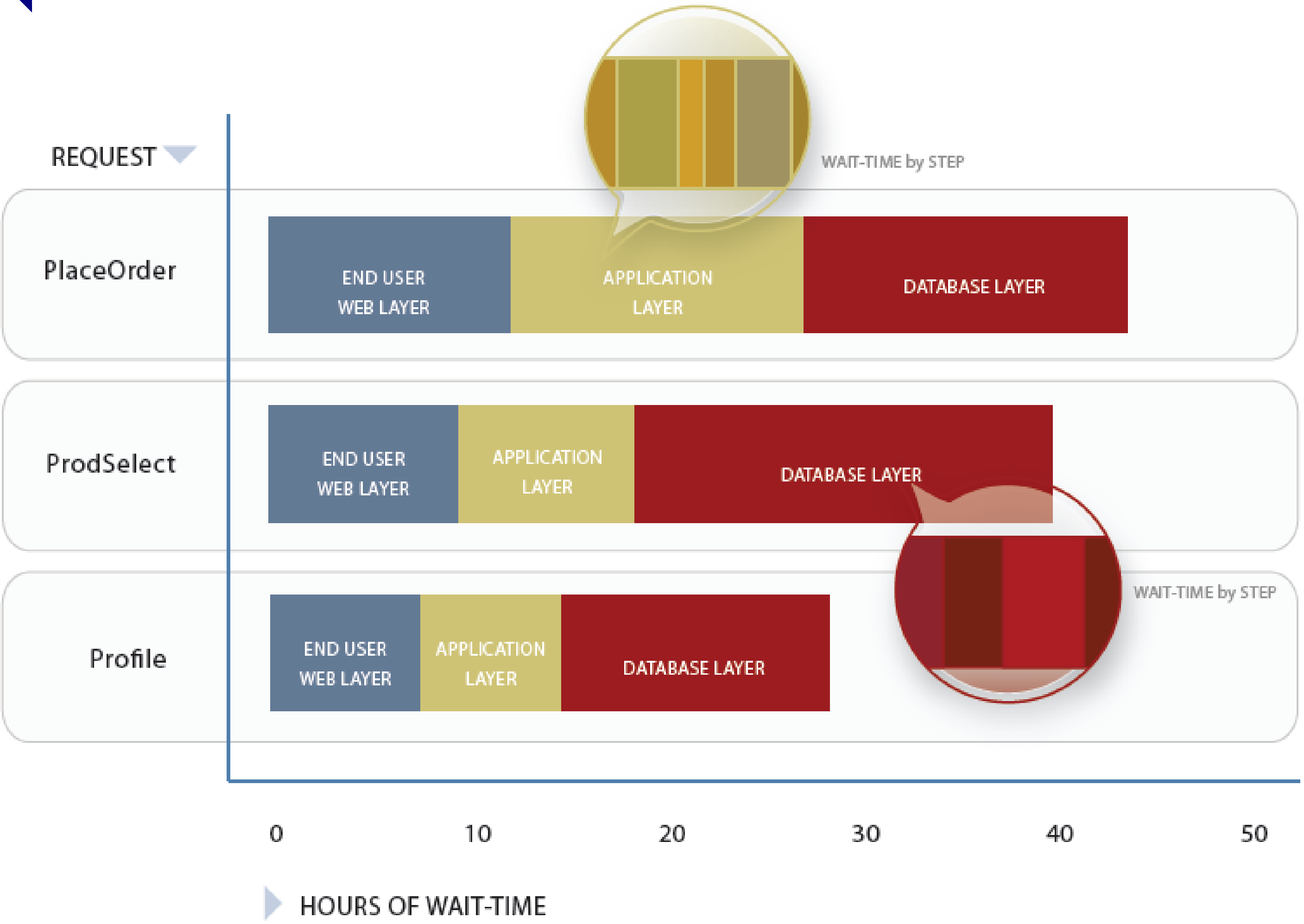
3. Increase the buffer cache so that more blocks are already in memory rather having to be read from disk. The query will still need to read the same number of blocks so tuning is the first recommendation, but if you cannot tune the statement, a query reading blocks from memory is much faster than from disk.

4. Slow disks could be causing Oracle to spend time reading the data into the buffer cache. Review the 'DB Single Block Disk Read Time' metric in SolarWinds DPA to determine disk speeds from Oracle's perspective. If the time to read data is above 20ms, that could indicate slow disks.

© 2015 SOLARWINDS WORLDWIDE, LLC. ALL RIGHTS RESERVED.

IDENTIFY END-TO-END TIME

Accurate End-to-End Response Time Analysis



© 2015 SOLARWINDS WORLDWIDE, LLC. ALL RIGHTS RESERVED.

2. GET EXECUTION PLAN

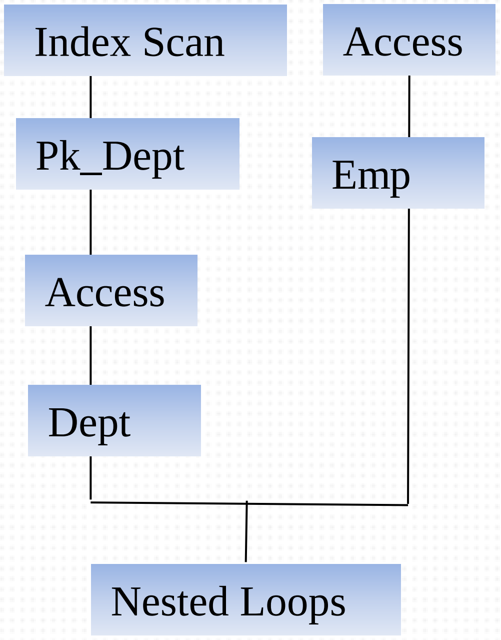
- **First, what is an execution plan?**
- Shows the **sequence of operations** performed to run SQL Statement
  - Order of the tables referenced in the statements
  - Access method for each table in the statement
    - INDEX
    - INLIST ITERATOR
    - TABLE ACCESS
    - VIEW
  - Join method in statement accessing multiple tables
    - HASH JOIN
    - MERGE JOIN
    - NESTED LOOPS
  - Data manipulations
    - CONCATENATION
    - COUNT
    - FILTER
    - SORT
  - Statistic Collectors
    - New in 12C

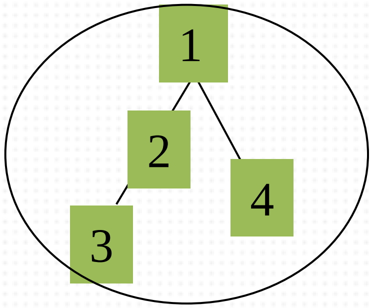
© 2015 SOLARWINDS WORLDWIDE, LLC. ALL RIGHTS RESERVED.

SIMPLE EXAMPLE

- Optimizer’s detailed steps to execute a SQL Statement

```
SELECT e.empno EID, e.ename "Employee_name",
            d.dname "Department“, e.hiredate "Hired_Date"
FROM  emp e,  dept d
WHERE  d.deptno = '40'
AND e.deptno = d.deptno;


------------------------------------------------------------
 Id  | Operation & Option                            | Name
------------------------------------------------------------
   0 | SELECT STATEMENT                              |
   1 |   NESTED LOOPS                                |
   2 |     TABLE ACCESS BY INDEX ROWID               | DEPT
   3 |        INDEX UNIQUE SCAN                      | PK_DEPT
   4 |     TABLE ACCESS FULL                         | EMP
```





© 2015 SOLARWINDS WORLDWIDE, LLC. ALL RIGHTS RESERVED.

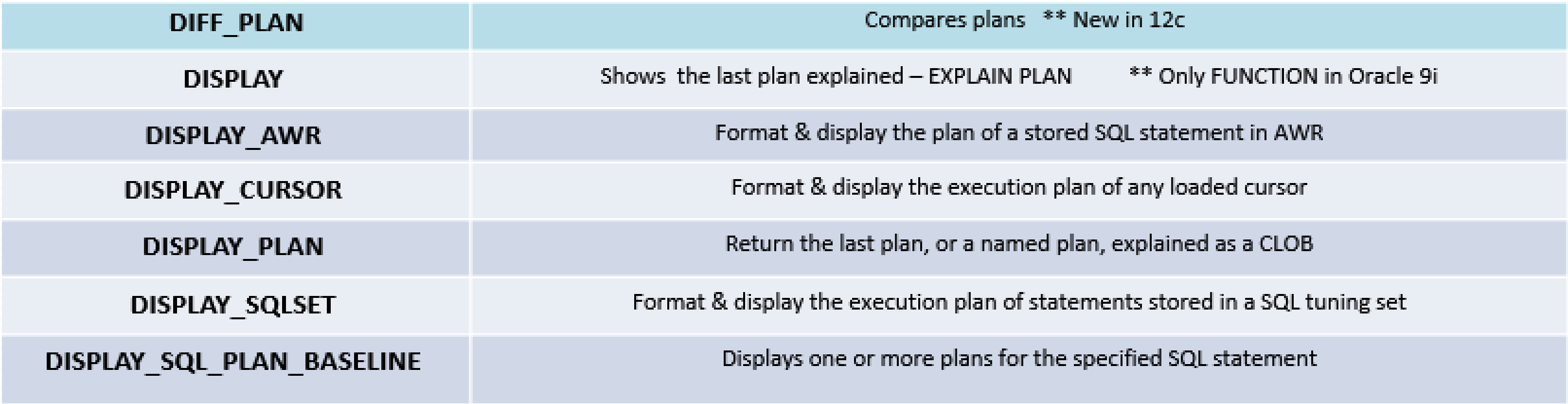
HOW TO VIEW THE EXECUTION PLAN

- » EXPLAIN PLAN
  - Estimated plan - can be wrong for many reasons
    - Best Guess, Blind to Bind Variables or Data types
    - Explain Plan For … sql statement & DBMS_XPLAN.display
    - Set autotrace (on | trace | exp | stat | off)
- » Tracing (all versions) / TKPROF
  - Get all sorts of good information
  - Works when you know a problem will occur
- » V$SQL_PLAN (Oracle 9i+)
  - Actual execution plan
  - Use DBMS_XPLAN.display_cursor for display
- » Historical Plans – AWR, Solarwinds DPA
  - Shows plan changes over time

© 2015 SOLARWINDS WORLDWIDE, LLC. ALL RIGHTS RESERVED.

DBMS_XPLAN

» Functions in 12c



DIFF_PLAN	Compares plans ** New in 12c
DISPLAY	Shows the last plan explained – EXPLAIN PLAN ** Only FUNCTION in Oracle 9i
DISPLAY_AWR	Format & display the plan of a stored SQL statement in AWR
DISPLAY_CURSOR	Format & display the execution plan of any loaded cursor
DISPLAY_PLAN	Return the last plan, or a named plan, explained as a CLOB
DISPLAY_SQLSET	Format & display the execution plan of statements stored in a SQL tuning set
DISPLAY_SQL_PLAN_BASELINE	Displays one or more plans for the specified SQL statement

» New format options for display_cursor

```
select * from table (dbms_xplan.display_cursor(&sql_id,&child,format=>'+adaptive'))
```

» Shorthand to get last statement run

```
select * from table(dbms_xplan.display_cursor(format =>'+report +adaptive'))
```

© 2015 SOLARWINDS WORLDWIDE, LLC. ALL RIGHTS RESERVED.

3. EXAMINE THE EXECUTION PLAN

» Find Expensive Operators

- Examine cost, row counts and time of each step
- Look for full table or index scans (expensive steps)

» Review the Predicate Information

- Know how bind variables are being interpreted
    - Review the data types
    - Implicit conversions
- Know which step filtering predicate is applied

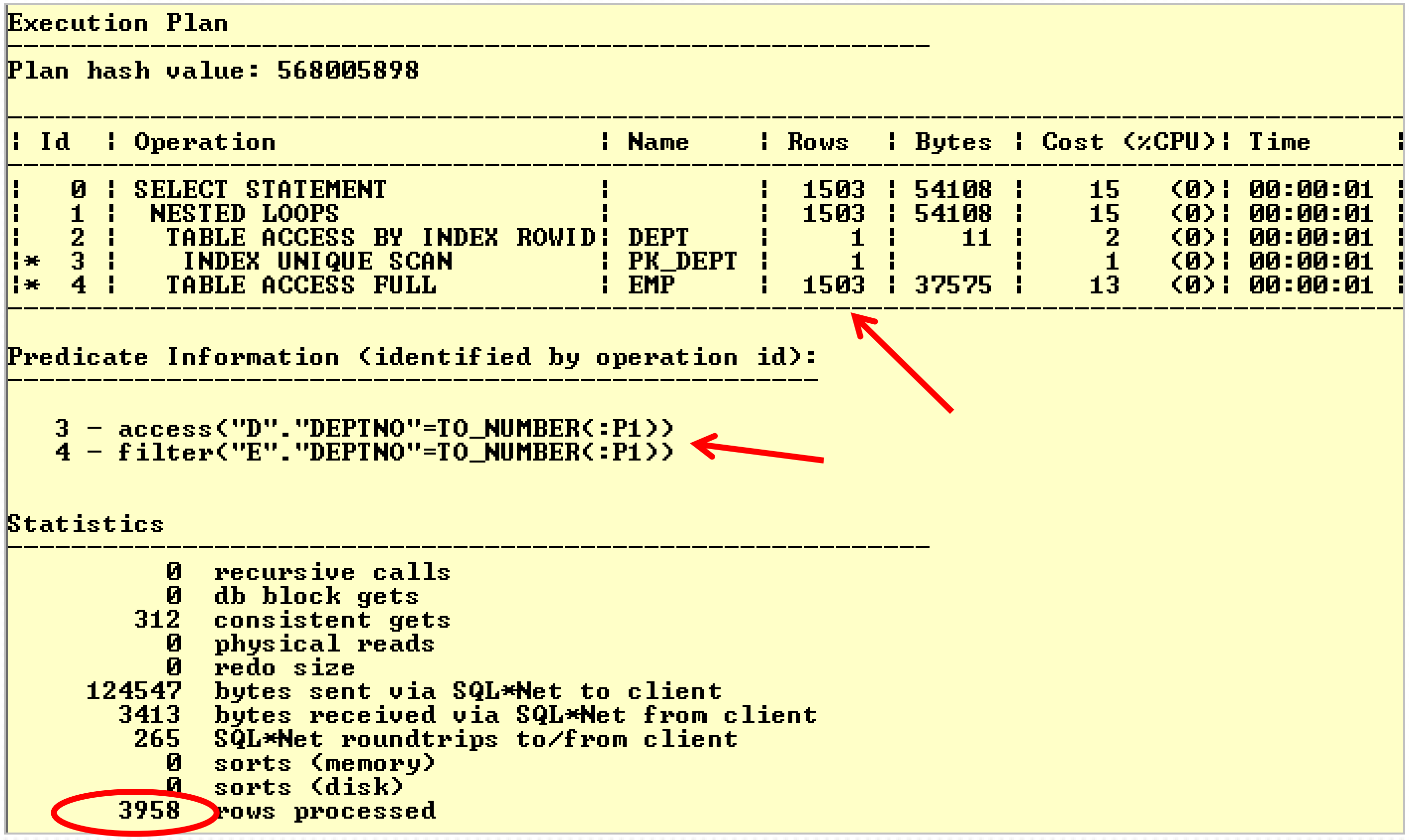
» Check out the Notes Section

© 2015 SOLARWINDS WORLDWIDE, LLC. ALL RIGHTS RESERVED.

EXECUTION PLAN DETAILS (EXPLAIN PLAN)

SELECT e.empno EID, e.ename "Employee_name”, d.dname "Department“, e.hiredate "Date_Hired"
FROM emp e, dept d WHERE d.deptno = :P1 AND e.deptno = d.deptno;

SET AUTOTRACE TRACEONLY:

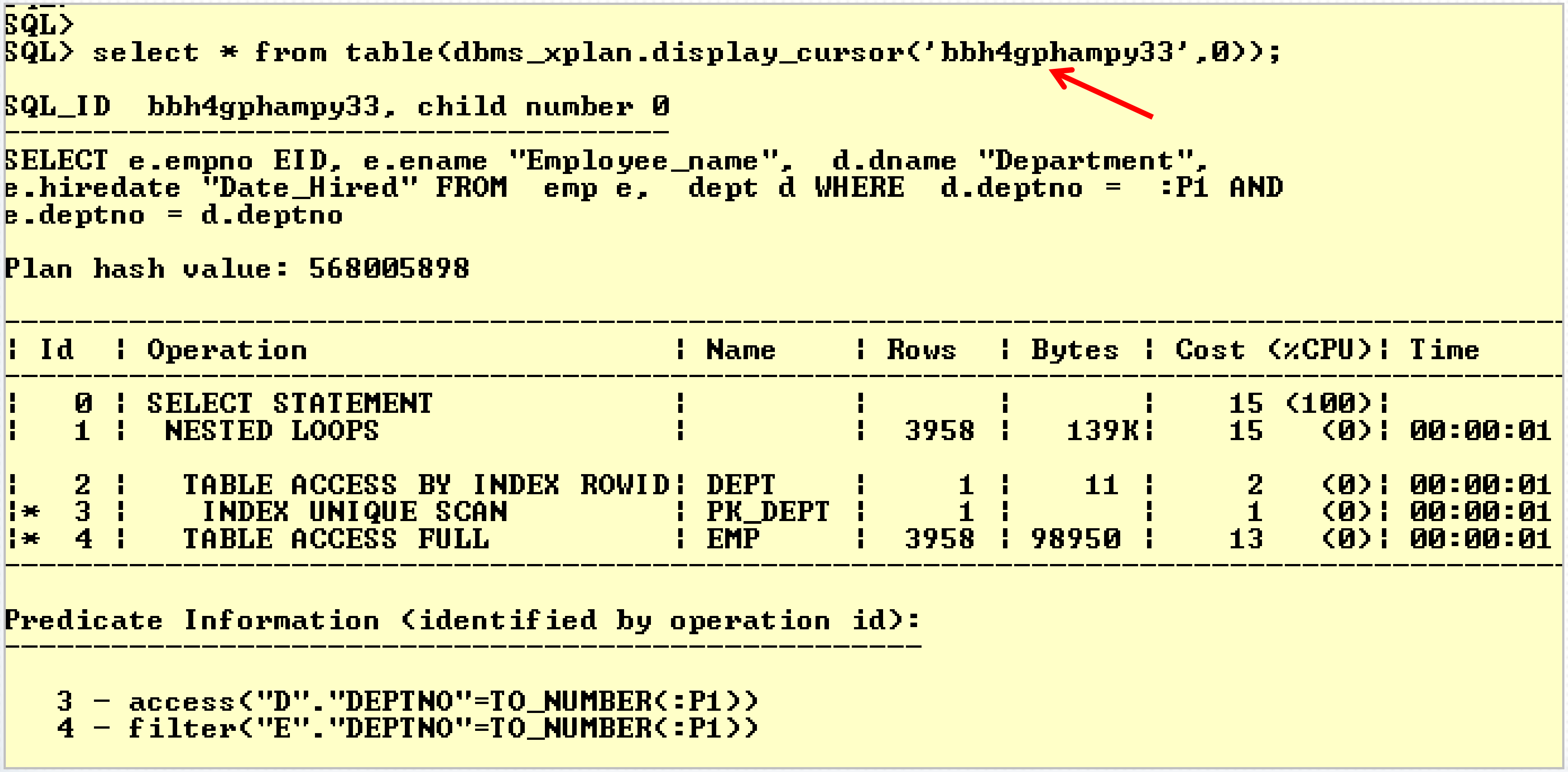


```
Execution Plan
----------------------------------------------------------
Plan hash value: 568005898

------------------------------------------------------------------------------------------
| Id  | Operation                    | Name    | Rows  | Bytes | Cost (%CPU)| Time     |
------------------------------------------------------------------------------------------
|   0 | SELECT STATEMENT             |         |  1503 | 54108 |    15   (0)| 00:00:01 |
|   1 |  NESTED LOOPS                |         |  1503 | 54108 |    15   (0)| 00:00:01 |
|   2 |   TABLE ACCESS BY INDEX ROWID| DEPT    |     1 |    11 |     2   (0)| 00:00:01 |
|*  3 |    INDEX UNIQUE SCAN         | PK_DEPT |     1 |       |     1   (0)| 00:00:01 |
|*  4 |   TABLE ACCESS FULL          | EMP     |  1503 | 37575 |    13   (0)| 00:00:01 |
------------------------------------------------------------------------------------------

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

   3 - access("D"."DEPTNO"=TO_NUMBER(:P1))
   4 - filter("E"."DEPTNO"=TO_NUMBER(:P1))


Statistics
----------------------------------------------------------
          0  recursive calls
          0  db block gets
        312  consistent gets
          0  physical reads
          0  redo size
     124547  bytes sent via SQL*Net to client
       3413  bytes received via SQL*Net from client
        265  SQL*Net roundtrips to/from client
          0  sorts (memory)
          0  sorts (disk)
       3958  rows processed
```

EXECUTION PLAN DETAILS (ACTUAL)

SELECT e.empno EID, e.ename "Employee_name",
            d.dname "Department", e.hiredate "Date_Hired"
FROM emp e, dept d WHERE d.deptno = :P1 AND e.deptno = d.deptno;

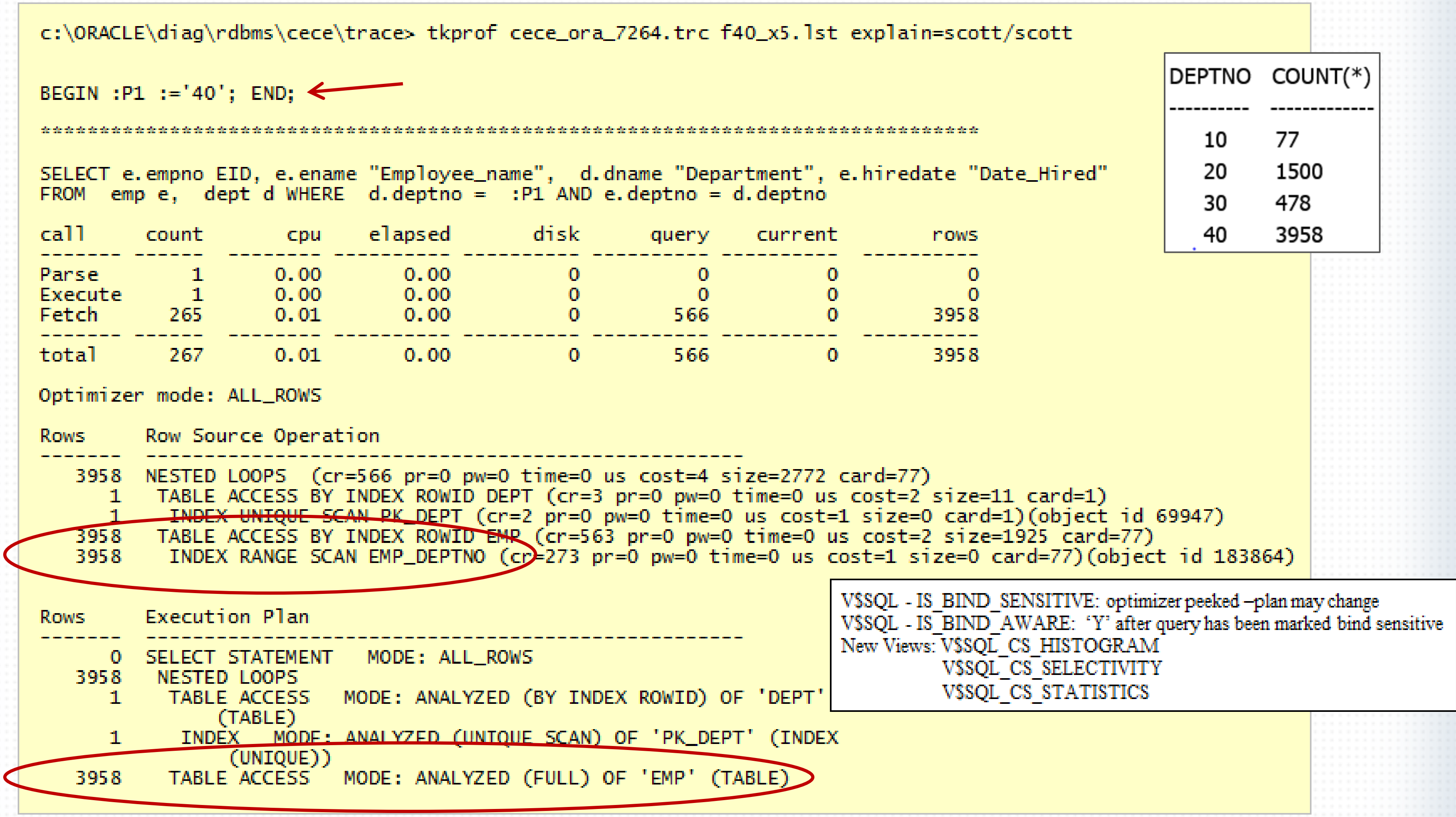
Actual Plan: V$SQL_PLAN using dbms_xplan.display_cursor



```
SQL>
SQL> select * from table(dbms_xplan.display_cursor('bbh4gphampy33',0));

SQL_ID  bbh4gphampy33, child number 0
-------------------------------------
SELECT e.empno EID, e.ename "Employee_name",  d.dname "Department",
e.hiredate "Date_Hired" FROM  emp e,  dept d WHERE  d.deptno =  :P1 AND
e.deptno = d.deptno

Plan hash value: 568005898

--------------------------------------------------------------------------------------
| Id  | Operation                    | Name    | Rows  | Bytes | Cost (%CPU)| Time
--------------------------------------------------------------------------------------
|   0 | SELECT STATEMENT             |         |       |       |    15 (100)|
|   1 |  NESTED LOOPS                |         |  3958 |   139K|    15   (0)| 00:00:01

|   2 |   TABLE ACCESS BY INDEX ROWID| DEPT    |     1 |    11 |     2   (0)| 00:00:01
|*  3 |    INDEX UNIQUE SCAN         | PK_DEPT |     1 |       |     1   (0)| 00:00:01
|*  4 |   TABLE ACCESS FULL          | EMP     |  3958 | 98950 |    13   (0)| 00:00:01
--------------------------------------------------------------------------------------

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

   3 - access("D"."DEPTNO"=TO_NUMBER(:P1))
   4 - filter("E"."DEPTNO"=TO_NUMBER(:P1))
```

EXECUTION – ACTUAL VS EXPLAIN PLAN

Bind Variable Peeking Example / Adaptive Cursor Sharing Fix (11g)



```
c:\ORACLE\diag\rdbms\cece\trace> tkprof cece_ora_7264.trc f40_x5.lst explain=scott/scott

BEGIN :P1 :='40'; END;

********************************************************************************

SELECT e.empno EID, e.ename "Employee_name",  d.dname "Department", e.hiredate "Date_Hired"
FROM  emp e,  dept d WHERE  d.deptno =  :P1 AND e.deptno = d.deptno

call     count       cpu    elapsed       disk      query    current        rows
------- ------  -------- ---------- ---------- ---------- ----------  ----------
Parse        1      0.00       0.00          0          0          0           0
Execute      1      0.00       0.00          0          0          0           0
Fetch      265      0.01       0.00          0        566          0        3958
------- ------  -------- ---------- ---------- ---------- ----------  ----------
total      267      0.01       0.00          0        566          0        3958

Optimizer mode: ALL_ROWS

Rows     Row Source Operation
-------  ---------------------------------------------------
   3958  NESTED LOOPS  (cr=566 pr=0 pw=0 time=0 us cost=4 size=2772 card=77)
      1   TABLE ACCESS BY INDEX ROWID DEPT (cr=3 pr=0 pw=0 time=0 us cost=2 size=11 card=1)
      1    INDEX UNIQUE SCAN PK_DEPT (cr=2 pr=0 pw=0 time=0 us cost=1 size=0 card=1)(object id 69947)
   3958   TABLE ACCESS BY INDEX ROWID EMP (cr=563 pr=0 pw=0 time=0 us cost=2 size=1925 card=77)
   3958    INDEX RANGE SCAN EMP_DEPTNO (cr=273 pr=0 pw=0 time=0 us cost=1 size=0 card=77)(object id 183864)

Rows     Execution Plan
-------  ---------------------------------------------------
      0  SELECT STATEMENT   MODE: ALL_ROWS
   3958   NESTED LOOPS
      1    TABLE ACCESS   MODE: ANALYZED (BY INDEX ROWID) OF 'DEPT'
               (TABLE)
      1     INDEX   MODE: ANALYZED (UNIQUE SCAN) OF 'PK_DEPT' (INDEX
                (UNIQUE))
   3958    TABLE ACCESS   MODE: ANALYZED (FULL) OF 'EMP' (TABLE)
```

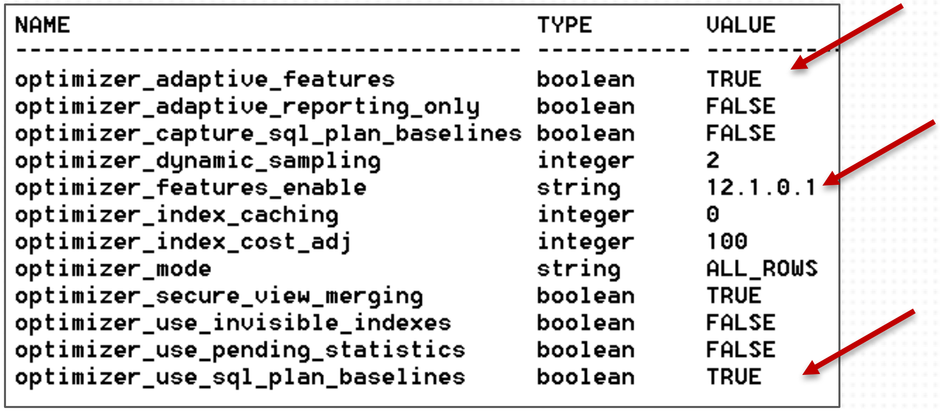
DEPTNO	COUNT(*)
10	77
20	1500
30	478
40	3958

V$SQL - IS_BIND_SENSITIVE: optimizer peeked –plan may change
V$SQL - IS_BIND_AWARE: 'Y' after query has been marked bind sensitive
New Views: V$SQL_CS_HISTOGRAM
V$SQL_CS_SELECTIVITY
V$SQL_CS_STATISTICS

© 2015 SOLARWINDS WORLDWIDE, LLC. ALL RIGHTS RESERVED.

4. KNOW THE OPTIMIZER FEATURES USED

» Show parameter optimizer

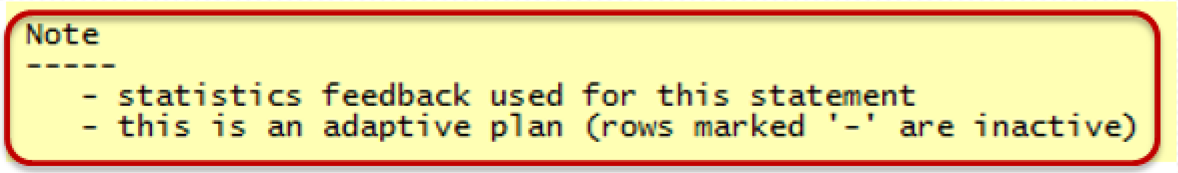


```
NAME                                 TYPE        VALUE
------------------------------------ ----------- ---------
optimizer_adaptive_features          boolean     TRUE
optimizer_adaptive_reporting_only    boolean     FALSE
optimizer_capture_sql_plan_baselines boolean     FALSE
optimizer_dynamic_sampling           integer     2
optimizer_features_enable            string      12.1.0.1
optimizer_index_caching              integer     0
optimizer_index_cost_adj             integer     100
optimizer_mode                       string      ALL_ROWS
optimizer_secure_view_merging        boolean     TRUE
optimizer_use_invisible_indexes      boolean     FALSE
optimizer_use_pending_statistics     boolean     FALSE
optimizer_use_sql_plan_baselines     boolean     TRUE
```

» What is supporting the Execution Plan

- SQL Plan Management (Baselines) / Profiles
- Dynamic Statistics or SQL Directives
- Adaptive Cursor Sharing
- Adaptive Plans

» Notes Section gives you clues



```
Note
-----
   - statistics feedback used for this statement
   - this is an adaptive plan (rows marked '-' are inactive)
```

© 2015 SOLARWINDS WORLDWIDE, LLC. ALL RIGHTS RESERVED.

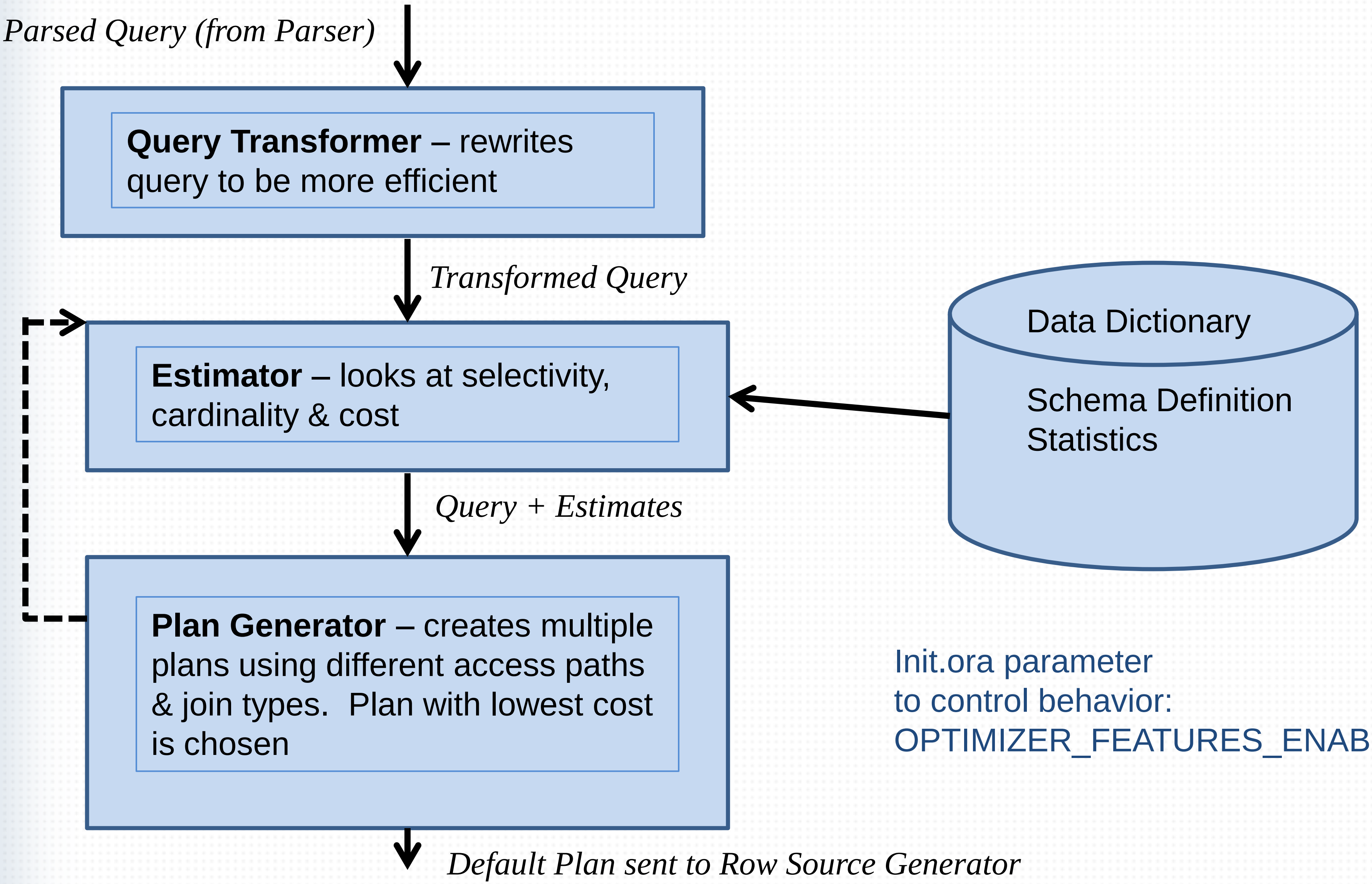
IN THE BEGINNING…

- Rule Based Optimizer (Version < = 6)
  - Rules based on 17 possible access paths
  - Only one Execution Plan chosen based on ranking of rules
  - Tricks were used to change the Optimizer's behavior
  - Simple rewrites of 'OR' to 'Union ALL'

- Cost Based Optimizer (Version > = 7.3)
  - Multiple plans generated with estimated cost of IO/CPU
    - Plan with lowest cost chosen
  - Allowed for Hash joins, Histograms, Partitioning & Parallel queries
  - More complex rewrites / transformations
  - Required statistics gathering / Plans Changed
    - 8.1.7, Stored Outlines to control plan changes
    - 9.2, Dynamic sampling of Statistics
    - 10g, SQL Profiles / Tuning Advisor
      - DBMS_SQLTUNE – Costs $$$
    - Oracle 11, Adaptive Cursor Sharing / SQL Plan Management
    - Oracle 12c, Adaptive Optimizer

© 2015 SOLARWINDS WORLDWIDE, LLC. ALL RIGHTS RESERVED.

HOW THE OPTIMIZER WORKS



Parsed Query (from Parser)

Query Transformer – rewrites query to be more efficient

Transformed Query

Estimator – looks at selectivity, cardinality & cost

Data Dictionary

Schema Definition
Statistics

Query + Estimates

Plan Generator – creates multiple plans using different access paths & join types. Plan with lowest cost is chosen

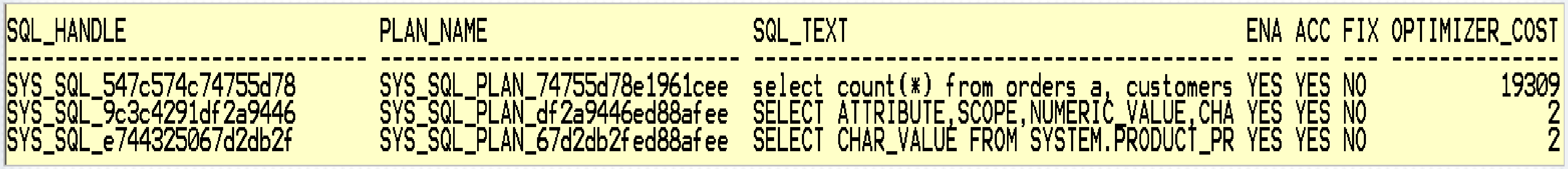
Default Plan sent to Row Source Generator

Init.ora parameter
to control behavior:
OPTIMIZER_FEATURES_ENABLED

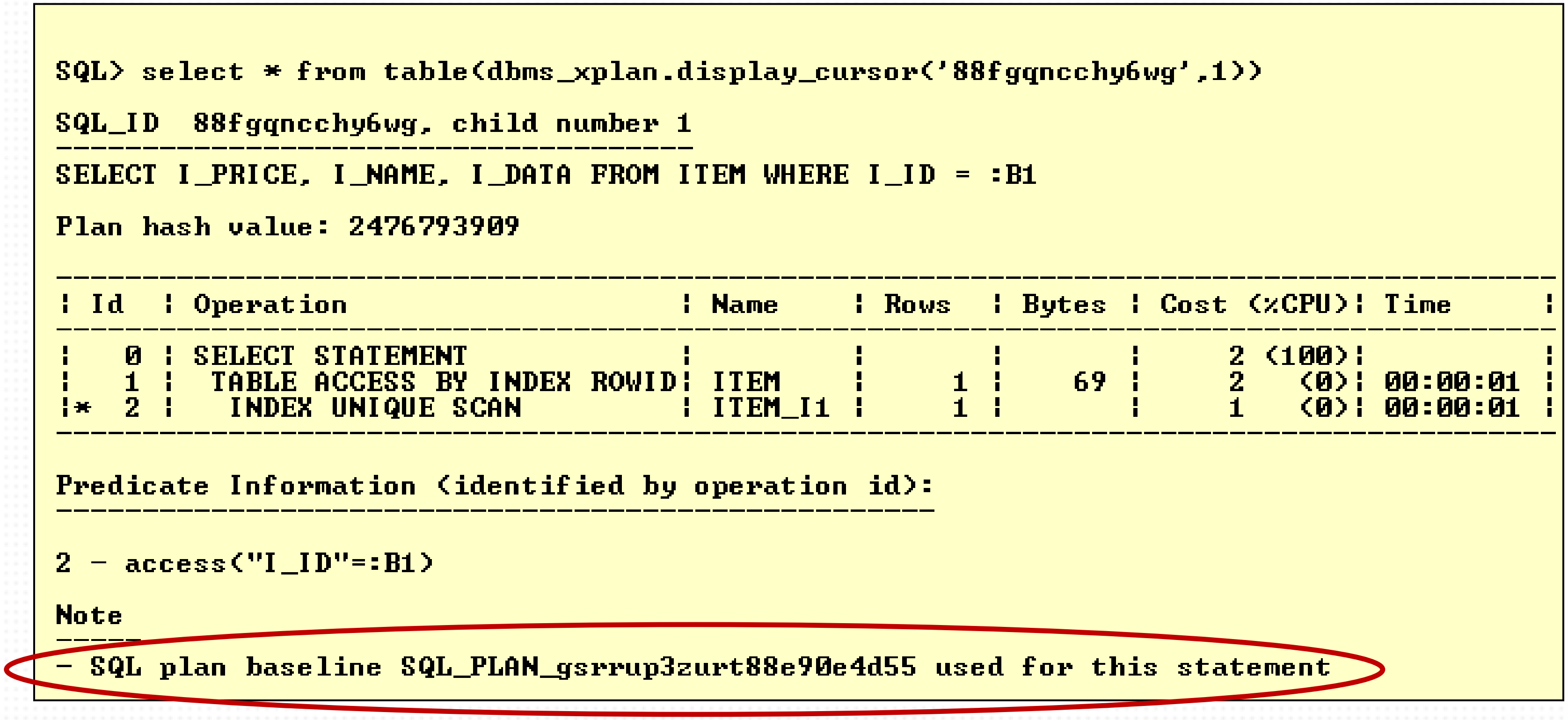
© 2015 SOLARWINDS WORLDWIDE, LLC. ALL RIGHTS RESERVED.

EXECUTION PLAN USING SPM (11G)

Select * from dba_sql_plans_baselines;



```
SQL_HANDLE                     PLAN_NAME                      SQL_TEXT                                 ENA ACC FIX OPTIMIZER_COST
------------------------------ ------------------------------ ---------------------------------------- --- --- --- --------------
SYS_SQL_547c574c74755d78       SYS_SQL_PLAN_74755d78e1961cee  select count(*) from orders a, customers YES YES NO           19309
SYS_SQL_9c3c4291df2a9446       SYS_SQL_PLAN_df2a9446ed88afee  SELECT ATTRIBUTE,SCOPE,NUMERIC_VALUE,CHA YES YES NO               2
SYS_SQL_e744325067d2db2f       SYS_SQL_PLAN_67d2db2fed88afee  SELECT CHAR_VALUE FROM SYSTEM.PRODUCT_PR YES YES NO               2
```

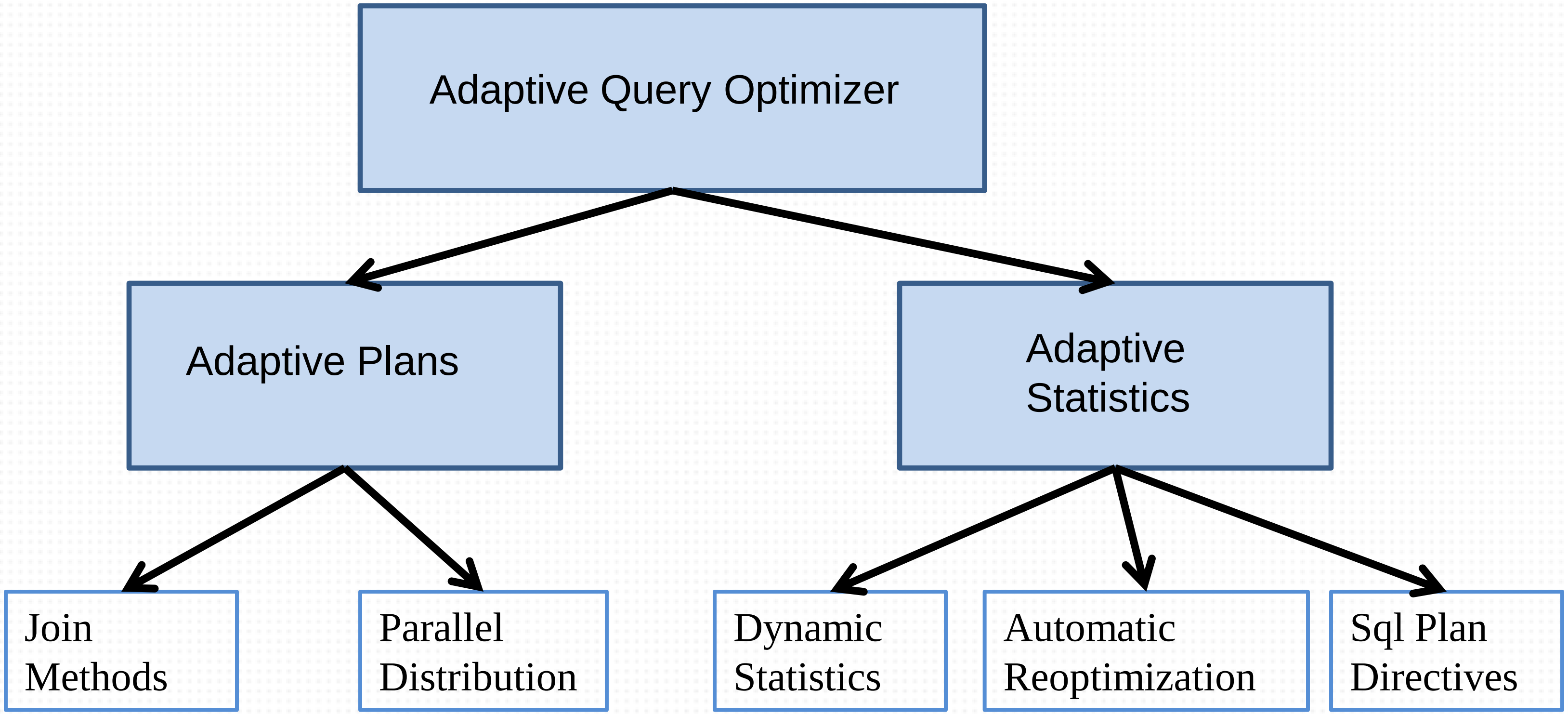


```
SQL> select * from table(dbms_xplan.display_cursor(’88fgqncchy6wg’,1))

SQL_ID  88fgqncchy6wg, child number 1
-------------------------------------
SELECT I_PRICE, I_NAME, I_DATA FROM ITEM WHERE I_ID = :B1

Plan hash value: 2476793909

---------------------------------------------------------------------------------------
| Id  | Operation                   | Name    | Rows  | Bytes | Cost (%CPU)| Time     |
---------------------------------------------------------------------------------------
|   0 | SELECT STATEMENT            |         |       |       |     2 (100)|          |
|   1 |  TABLE ACCESS BY INDEX ROWID| ITEM    |     1 |    69 |     2   (0)| 00:00:01 |
|*  2 |   INDEX UNIQUE SCAN         | ITEM_I1 |     1 |       |     1   (0)| 00:00:01 |
---------------------------------------------------------------------------------------

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

2 - access("I_ID"=:B1)

Note
-----
- SQL plan baseline SQL_PLAN_gsrrup3zurt88e90e4d55 used for this statement
```

© 2015 SOLARWINDS WORLDWIDE, LLC. ALL RIGHTS RESERVED.

12C ADAPTIVE QUERY OPTIMIZER

- Allows for run-time adjustments to execution plans
- Can discover additional information
  - which can lead to better statistics & optimal plans



© 2015 SOLARWINDS WORLDWIDE, LLC. ALL RIGHTS RESERVED.

HOW TO VIEW ADAPTATIONS

- Use DBMS_XPLAN.DISPLAY_CURSOR
  - Explain Plan (dbms_xplan.display) may only show default or initial plan
    - Be Careful!
  - Use format parameter ‘+report’ for testing
    - Shows what the adaptive plan would be but doesn’t use it

```
select * from table(dbms_xplan.display_cursor
('&sql_id',&child,format=>'+report'));
```

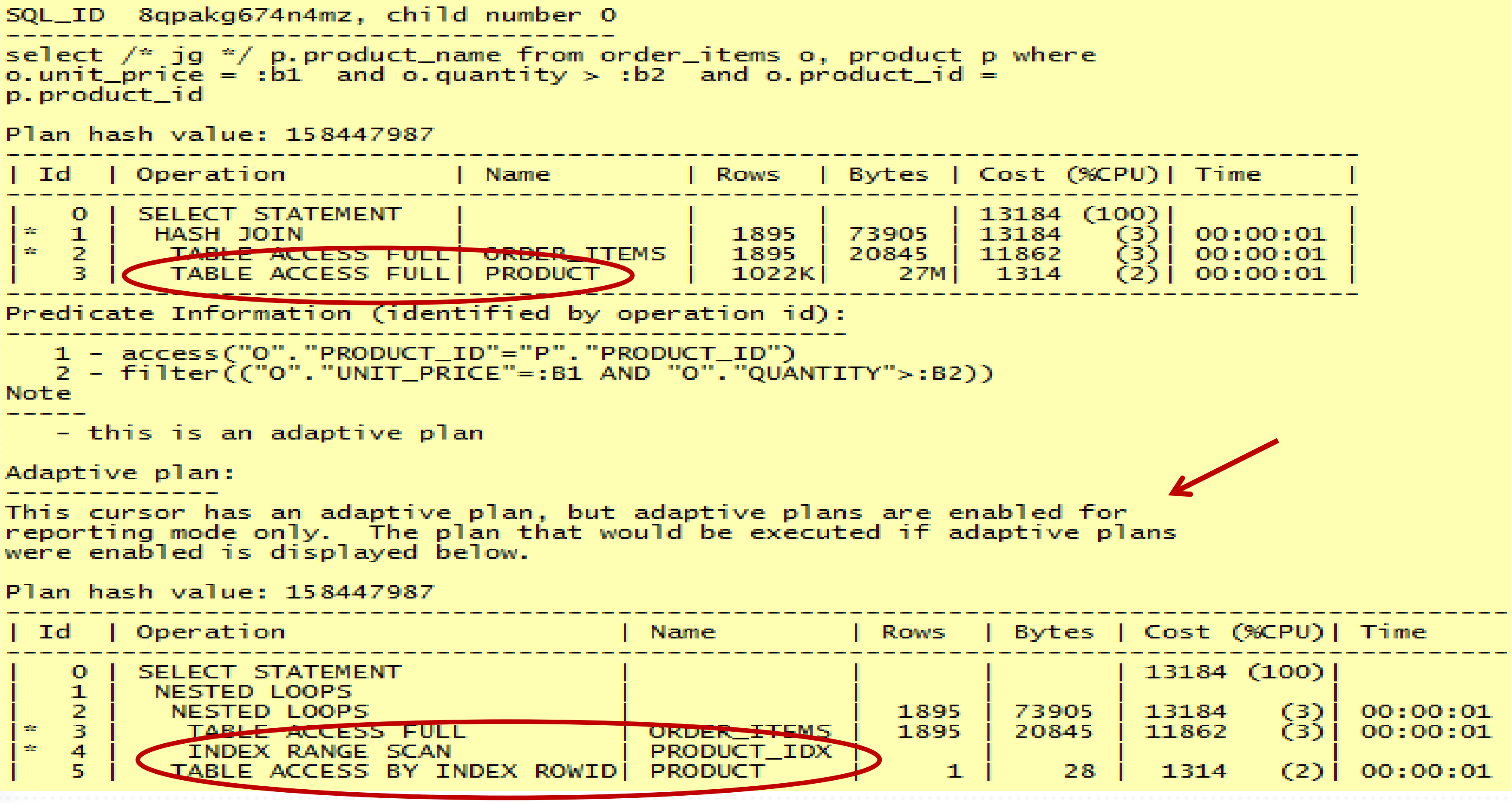
  - Use format parameter ‘+adaptive’ to see all steps (active / inactive)
    - including optimizer statistics collectors

```
select * from
table(dbms_xplan.display_cursor('&sql_id',&child,format=>'+adaptive'));
```

© 2015 SOLARWINDS WORLDWIDE, LLC. ALL RIGHTS RESERVED.

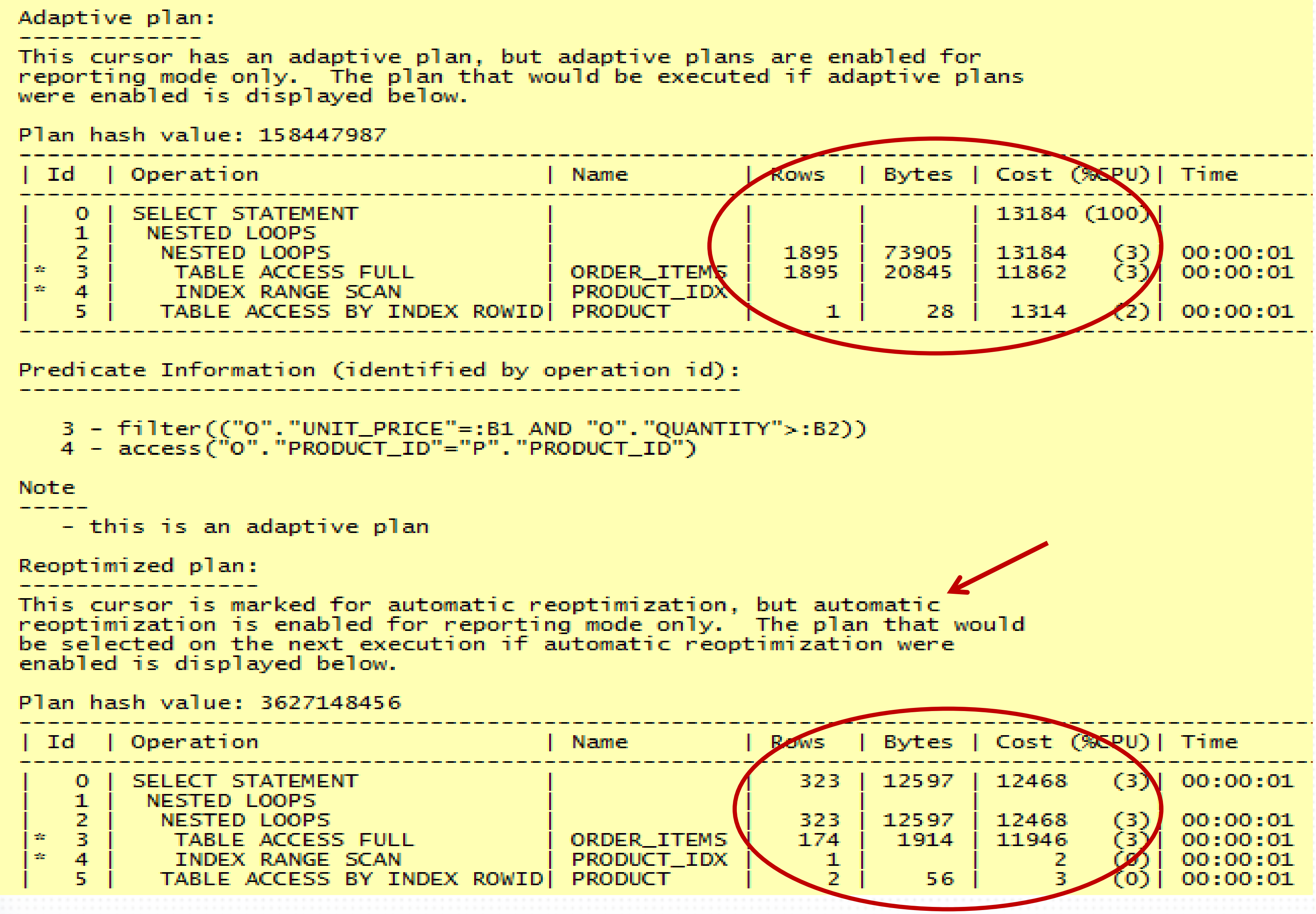
REPORT MODE

alter session set optimizer_adaptive_reporting_only=TRUE;
select * from table(dbms_xplan.display_cursor('8qpakg674n4mz',0,format=>'+report'));



```
SQL_ID  8qpakg674n4mz, child number 0
-------------------------------------
select /* jg */ p.product_name from order_items o, product p where
o.unit_price = :b1  and o.quantity > :b2  and o.product_id =
p.product_id

Plan hash value: 158447987

-----------------------------------------------------------------------------------
| Id  | Operation           | Name          | Rows  | Bytes | Cost (%CPU)| Time     |
-----------------------------------------------------------------------------------
|   0 | SELECT STATEMENT    |               |       |       | 13184 (100)|          |
|*  1 |  HASH JOIN          |               |  1895 | 73905 | 13184   (3)| 00:00:01 |
|*  2 |   TABLE ACCESS FULL| ORDER_ITEMS   |  1895 | 20845 | 11862   (3)| 00:00:01 |
|   3 |   TABLE ACCESS FULL| PRODUCT       |  1022K|   27M|  1314   (2)| 00:00:01 |
-----------------------------------------------------------------------------------
Predicate Information (identified by operation id):
---------------------------------------------------
   1 - access("O"."PRODUCT_ID"="P"."PRODUCT_ID")
   2 - filter(("O"."UNIT_PRICE"=:B1 AND "O"."QUANTITY">:B2))
Note
-----
   - this is an adaptive plan

Adaptive plan:
-------------
This cursor has an adaptive plan, but adaptive plans are enabled for
reporting mode only.  The plan that would be executed if adaptive plans
were enabled is displayed below.

Plan hash value: 158447987

-------------------------------------------------------------------------------------------
| Id  | Operation                    | Name          | Rows  | Bytes | Cost (%CPU)| Time     
-------------------------------------------------------------------------------------------
|   0 | SELECT STATEMENT             |               |       |       | 13184 (100)|
|   1 |  NESTED LOOPS                |               |       |       |            |
|   2 |   NESTED LOOPS               |               |  1895 | 73905 | 13184   (3)| 00:00:01
|*  3 |    TABLE ACCESS FULL         | ORDER_ITEMS   |  1895 | 20845 | 11862   (3)| 00:00:01
|*  4 |    INDEX RANGE SCAN          | PRODUCT_IDX   |       |       |            |
|   5 |   TABLE ACCESS BY INDEX ROWID| PRODUCT       |     1 |    28 |  1314   (2)| 00:00:01
```

REPORT MODE – CONT.



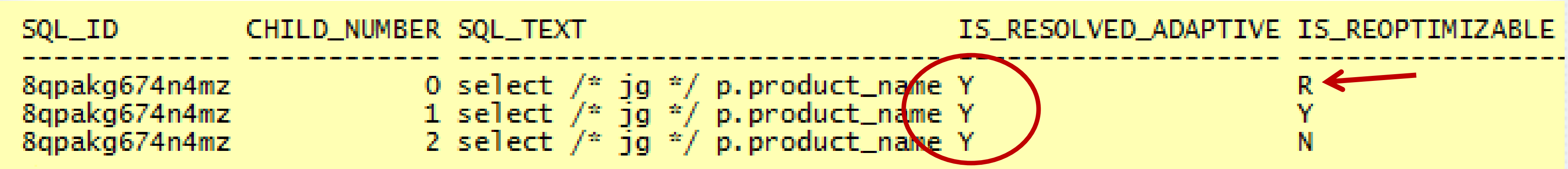
```
Adaptive plan:
-------------
This cursor has an adaptive plan, but adaptive plans are enabled for
reporting mode only.  The plan that would be executed if adaptive plans
were enabled is displayed below.

Plan hash value: 158447987

-------------------------------------------------------------------------------------------
| Id  | Operation                     | Name          | Rows  | Bytes | Cost (%CPU)| Time     
-------------------------------------------------------------------------------------------
|   0 | SELECT STATEMENT              |               |       |       | 13184 (100)|          
|   1 |  NESTED LOOPS                 |               |       |       |            |          
|   2 |   NESTED LOOPS                |               |  1895 | 73905 | 13184   (3)| 00:00:01 
|*  3 |    TABLE ACCESS FULL          | ORDER_ITEMS   |  1895 | 20845 | 11862   (3)| 00:00:01 
|*  4 |    INDEX RANGE SCAN           | PRODUCT_IDX   |       |       |            |          
|   5 |   TABLE ACCESS BY INDEX ROWID| PRODUCT       |     1 |    28 |  1314   (2)| 00:00:01 
-------------------------------------------------------------------------------------------

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

   3 - filter(("O"."UNIT_PRICE"=:B1 AND "O"."QUANTITY">:B2))
   4 - access("O"."PRODUCT_ID"="P"."PRODUCT_ID")

Note
-----
   - this is an adaptive plan

Reoptimized plan:
-----------------
This cursor is marked for automatic reoptimization, but automatic
reoptimization is enabled for reporting mode only.  The plan that would
be selected on the next execution if automatic reoptimization were
enabled is displayed below.

Plan hash value: 3627148456

-------------------------------------------------------------------------------------------
| Id  | Operation                     | Name          | Rows  | Bytes | Cost (%CPU)| Time     
-------------------------------------------------------------------------------------------
|   0 | SELECT STATEMENT              |               |   323 | 12597 | 12468   (3)| 00:00:01 
|   1 |  NESTED LOOPS                 |               |       |       |            |          
|   2 |   NESTED LOOPS                |               |   323 | 12597 | 12468   (3)| 00:00:01 
|*  3 |    TABLE ACCESS FULL          | ORDER_ITEMS   |   174 |  1914 | 11946   (3)| 00:00:01 
|*  4 |    INDEX RANGE SCAN           | PRODUCT_IDX   |     1 |       |     2   (0)| 00:00:01 
|   5 |   TABLE ACCESS BY INDEX ROWID| PRODUCT       |     2 |    56 |     3   (0)| 00:00:01 
```

© 2015 SOLARWINDS WORLDWIDE, LLC. ALL RIGHTS RESERVED.

ADAPTIVE PLANS (12C)

```
SELECT sql_id, child_number,
             SUBSTR(sql_text, 1,30) sql_text,
              IS_RESOLVED_ADAPTIVE_PLAN,
              IS_REOPTIMIZABLE
FROM v$sql
WHERE sql_text like 'select /* jg */%'
ORDER BY sql_id,child_number
```

```
select /* jg */ p.product_name
from order_items o, product p
where o.unit_price = :b1
and o.quantity > :b2
and o.product_id = p.product_id;
```



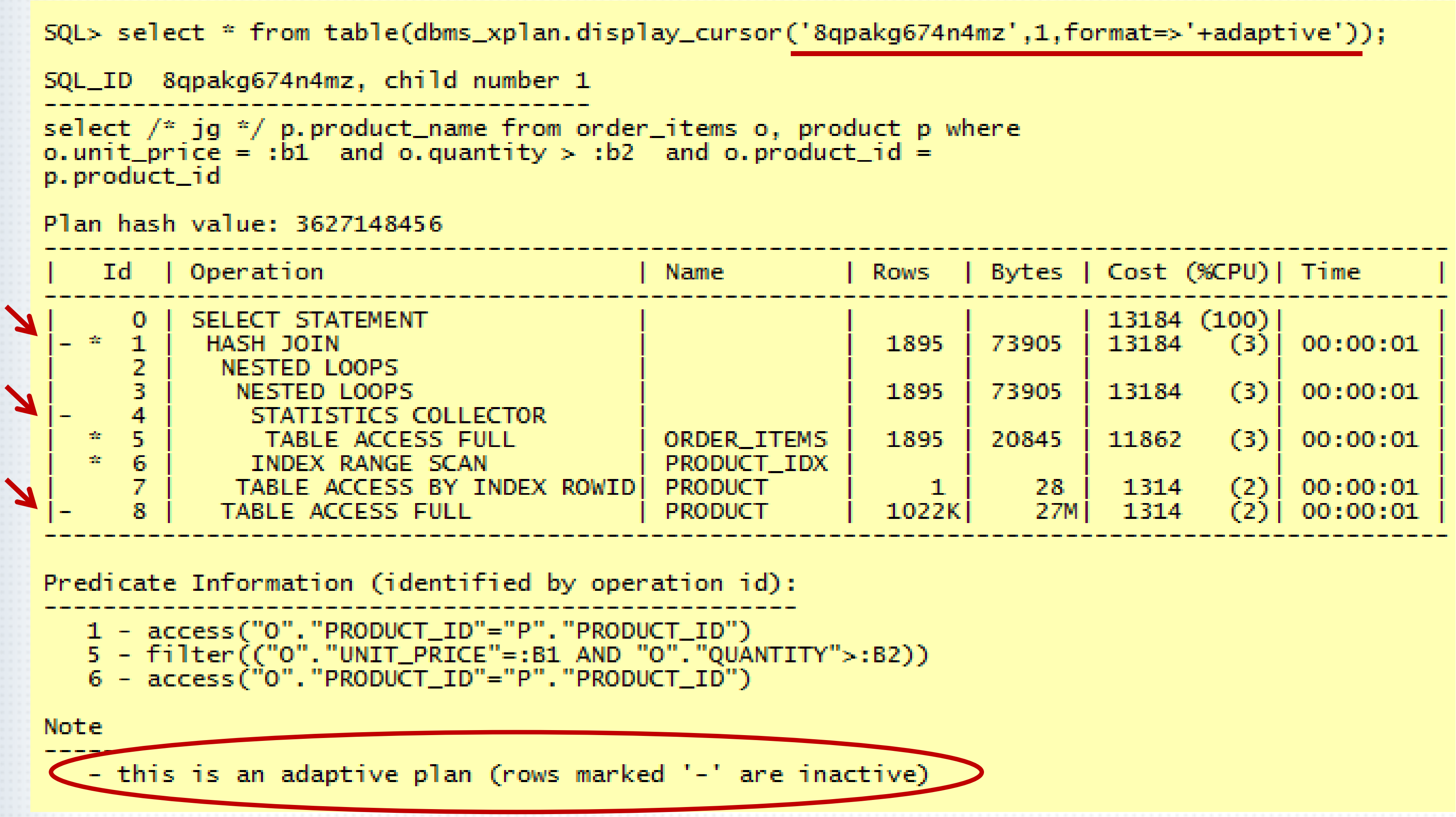
```
SQL_ID        CHILD_NUMBER SQL_TEXT                       IS_RESOLVED_ADAPTIVE IS_REOPTIMIZABLE
------------- ------------ ------------------------------ -------------------- ----------------
8qpakg674n4mz            0 select /* jg */ p.product_name Y                    R
8qpakg674n4mz            1 select /* jg */ p.product_name Y                    Y
8qpakg674n4mz            2 select /* jg */ p.product_name Y                    N
```

- IS_REOPTIMIZABLE is for next execution
  - Y - the next execution will trigger a reoptimization
  - R – has reoptimization info but won’t trigger due to reporting mode
  - N -the child cursor has no reoptimization info

© 2015 SOLARWINDS WORLDWIDE, LLC. ALL RIGHTS RESERVED.

ADAPTIVE PLAN EXAMPLE

Adapted on first execution
alter session set optimizer_adaptive_reporting_only=FALSE;

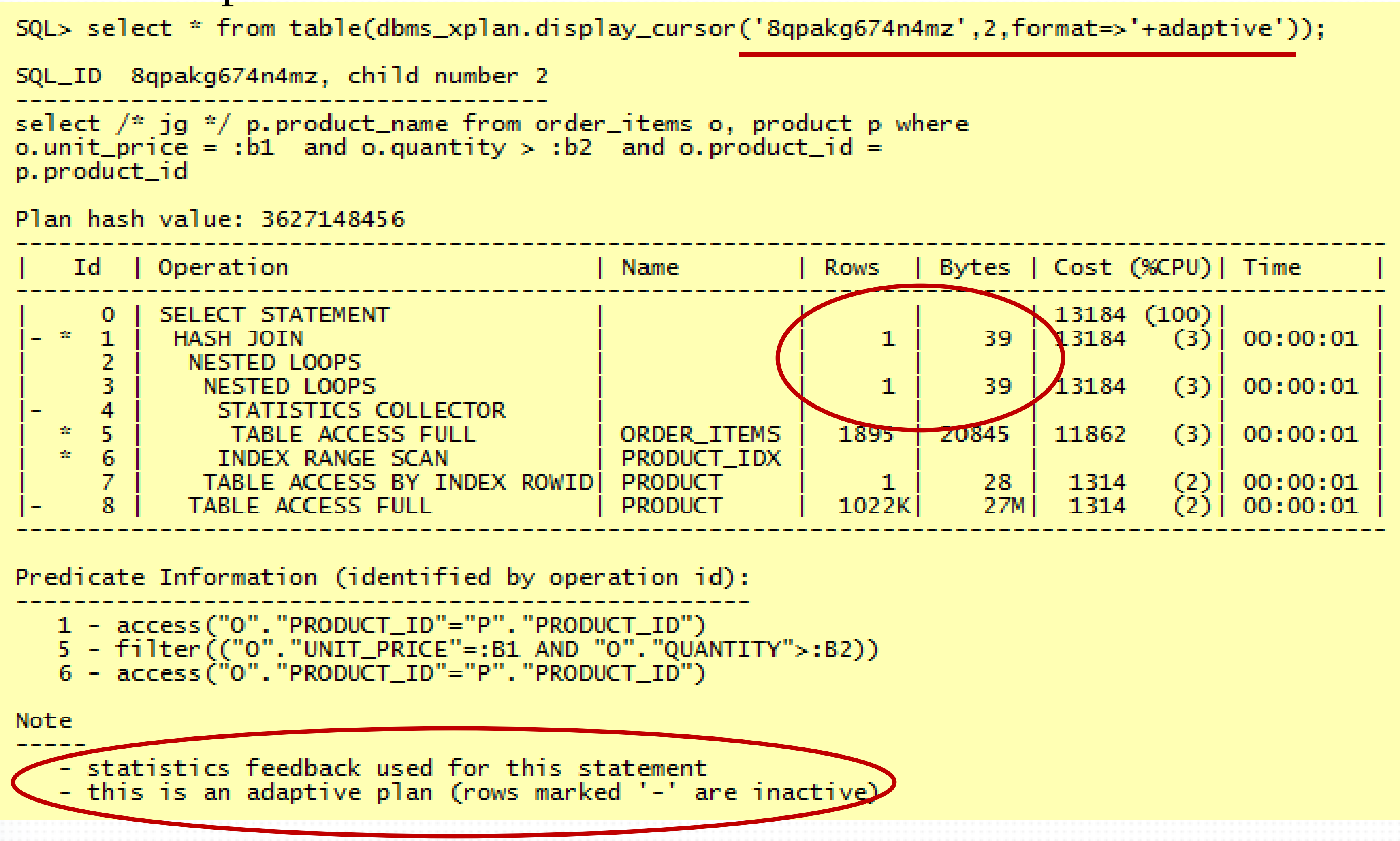


```
SQL> select * from table(dbms_xplan.display_cursor('8qpakg674n4mz',1,format=>'+adaptive'));

SQL_ID  8qpakg674n4mz, child number 1
-------------------------------------
select /* jg */ p.product_name from order_items o, product p where
o.unit_price = :b1  and o.quantity > :b2  and o.product_id =
p.product_id

Plan hash value: 3627148456

-----------------------------------------------------------------------------------------------
|   Id  | Operation                      | Name          | Rows  | Bytes | Cost (%CPU)| Time     |
-----------------------------------------------------------------------------------------------
|     0 | SELECT STATEMENT               |               |       |       | 13184 (100)|          |
|-  * 1 |  HASH JOIN                     |               |  1895 | 73905 | 13184   (3)| 00:00:01 |
|     2 |   NESTED LOOPS                 |               |       |       |            |          |
|     3 |    NESTED LOOPS                |               |  1895 | 73905 | 13184   (3)| 00:00:01 |
|-    4 |     STATISTICS COLLECTOR       |               |       |       |            |          |
|   * 5 |      TABLE ACCESS FULL         | ORDER_ITEMS   |  1895 | 20845 | 11862   (3)| 00:00:01 |
|   * 6 |     INDEX RANGE SCAN           | PRODUCT_IDX   |       |       |            |          |
|     7 |    TABLE ACCESS BY INDEX ROWID| PRODUCT       |     1 |    28 |  1314   (2)| 00:00:01 |
|-    8 |   TABLE ACCESS FULL            | PRODUCT       |  1022K|   27M|  1314   (2)| 00:00:01 |
-----------------------------------------------------------------------------------------------

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

   1 - access("O"."PRODUCT_ID"="P"."PRODUCT_ID")
   5 - filter(("O"."UNIT_PRICE"=:B1 AND "O"."QUANTITY">:B2))
   6 - access("O"."PRODUCT_ID"="P"."PRODUCT_ID")

Note
-----
   - this is an adaptive plan (rows marked '-' are inactive)
```

© 2015 SOLARWINDS WORLDWIDE, LLC. ALL RIGHTS RESERVED.

WHAT CHANGED?

After Reoptimization has occurred



```
SQL> select * from table(dbms_xplan.display_cursor('8qpakg674n4mz',2,format=>'+adaptive'));

SQL_ID  8qpakg674n4mz, child number 2
-------------------------------------
select /* jg */ p.product_name from order_items o, product p where
o.unit_price = :b1  and o.quantity > :b2  and o.product_id =
p.product_id

Plan hash value: 3627148456

-----------------------------------------------------------------------------------------------
|   Id  | Operation                      | Name         | Rows  | Bytes | Cost (%CPU)| Time     |
-----------------------------------------------------------------------------------------------
|     0 | SELECT STATEMENT               |              |       |       | 13184 (100)|          |
|-  * 1 |  HASH JOIN                     |              |     1 |    39 | 13184   (3)| 00:00:01 |
|     2 |   NESTED LOOPS                 |              |       |       |            |          |
|     3 |    NESTED LOOPS                |              |     1 |    39 | 13184   (3)| 00:00:01 |
|-    4 |     STATISTICS COLLECTOR       |              |       |       |            |          |
|   * 5 |      TABLE ACCESS FULL         | ORDER_ITEMS  |  1895 | 20845 | 11862   (3)| 00:00:01 |
|   * 6 |     INDEX RANGE SCAN           | PRODUCT_IDX  |       |       |            |          |
|     7 |    TABLE ACCESS BY INDEX ROWID| PRODUCT      |     1 |    28 |  1314   (2)| 00:00:01 |
|-    8 |   TABLE ACCESS FULL            | PRODUCT      |  1022K|   27M|  1314   (2)| 00:00:01 |
-----------------------------------------------------------------------------------------------

Predicate Information (identified by operation id):
---------------------------------------------------
   1 - access("O"."PRODUCT_ID"="P"."PRODUCT_ID")
   5 - filter(("O"."UNIT_PRICE"=:B1 AND "O"."QUANTITY">:B2))
   6 - access("O"."PRODUCT_ID"="P"."PRODUCT_ID")

Note
-----
   - statistics feedback used for this statement
   - this is an adaptive plan (rows marked '-' are inactive)
```

© 2015 SOLARWINDS WORLDWIDE, LLC. ALL RIGHTS RESERVED.

5. GET TABLE & COLUMN INFO

- Understand objects in execution plans
  - Table Definitions & Segment sizes
    - Is it a View – get underlying definition
    - Number of Rows / Partitioning
  - Examine Columns in Where Clause
    - Cardinality of columns /
    - Data Skew / Histograms
  - Statistic Gathering
    - Tip: Out-of-date statistics can impact performance

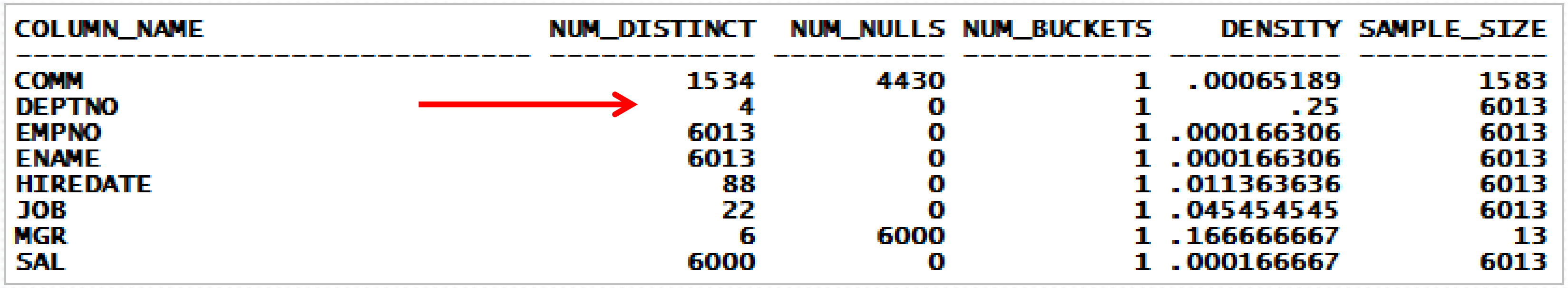
```
SELECT e.empno EID,
etc…
FROM  emp e,  dept d
WHERE  d.deptno =  :P1
AND e.deptno = d.deptno;
```

- Use TuningStats.sql
  - OracleTuningStats.sql
- Run it for expensive data access targets

© 2015 SOLARWINDS WORLDWIDE, LLC. ALL RIGHTS RESERVED.

REVIEW TABLE & COLUMN STATISTICS

SELECT column_name, num_distinct, num_nulls, num_buckets, density, sample_size
FROM user_tab_columns
WHERE table_name = ’EMP’
ORDER BY column_name;



```
COLUMN_NAME                    NUM_DISTINCT  NUM_NULLS NUM_BUCKETS    DENSITY SAMPLE_SIZE
------------------------------ ------------ ---------- ----------- ---------- -----------
COMM                                   1534       4430           1  .00065189        1583
DEPTNO                                    4          0           1        .25        6013
EMPNO                                  6013          0           1 .000166306        6013
ENAME                                  6013          0           1 .000166306        6013
HIREDATE                                 88          0           1 .011363636        6013
JOB                                      22          0           1 .045454545        6013
MGR                                       6       6000           1 .166666667          13
SAL                                    6000          0           1 .000166667        6013
```

SELECT count(*) FROM EMP;

```
COUNT(*)
-------------
 6013
```

SELECT 6013/4 dist FROM DUAL;

```
DIST
------
1503
```

SELECT DEPTNO, count(*) FROM EMP
GROUP BY DEPTNO;

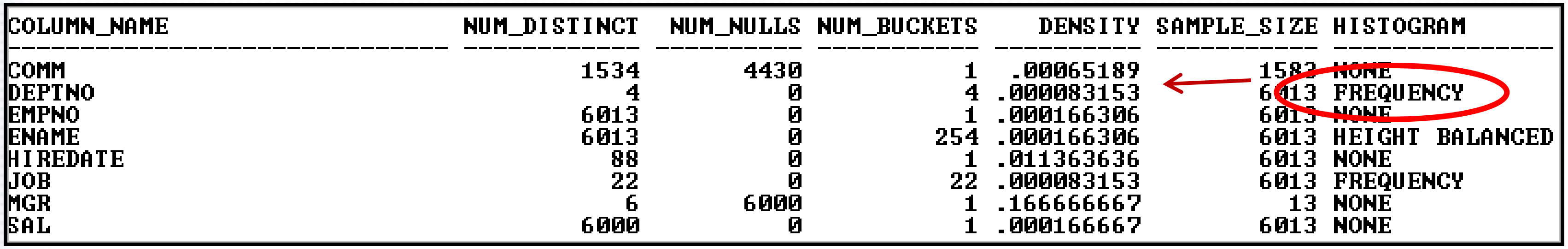
```
DEPTNO   COUNT(*)
---------- -------------
  10      77
  20      1500
  30      478
  40      3958
```

Would an index on EMP.DEPTNO increase performance?

© 2015 SOLARWINDS WORLDWIDE, LLC. ALL RIGHTS RESERVED.

HISTOGRAMS

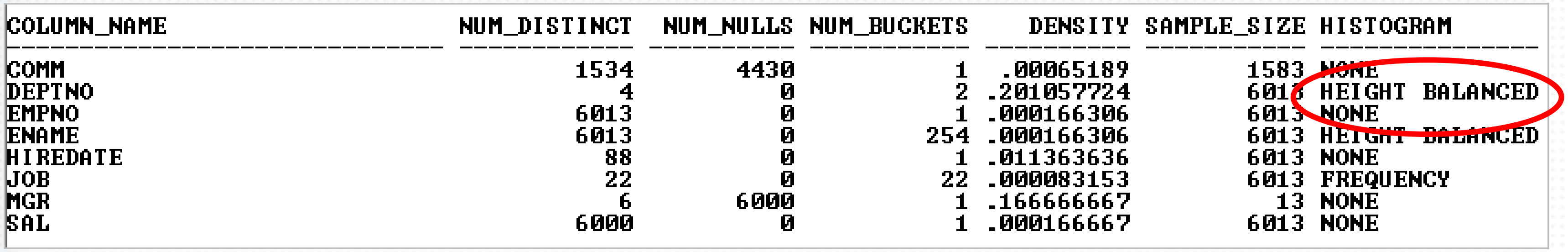
exec dbms_stats.gather_schema_stats(ownname => 'SCOTT', options => 'GATHER AUTO', estimate_percent => dbms_stats.auto_sample_size, method_opt => 'for all columns size auto')



```
COLUMN_NAME                    NUM_DISTINCT  NUM_NULLS NUM_BUCKETS    DENSITY SAMPLE_SIZE HISTOGRAM
------------------------------ ------------ ---------- ----------- ---------- ----------- ---------------
COMM                                   1534       4430           1  .00065189        1583 NONE
DEPTNO                                    4          0           4 .000083153        6013 FREQUENCY
EMPNO                                  6013          0           1 .000166306        6013 NONE
ENAME                                  6013          0         254 .000166306        6013 HEIGHT BALANCED
HIREDATE                                 88          0           1 .011363636        6013 NONE
JOB                                      22          0          22 .000083153        6013 FREQUENCY
MGR                                       6       6000           1 .166666667          13 NONE
SAL                                    6000          0           1 .000166667        6013 NONE
```

exec dbms_stats.gather_table_stats(ownname => 'SCOTT',

tabname => 'EMP', method_opt=>'FOR COLUMNS deptno SIZE 2');



```
COLUMN_NAME                    NUM_DISTINCT  NUM_NULLS NUM_BUCKETS    DENSITY SAMPLE_SIZE HISTOGRAM
------------------------------ ------------ ---------- ----------- ---------- ----------- ---------------
COMM                                   1534       4430           1  .00065189        1583 NONE
DEPTNO                                    4          0           2 .201057724        6013 HEIGHT BALANCED
EMPNO                                  6013          0           1 .000166306        6013 NONE
ENAME                                  6013          0         254 .000166306        6013 HEIGHT BALANCED
HIREDATE                                 88          0           1 .011363636        6013 NONE
JOB                                      22          0          22 .000083153        6013 FREQUENCY
MGR                                       6       6000           1 .166666667          13 NONE
SAL                                    6000          0           1 .000166667        6013 NONE
```

New in 12c – Top Frequency / Hybrid (Height Balanced going away)

© 2015 SOLARWINDS WORLDWIDE, LLC. ALL RIGHTS RESERVED.

6. REVIEW INDEXES & CONSTRAINTS

- Get Index definitions
  - Know the order of columns and their selectivity
- Review existing keys and constraints
  - Know Multi-Table Relationships (ERD)
    - Primary key and foreign definitions
  - Check and not null constraints
- Tip: Keys & constraints help the optimizer create better execution plans
- Make sure the optimizer can use the index
  - Functions on indexed columns can turn off index
    - Consider a function index
  - Look for implicit conversions
    - Get sample bind variable values

```
SELECT name, position, datatype_string, value_string
FROM v$sql_bind_capture
WHERE sql_id = '0zz5h1003f2dw';
```

© 2015 SOLARWINDS WORLDWIDE, LLC. ALL RIGHTS RESERVED.

MORE ON INDEXES

- If proper indexes exist but are ignored
  - Is the index invisible?
    - If so, the index won’t be used
    - Unless OPTIMIZER_USE_INVISIBLE_INDEXES parameter is true
    - Check the VISIBILITY column in DBA_INDEXES for this.
  - Are optimizer statistics up to date and correct for the table and column?
    - Check the LAST_ANALYZED column in the DBA_INDEXES & DBA_TABLES
      - Gives the most recent date when statistics were gathered.
    - Review SAMPLE_SIZE column in these tables to ensure proper number of rows
  - Does the criteria in WHERE clause match leading edge (1st column) of index?
    - If not, a skip scan could be used, - better than no index but not as efficient
    - Try to create index with a leading edge matching the criteria in the SQL statement

© 2015 SOLARWINDS WORLDWIDE, LLC. ALL RIGHTS RESERVED.

MORE ON INDEXES

- Sometimes a full scan is necessary
  - Due to the amount of data needed by the query
  - Avoid indexing small tables where a full scan may be more efficient
  - Make sure the columns in the index have good selectivity

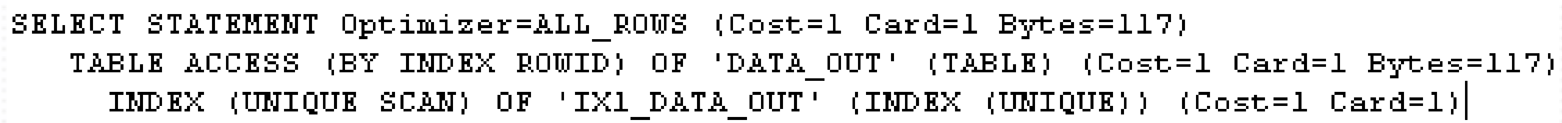
- If lots of data needs to be read, reduce wait times by:
  - If the query is summarizing data from a detailed table:
    - Consider creating a materialized view
    - Note: may not get fresh data each time - based on the frequency of refreshes
  - If many sessions running same SQL statement & data hardly changes
    - Review if a RESULTS CACHE can be used
      - Can be turned on with a hint or database parameter
    - Consider caching at the application layer

© 2015 SOLARWINDS WORLDWIDE, LLC. ALL RIGHTS RESERVED.

Data Type Issues - db file scattered read

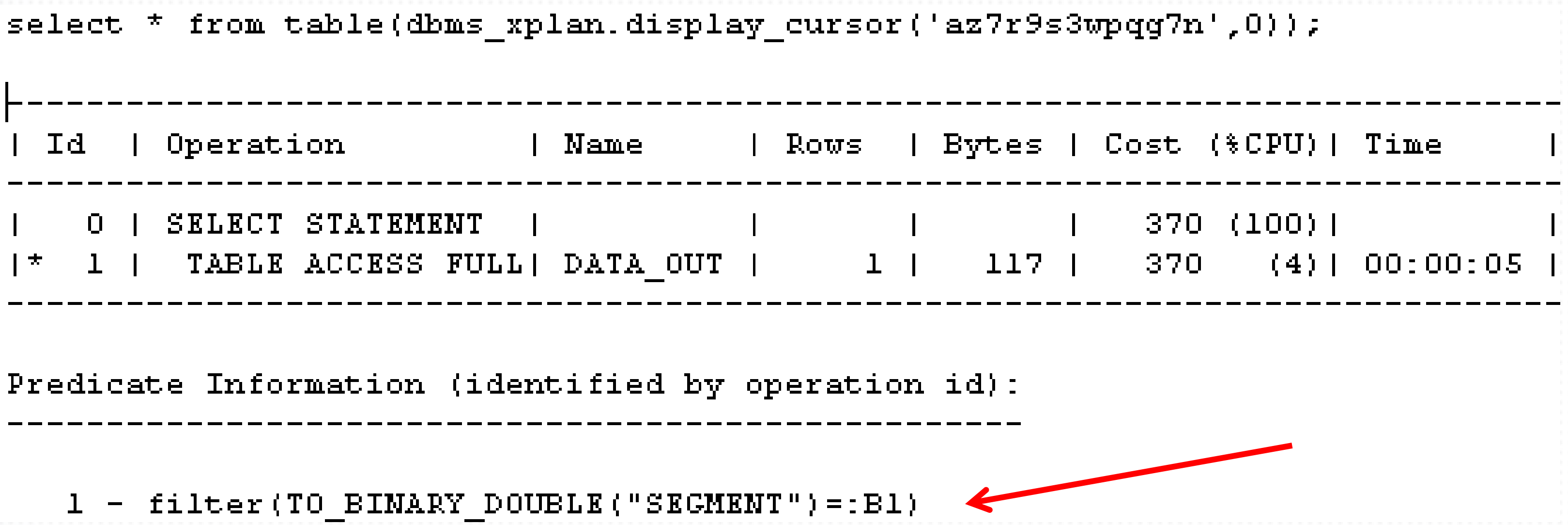
```
SELECT company, attribute FROM data_out WHERE segment = :B1
```

- Wait Time – 100% on “db file scattered read”
- Plan from EXPLAIN PLAN



```
SELECT STATEMENT Optimizer=ALL_ROWS (Cost=1 Card=1 Bytes=117)
   TABLE ACCESS (BY INDEX ROWID) OF 'DATA_OUT' (TABLE) (Cost=1 Card=1 Bytes=117)
     INDEX (UNIQUE SCAN) OF 'IX1_DATA_OUT' (INDEX (UNIQUE)) (Cost=1 Card=1)
```

- Plan from V$SQL_PLAN using DBMS_XPLAN



```
select * from table(dbms_xplan.display_cursor('az7r9s3wpqg7n',0));

-----------------------------------------------------------------------------
| Id  | Operation         | Name     | Rows  | Bytes | Cost (%CPU)| Time     |
-----------------------------------------------------------------------------
|   0 | SELECT STATEMENT  |          |       |       |   370 (100)|          |
|*  1 |  TABLE ACCESS FULL| DATA_OUT |     1 |   117 |   370   (4)| 00:00:05 |
-----------------------------------------------------------------------------

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

   1 - filter(TO_BINARY_DOUBLE("SEGMENT")=:B1)
```

© 2015 SOLARWINDS WORLDWIDE, LLC. ALL RIGHTS RESERVED.

7. CAN'T CHANGE THE QUERY

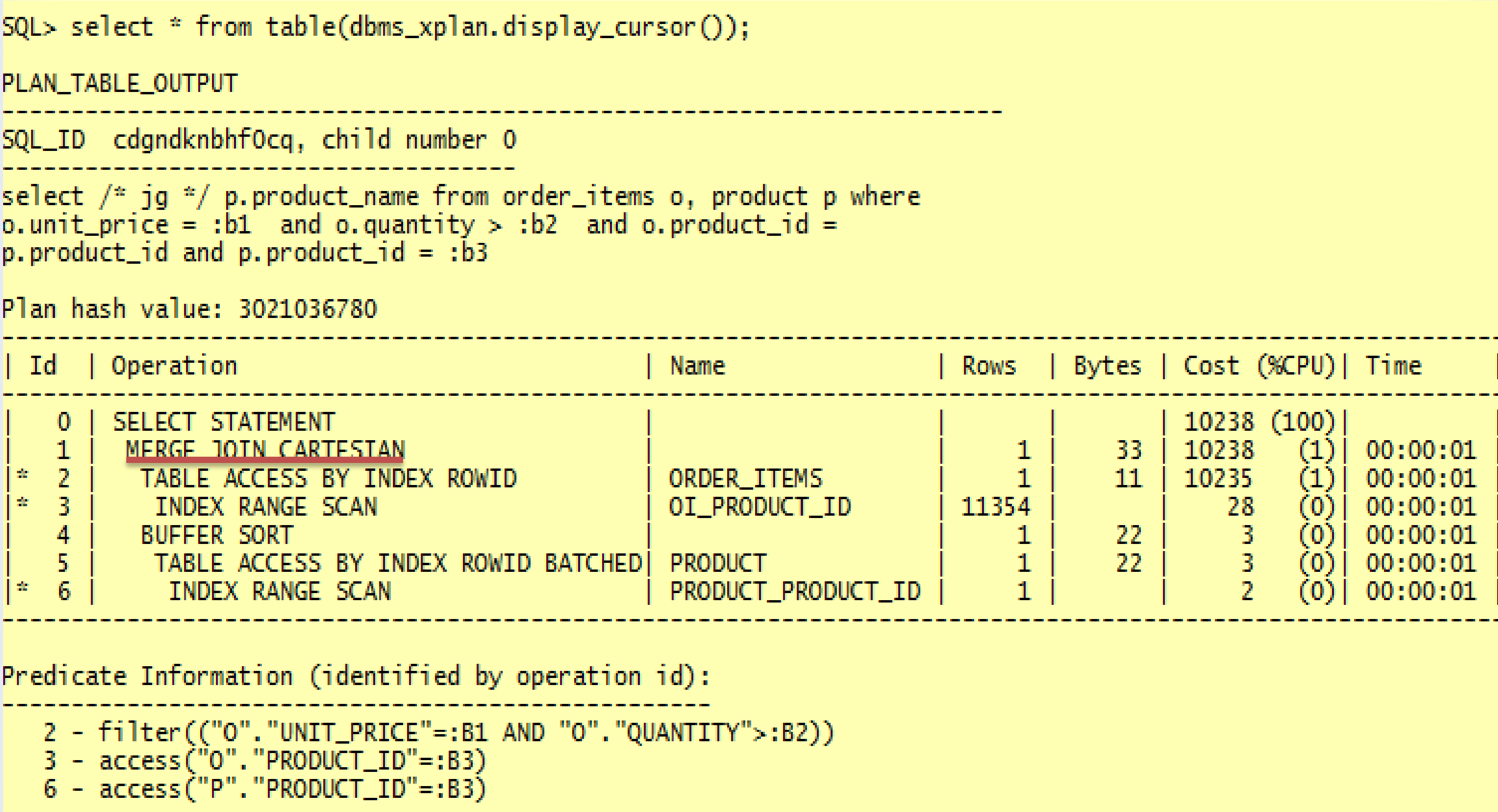
- If you can hint it, baseline it (per Tom Kyte)
- Alternative to using hints
  - 3rd Party Software – can't modify code
  - Hints difficult to manage over time
  - Once added, usually forgotten about
- Example:

```
SQL> var b1 number
  2  var b2 number
  3  var b3 number
  4  exec :b1 := 3358;
  5  exec :b2 :=1;
  6* exec :b3 :=205;
```

```
SQL> select /* jg */ p.product_name
  2  from order_items o, product p
  3  where o.unit_price = :b1
  4  and o.quantity > :b2
  5  and o.product_id = p.product_id
  6* and p.product_id = :b3;

PRODUCT_NAME
--------------------------------------------------
1L2H8Zq e D2ex9blrIcUXzF2q4j
```

© 2015 SOLARWINDS WORLDWIDE, LLC. ALL RIGHTS RESERVED.

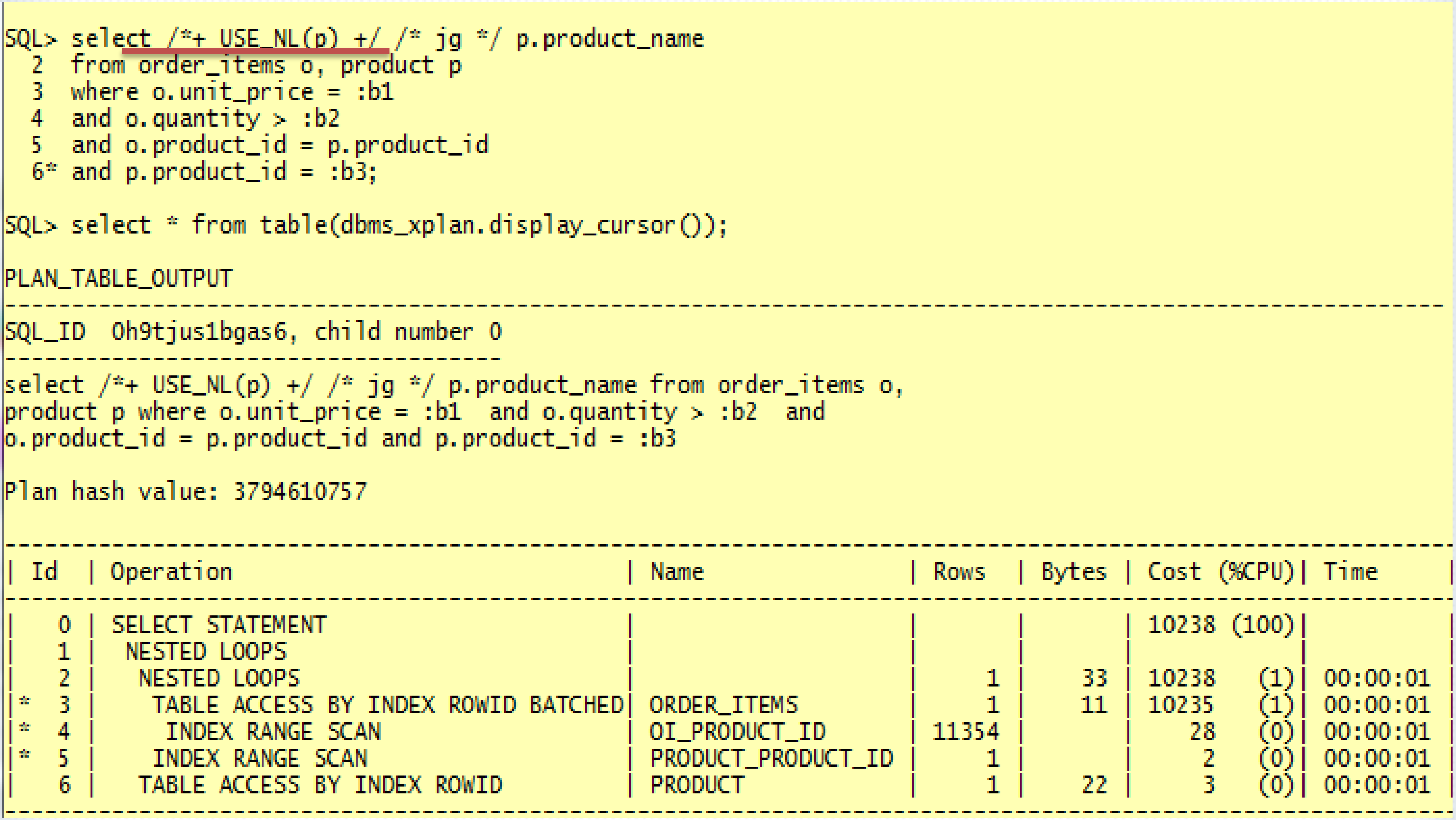
CHANGE THE BASELINE



```
SQL> select * from table(dbms_xplan.display_cursor());

PLAN_TABLE_OUTPUT
------------------------------------------------------------------------
SQL_ID  cdgndknbhf0cq, child number 0
-------------------------------------
select /* jg */ p.product_name from order_items o, product p where
o.unit_price = :b1  and o.quantity > :b2  and o.product_id =
p.product_id and p.product_id = :b3

Plan hash value: 3021036780

-----------------------------------------------------------------------------------------------------------
| Id  | Operation                             | Name               | Rows  | Bytes | Cost (%CPU)| Time     |
-----------------------------------------------------------------------------------------------------------
|   0 | SELECT STATEMENT                      |                    |       |       | 10238 (100)|          |
|   1 |  MERGE JOIN CARTESIAN                 |                    |     1 |    33 | 10238   (1)| 00:00:01 |
|*  2 |   TABLE ACCESS BY INDEX ROWID         | ORDER_ITEMS        |     1 |    11 | 10235   (1)| 00:00:01 |
|*  3 |    INDEX RANGE SCAN                   | OI_PRODUCT_ID      | 11354 |       |    28   (0)| 00:00:01 |
|   4 |   BUFFER SORT                         |                    |     1 |    22 |     3   (0)| 00:00:01 |
|   5 |    TABLE ACCESS BY INDEX ROWID BATCHED| PRODUCT            |     1 |    22 |     3   (0)| 00:00:01 |
|*  6 |     INDEX RANGE SCAN                  | PRODUCT_PRODUCT_ID |     1 |       |     2   (0)| 00:00:01 |
-----------------------------------------------------------------------------------------------------------

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

   2 - filter(("O"."UNIT_PRICE"=:B1 AND "O"."QUANTITY">:B2))
   3 - access("O"."PRODUCT_ID"=:B3)
   6 - access("P"."PRODUCT_ID"=:B3)
```

© 2015 SOLARWINDS WORLDWIDE, LLC. ALL RIGHTS RESERVED.

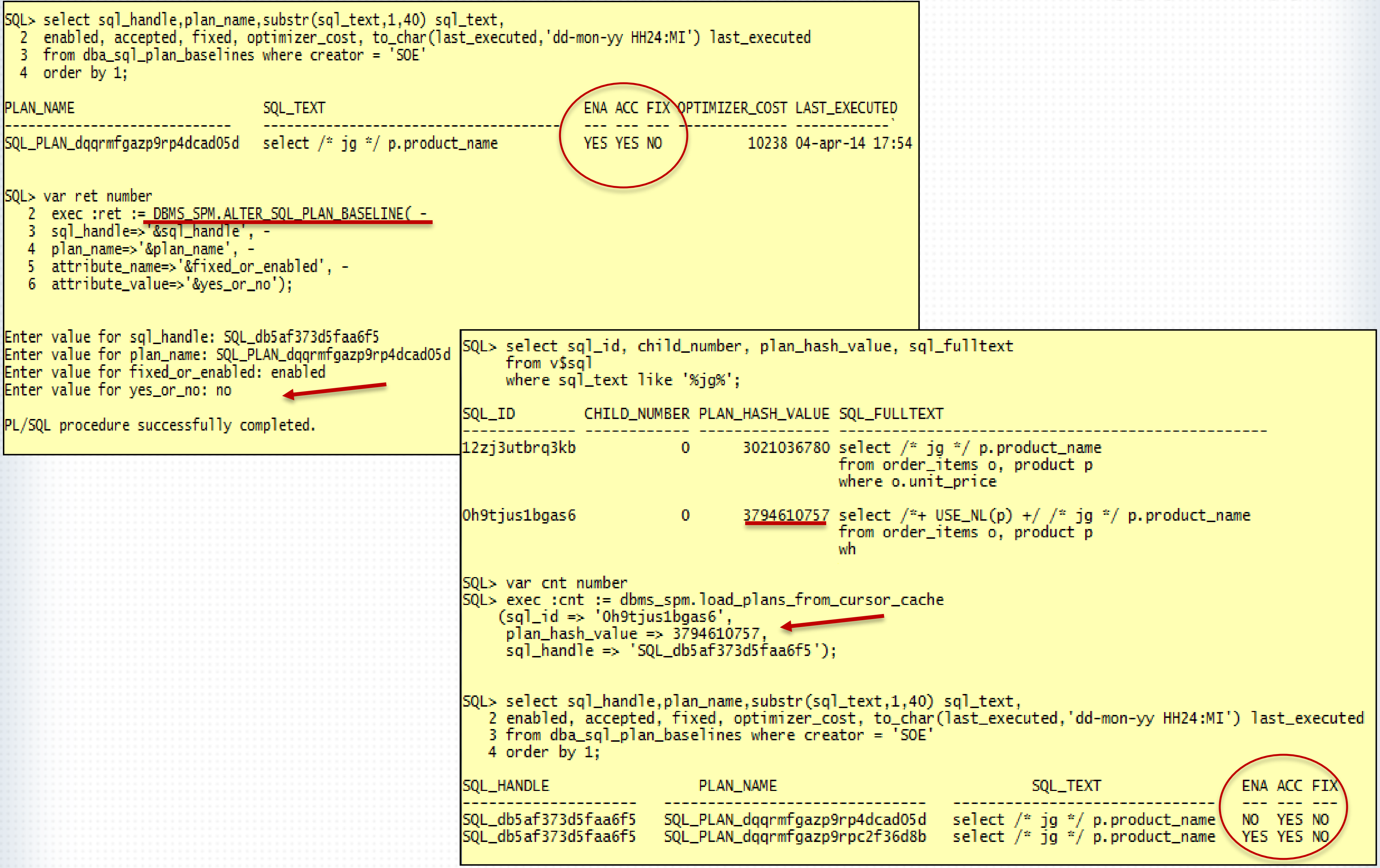
CHANGE THE BASELINE



```
SQL> select /*+ USE_NL(p) +/ /* jg */ p.product_name
  2  from order_items o, product p
  3  where o.unit_price = :b1
  4  and o.quantity > :b2
  5  and o.product_id = p.product_id
  6* and p.product_id = :b3;

SQL> select * from table(dbms_xplan.display_cursor());

PLAN_TABLE_OUTPUT
-----------------------------------------------------------------------------------------------------------
SQL_ID  0h9tjus1bgas6, child number 0
-------------------------------------
select /*+ USE_NL(p) +/ /* jg */ p.product_name from order_items o,
product p where o.unit_price = :b1  and o.quantity > :b2  and
o.product_id = p.product_id and p.product_id = :b3

Plan hash value: 3794610757

-----------------------------------------------------------------------------------------------------------
| Id  | Operation                              | Name                 | Rows  | Bytes | Cost (%CPU)| Time     |
-----------------------------------------------------------------------------------------------------------
|   0 | SELECT STATEMENT                       |                      |       |       | 10238 (100)|          |
|   1 |  NESTED LOOPS                          |                      |       |       |            |          |
|   2 |   NESTED LOOPS                         |                      |     1 |    33 | 10238   (1)| 00:00:01 |
|*  3 |    TABLE ACCESS BY INDEX ROWID BATCHED| ORDER_ITEMS          |     1 |    11 | 10235   (1)| 00:00:01 |
|*  4 |     INDEX RANGE SCAN                   | OI_PRODUCT_ID        | 11354 |       |    28   (0)| 00:00:01 |
|*  5 |    INDEX RANGE SCAN                    | PRODUCT_PRODUCT_ID   |     1 |       |     2   (0)| 00:00:01 |
|   6 |   TABLE ACCESS BY INDEX ROWID          | PRODUCT              |     1 |    22 |     3   (0)| 00:00:01 |
-----------------------------------------------------------------------------------------------------------
```

© 2015 SOLARWINDS WORLDWIDE, LLC. ALL RIGHTS RESERVED.

CHANGE THE BASELINE



```
SQL> select sql_handle,plan_name,substr(sql_text,1,40) sql_text,
  2  enabled, accepted, fixed, optimizer_cost, to_char(last_executed,'dd-mon-yy HH24:MI') last_executed
  3  from dba_sql_plan_baselines where creator = 'SOE'
  4  order by 1;

PLAN_NAME                      SQL_TEXT                                 ENA ACC FIX OPTIMIZER_COST LAST_EXECUTED
------------------------------ ---------------------------------------- --- --- --- -------------- ------------
SQL_PLAN_dqqrmfgazp9rp4dcad05d select /* jg */ p.product_name           YES YES NO           10238 04-apr-14 17:54


SQL> var ret number
   2  exec :ret := DBMS_SPM.ALTER_SQL_PLAN_BASELINE( -
   3  sql_handle=>'&sql_handle', -
   4  plan_name=>'&plan_name', -
   5  attribute_name=>'&fixed_or_enabled', -
   6  attribute_value=>'&yes_or_no');


Enter value for sql_handle: SQL_db5af373d5faa6f5
Enter value for plan_name: SQL_PLAN_dqqrmfgazp9rp4dcad05d
Enter value for fixed_or_enabled: enabled
Enter value for yes_or_no: no

PL/SQL procedure successfully completed.
```

```
SQL> select sql_id, child_number, plan_hash_value, sql_fulltext
     from v$sql
     where sql_text like '%jg%';

SQL_ID        CHILD_NUMBER PLAN_HASH_VALUE SQL_FULLTEXT
------------- ------------ --------------- --------------------------------------------------
12zj3utbrq3kb            0      3021036780 select /* jg */ p.product_name
                                           from order_items o, product p
                                           where o.unit_price

0h9tjus1bgas6            0      3794610757 select /*+ USE_NL(p) +/ /* jg */ p.product_name
                                           from order_items o, product p
                                           wh

SQL> var cnt number
SQL> exec :cnt := dbms_spm.load_plans_from_cursor_cache
    (sql_id => '0h9tjus1bgas6',
     plan_hash_value => 3794610757,
     sql_handle => 'SQL_db5af373d5faa6f5');


SQL> select sql_handle,plan_name,substr(sql_text,1,40) sql_text,
  2 enabled, accepted, fixed, optimizer_cost, to_char(last_executed,'dd-mon-yy HH24:MI') last_executed
  3 from dba_sql_plan_baselines where creator = 'SOE'
  4 order by 1;

SQL_HANDLE           PLAN_NAME                      SQL_TEXT                       ENA ACC FIX
-------------------- ------------------------------ ------------------------------ --- --- ---
SQL_db5af373d5faa6f5 SQL_PLAN_dqqrmfgazp9rp4dcad05d select /* jg */ p.product_name NO  YES NO
SQL_db5af373d5faa6f5 SQL_PLAN_dqqrmfgazp9rpc2f36d8b select /* jg */ p.product_name YES YES NO
```

© 2015 SOLARWINDS WORLDWIDE, LLC. ALL RIGHTS RESERVED.

8. ENGINEER OUT THE STUPID

» Look for Performance Inhibitors

- Cursor or row by row processing
- Parallel processing
- Hard-coded Hints
- Nested views that use db_links
- Abuse of Wild Cards (*) or No Where Clause
- Code-based SQL Generators (e.g. Hibernate)
- Non-SARG-able / Scalar Functions
  - Select… where upper(first_name) = 'JANIS'

© 2015 SOLARWINDS WORLDWIDE, LLC. ALL RIGHTS RESERVED.

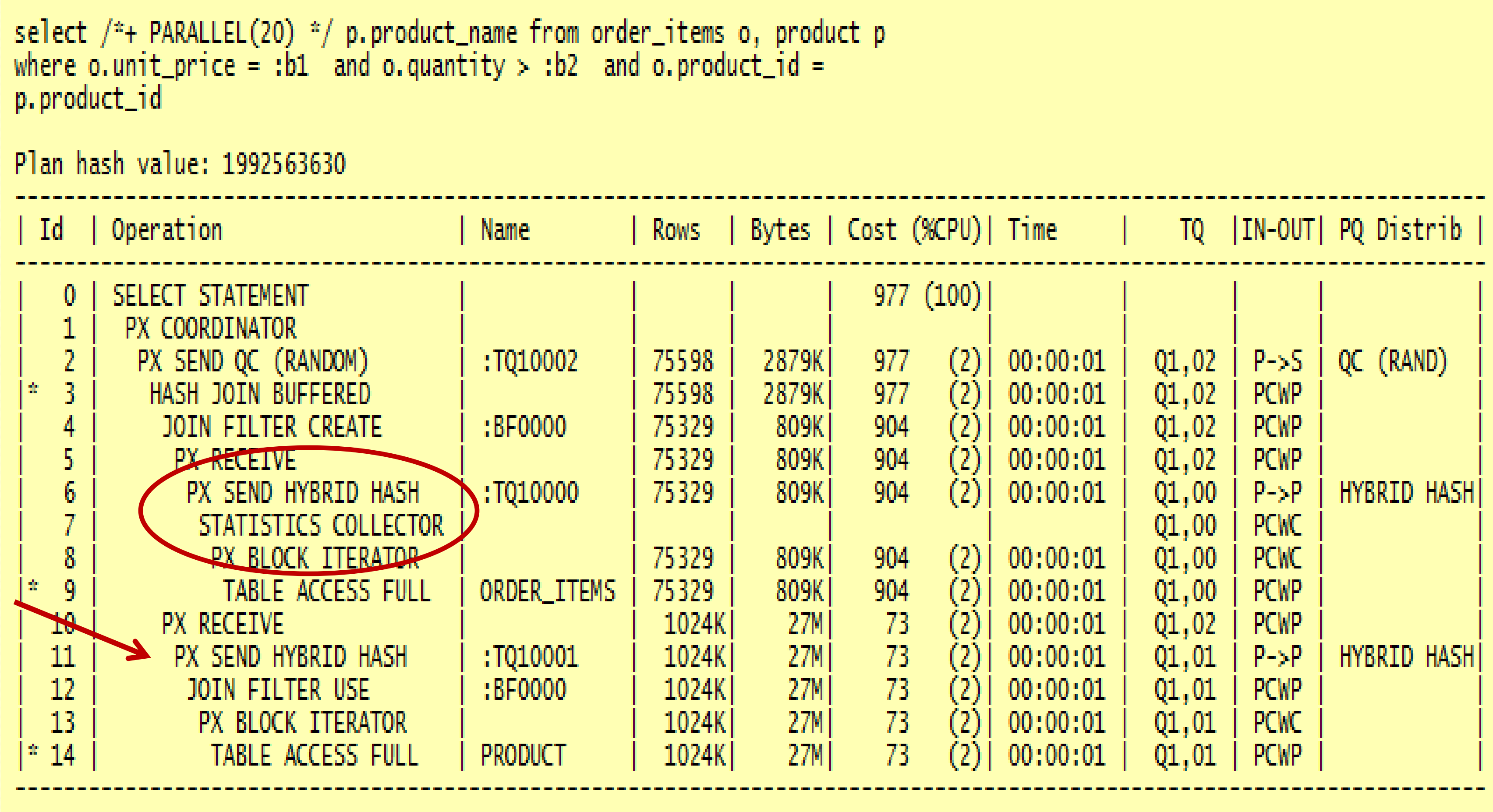
12C PARALLEL DISTRIBUTION

- Parallel execution needs to distribute data across all parallel processes
  - For sorts, aggregation & join operations
  - Chosen method depends on number of rows & Degree of Parallelism (DOP)
- Potential performance problem if few parallel processes distribute many rows
  - Data skew could cause unequal distribution of rows
- New Hybrid Hash distribution technique
  - Optimizer decides final data distribution method during execution time
  - Statistic collectors are inserted in front of the parallel server processes
    - On producer side of the operation.
  - Chooses:
    - Hash, if rows > than threshold
    - Broadcast, if rows < than threshold
    - Threshold defined as 2 X DOP

© 2015 SOLARWINDS WORLDWIDE, LLC. ALL RIGHTS RESERVED.

12C PARALLEL DISTRIBUTION

Uses Hybrid Hash - 77,329 rows greater than threshold of 40 (2 x 20 DOP = 40)



```
select /*+ PARALLEL(20) */ p.product_name from order_items o, product p
where o.unit_price = :b1  and o.quantity > :b2  and o.product_id =
p.product_id

Plan hash value: 1992563630

--------------------------------------------------------------------------------------------------------------
| Id  | Operation                    | Name        | Rows  | Bytes | Cost (%CPU)| Time     |    TQ  |IN-OUT| PQ Distrib |
--------------------------------------------------------------------------------------------------------------
|   0 | SELECT STATEMENT             |             |       |       |   977 (100)|          |        |      |            |
|   1 |  PX COORDINATOR              |             |       |       |            |          |        |      |            |
|   2 |   PX SEND QC (RANDOM)        | :TQ10002    | 75598 | 2879K|   977   (2)| 00:00:01 |  Q1,02 | P->S | QC (RAND)  |
|*  3 |    HASH JOIN BUFFERED        |             | 75598 | 2879K|   977   (2)| 00:00:01 |  Q1,02 | PCWP |            |
|   4 |     JOIN FILTER CREATE       | :BF0000     | 75329 |  809K|   904   (2)| 00:00:01 |  Q1,02 | PCWP |            |
|   5 |      PX RECEIVE              |             | 75329 |  809K|   904   (2)| 00:00:01 |  Q1,02 | PCWP |            |
|   6 |       PX SEND HYBRID HASH    | :TQ10000    | 75329 |  809K|   904   (2)| 00:00:01 |  Q1,00 | P->P | HYBRID HASH|
|   7 |        STATISTICS COLLECTOR  |             |       |       |            |          |  Q1,00 | PCWC |            |
|   8 |         PX BLOCK ITERATOR    |             | 75329 |  809K|   904   (2)| 00:00:01 |  Q1,00 | PCWC |            |
|*  9 |          TABLE ACCESS FULL   | ORDER_ITEMS | 75329 |  809K|   904   (2)| 00:00:01 |  Q1,00 | PCWP |            |
|  10 |     PX RECEIVE               |             |  1024K|   27M|    73   (2)| 00:00:01 |  Q1,02 | PCWP |            |
|  11 |      PX SEND HYBRID HASH     | :TQ10001    |  1024K|   27M|    73   (2)| 00:00:01 |  Q1,01 | P->P | HYBRID HASH|
|  12 |       JOIN FILTER USE        | :BF0000     |  1024K|   27M|    73   (2)| 00:00:01 |  Q1,01 | PCWP |            |
|  13 |        PX BLOCK ITERATOR     |             |  1024K|   27M|    73   (2)| 00:00:01 |  Q1,01 | PCWC |            |
|* 14 |         TABLE ACCESS FULL    | PRODUCT     |  1024K|   27M|    73   (2)| 00:00:01 |  Q1,01 | PCWP |            |
--------------------------------------------------------------------------------------------------------------
```

© 2015 SOLARWINDS WORLDWIDE, LLC. ALL RIGHTS RESERVED.

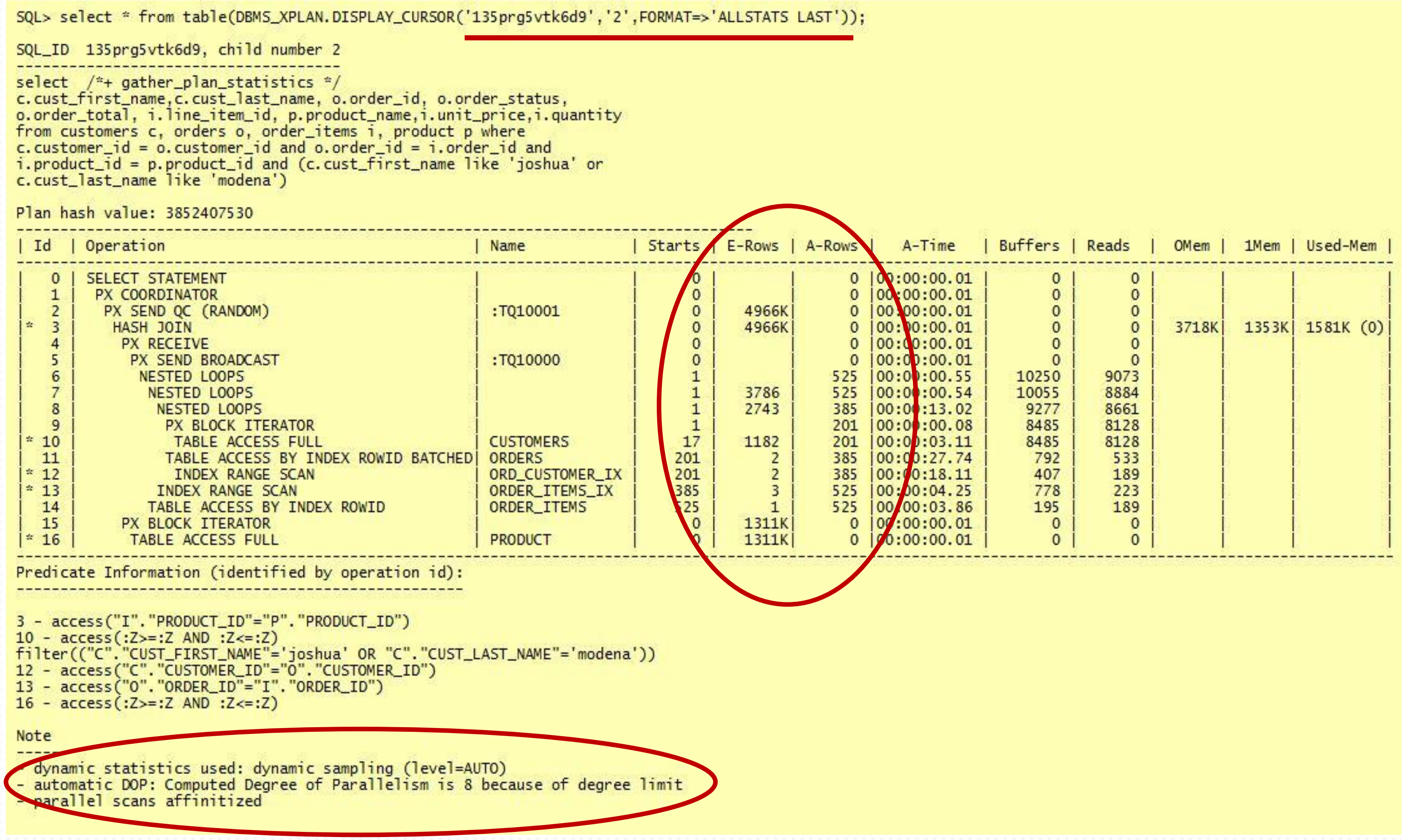
12C PERFORMANCE FEEDBACK

- Automatically improves the degree of parallelism
  - Init.ora parameter, PARALLEL_DEGREE_POLICY = ’ADAPTIVE’
- On 1st execution, the optimizer decides
  - Whether to execute the statement in parallel
  - The degree of parallelism based on estimates
- After 1st execution, optimizer compares
  - Estimates with actual performance statistics
    - e.g. CPU Time
    - i.e. PARALLEL_MIN_TIME_THRESHOD
  - If significantly different, the statement
    - is marked for reparsing
    - new execution statistics are stored as feedback
- Following executions use the performance feedback to determine DOP
- If PARALLEL_DEGREE_POLICY not set, statistics feedback may change DOP

© 2015 SOLARWINDS WORLDWIDE, LLC. ALL RIGHTS RESERVED.

PERFORMANCE FEEDBACK

Alter session set PARALLEL_DEGREE_POLICY = ‘ADAPTIVE’;



```
SQL> select * from table(DBMS_XPLAN.DISPLAY_CURSOR('135prg5vtk6d9','2',FORMAT=>'ALLSTATS LAST'));

SQL_ID  135prg5vtk6d9, child number 2
-------------------------------------
select  /*+ gather_plan_statistics */
c.cust_first_name,c.cust_last_name, o.order_id, o.order_status,
o.order_total, i.line_item_id, p.product_name,i.unit_price,i.quantity
from customers c, orders o, order_items i, product p where
c.customer_id = o.customer_id and o.order_id = i.order_id and
i.product_id = p.product_id and (c.cust_first_name like 'joshua' or
c.cust_last_name like 'modena')

Plan hash value: 3852407530

---------------------------------------------------------------------------------------------------------------------------------------------------------------
| Id  | Operation                                   | Name             | Starts | E-Rows | A-Rows |   A-Time    | Buffers |  Reads |  OMem |  1Mem | Used-Mem |
---------------------------------------------------------------------------------------------------------------------------------------------------------------
|   0 | SELECT STATEMENT                            |                  |      0 |        |      0 | 00:00:00.01 |       0 |      0 |       |       |          |
|   1 |  PX COORDINATOR                             |                  |      0 |        |      0 | 00:00:00.01 |       0 |      0 |       |       |          |
|   2 |   PX SEND QC (RANDOM)                       | :TQ10001         |      0 |  4966K |      0 | 00:00:00.01 |       0 |      0 |       |       |          |
|*  3 |    HASH JOIN                                |                  |      0 |  4966K |      0 | 00:00:00.01 |       0 |      0 | 3718K | 1353K | 1581K (0)|
|   4 |     PX RECEIVE                              |                  |      0 |        |      0 | 00:00:00.01 |       0 |      0 |       |       |          |
|   5 |      PX SEND BROADCAST                      | :TQ10000         |      0 |        |      0 | 00:00:00.01 |       0 |      0 |       |       |          |
|   6 |       NESTED LOOPS                          |                  |      1 |        |    525 | 00:00:00.55 |   10250 |   9073 |       |       |          |
|   7 |        NESTED LOOPS                         |                  |      1 |   3786 |    525 | 00:00:00.54 |   10055 |   8884 |       |       |          |
|   8 |         NESTED LOOPS                        |                  |      1 |   2743 |    385 | 00:00:13.02 |    9277 |   8661 |       |       |          |
|   9 |          PX BLOCK ITERATOR                  |                  |      1 |        |    201 | 00:00:00.08 |    8485 |   8128 |       |       |          |
|* 10 |           TABLE ACCESS FULL                 | CUSTOMERS        |     17 |   1182 |    201 | 00:00:03.11 |    8485 |   8128 |       |       |          |
|  11 |          TABLE ACCESS BY INDEX ROWID BATCHED| ORDERS           |    201 |      2 |    385 | 00:00:27.74 |     792 |    533 |       |       |          |
|* 12 |           INDEX RANGE SCAN                  | ORD_CUSTOMER_IX  |    201 |      2 |    385 | 00:00:18.11 |     407 |    189 |       |       |          |
|* 13 |         INDEX RANGE SCAN                    | ORDER_ITEMS_IX   |    385 |      3 |    525 | 00:00:04.25 |     778 |    223 |       |       |          |
|  14 |        TABLE ACCESS BY INDEX ROWID          | ORDER_ITEMS      |    525 |      1 |    525 | 00:00:03.86 |     195 |    189 |       |       |          |
|  15 |     PX BLOCK ITERATOR                       |                  |      0 |  1311K |      0 | 00:00:00.01 |       0 |      0 |       |       |          |
|* 16 |      TABLE ACCESS FULL                      | PRODUCT          |      0 |  1311K |      0 | 00:00:00.01 |       0 |      0 |       |       |          |
---------------------------------------------------------------------------------------------------------------------------------------------------------------
Predicate Information (identified by operation id):
---------------------------------------------------

3 - access("I"."PRODUCT_ID"="P"."PRODUCT_ID")
10 - access(:Z>=:Z AND :Z<=:Z)
filter(("C"."CUST_FIRST_NAME"='joshua' OR "C"."CUST_LAST_NAME"='modena'))
12 - access("C"."CUSTOMER_ID"="O"."CUSTOMER_ID")
13 - access("O"."ORDER_ID"="I"."ORDER_ID")
16 - access(:Z>=:Z AND :Z<=:Z)

Note
-----
- dynamic statistics used: dynamic sampling (level=AUTO)
- automatic DOP: Computed Degree of Parallelism is 8 because of degree limit
- parallel scans affinitized
```

© 2015 SOLARWINDS WORLDWIDE, LLC. ALL RIGHTS RESERVED.

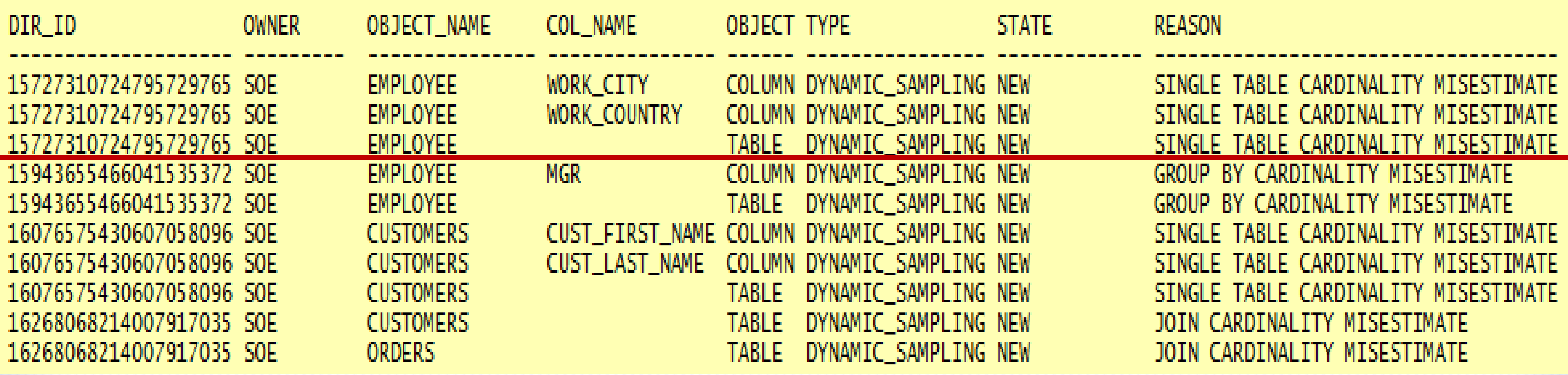
SQL PLAN DIRECTIVES

- Are additional Instructions for missing column group statistics or histograms
  - Dynamic sampling performed on directive
    - Until statistics are gathered for the column group (e.g. City / State / Country)
- Not tied to a specific sql statement – defined on a query expression
  - Can be used by similar queries
- Are created in shared_pool & periodically written to SYSAUX tablespace
  - DBA_SQL_PLAN_DIRECTIVES
  - DBA_SQL_PLAN_DIR_OBJECTS
- Use DBMS_STATS extended functions & procedures
  - CREATE_EXTENDED_STATS
  - SHOW_EXTENDED_STATS_NAME
  - DROP_EXENTED_STATS

© 2015 SOLARWINDS WORLDWIDE, LLC. ALL RIGHTS RESERVED.

SQL PLAN DIRECTIVES

```
SELECT  TO_CHAR(d.directive_id) dir_id,
          o.owner, o.object_name, o.subobject_name col_name,
          o.object_type, d.type,d.state,d.reason
FROM dba_sql_plan_directives d, dba_sql_plan_dir_objects o
WHERE d.directive_id = o.directive_id
AND o.owner IN ('SOE')
ORDER BY 1,2,3,4,5;
```

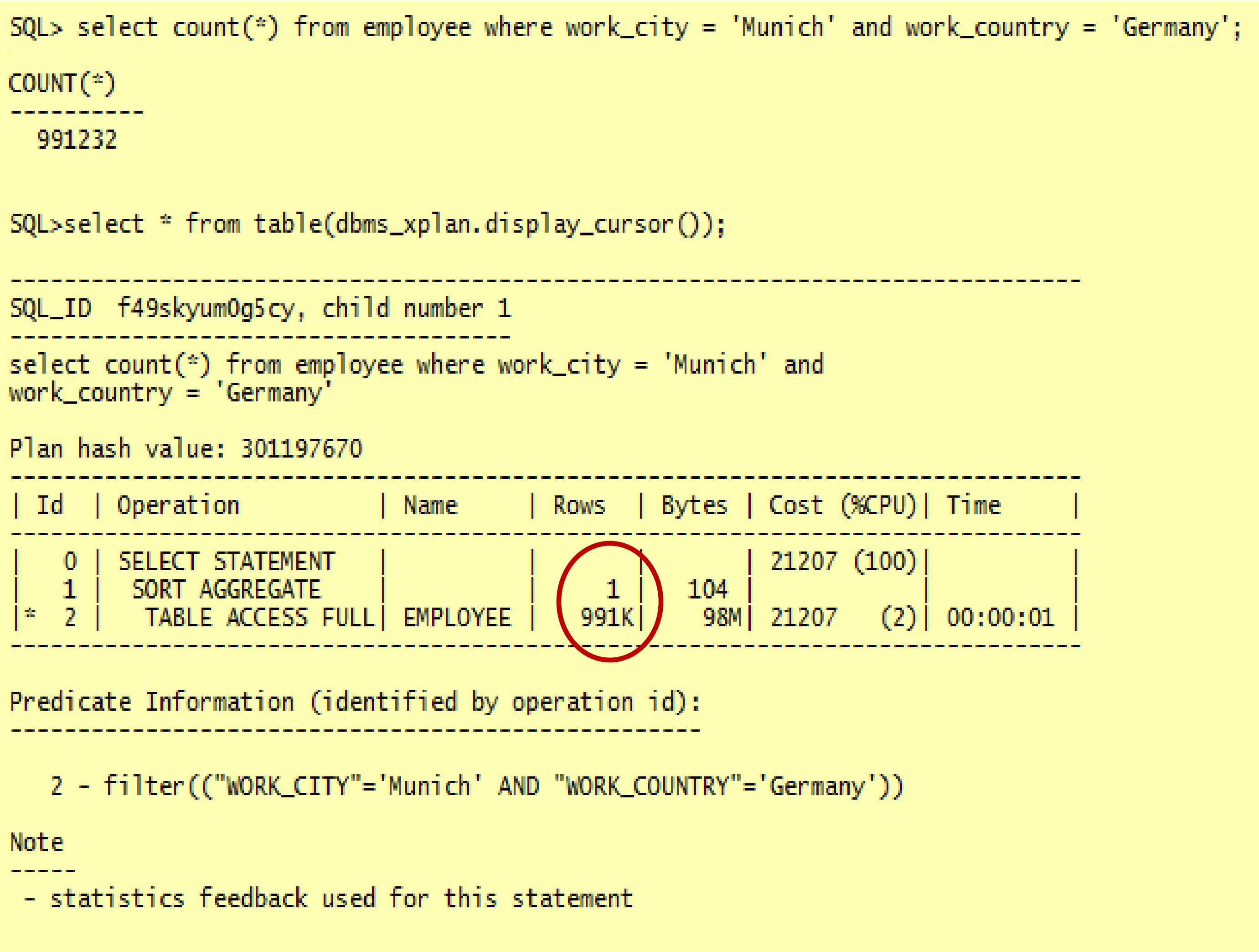


```
DIR_ID               OWNER      OBJECT_NAME     COL_NAME        OBJECT TYPE             STATE         REASON
-------------------- ---------  --------------- --------------- ------ ---------------- ------------- ------------------------------------
15727310724795729765 SOE        EMPLOYEE        WORK_CITY       COLUMN DYNAMIC_SAMPLING NEW           SINGLE TABLE CARDINALITY MISESTIMATE
15727310724795729765 SOE        EMPLOYEE        WORK_COUNTRY    COLUMN DYNAMIC_SAMPLING NEW           SINGLE TABLE CARDINALITY MISESTIMATE
15727310724795729765 SOE        EMPLOYEE                        TABLE  DYNAMIC_SAMPLING NEW           SINGLE TABLE CARDINALITY MISESTIMATE
15943655466041535372 SOE        EMPLOYEE        MGR             COLUMN DYNAMIC_SAMPLING NEW           GROUP BY CARDINALITY MISESTIMATE
15943655466041535372 SOE        EMPLOYEE                        TABLE  DYNAMIC_SAMPLING NEW           GROUP BY CARDINALITY MISESTIMATE
16076575430607058096 SOE        CUSTOMERS       CUST_FIRST_NAME COLUMN DYNAMIC_SAMPLING NEW           SINGLE TABLE CARDINALITY MISESTIMATE
16076575430607058096 SOE        CUSTOMERS       CUST_LAST_NAME  COLUMN DYNAMIC_SAMPLING NEW           SINGLE TABLE CARDINALITY MISESTIMATE
16076575430607058096 SOE        CUSTOMERS                       TABLE  DYNAMIC_SAMPLING NEW           SINGLE TABLE CARDINALITY MISESTIMATE
16268068214007917035 SOE        CUSTOMERS                       TABLE  DYNAMIC_SAMPLING NEW           JOIN CARDINALITY MISESTIMATE
16268068214007917035 SOE        ORDERS                          TABLE  DYNAMIC_SAMPLING NEW           JOIN CARDINALITY MISESTIMATE
```

© 2015 SOLARWINDS WORLDWIDE, LLC. ALL RIGHTS RESERVED.

SQL PLAN DIRECTIVES

No Statistics on Employee table so Optimizer uses Directive

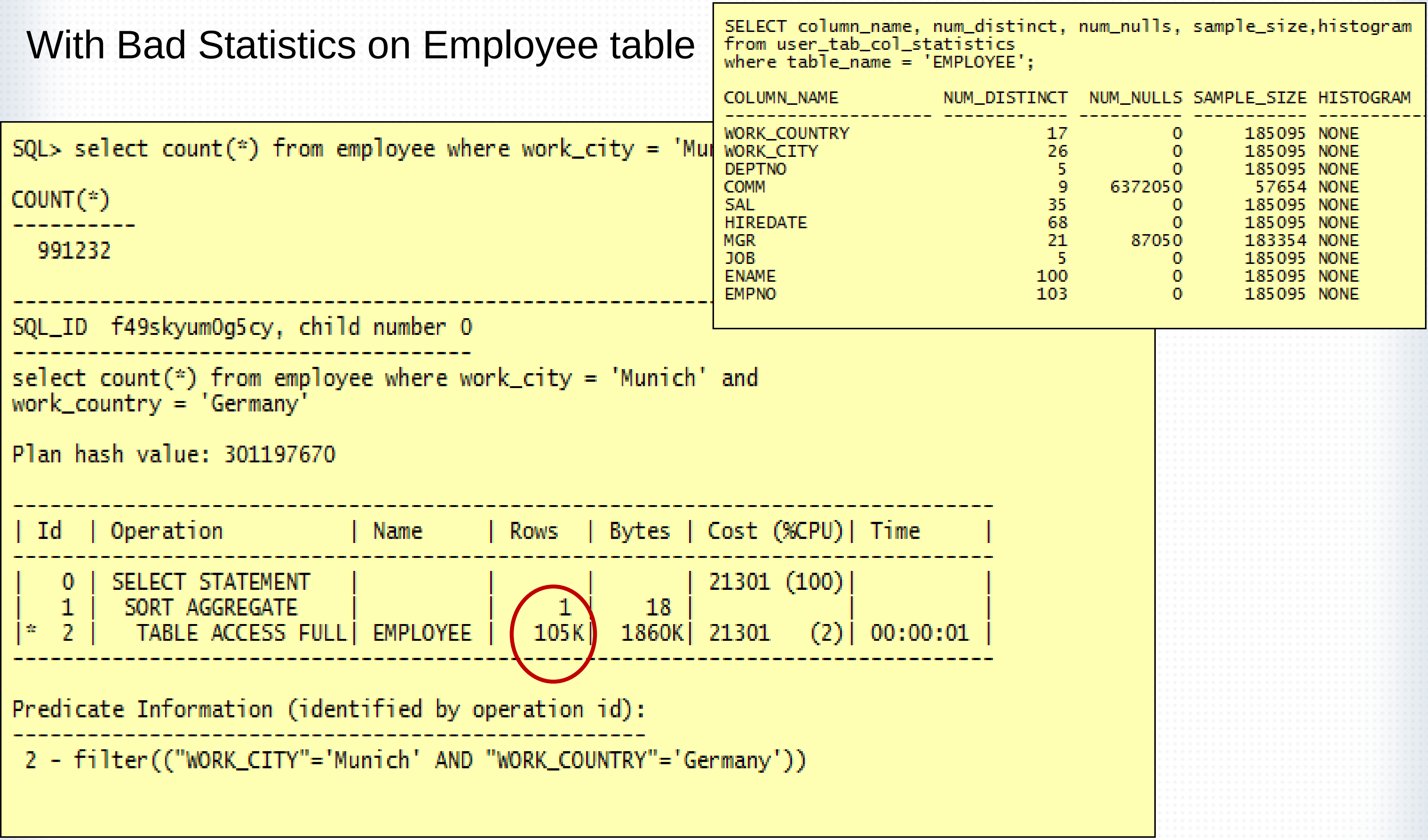


```
SQL> select count(*) from employee where work_city = 'Munich' and work_country = 'Germany';

COUNT(*)
----------
  991232


SQL>select * from table(dbms_xplan.display_cursor());

--------------------------------------------------------------------------------
SQL_ID  f49skyum0g5cy, child number 1
-------------------------------------
select count(*) from employee where work_city = 'Munich' and
work_country = 'Germany'

Plan hash value: 301197670
--------------------------------------------------------------------------------
| Id  | Operation          | Name     | Rows  | Bytes | Cost (%CPU)| Time     |
--------------------------------------------------------------------------------
|   0 | SELECT STATEMENT   |          |       |       | 21207 (100)|          |
|   1 |  SORT AGGREGATE    |          |     1 |   104 |            |          |
|*  2 |   TABLE ACCESS FULL| EMPLOYEE |   991K|    98M| 21207   (2)| 00:00:01 |
--------------------------------------------------------------------------------

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

   2 - filter(("WORK_CITY"='Munich' AND "WORK_COUNTRY"='Germany'))

Note
-----
 - statistics feedback used for this statement
```

© 2015 SOLARWINDS WORLDWIDE, LLC. ALL RIGHTS RESERVED.

SQL PLAN DIRECTIVES

With Bad Statistics on Employee table

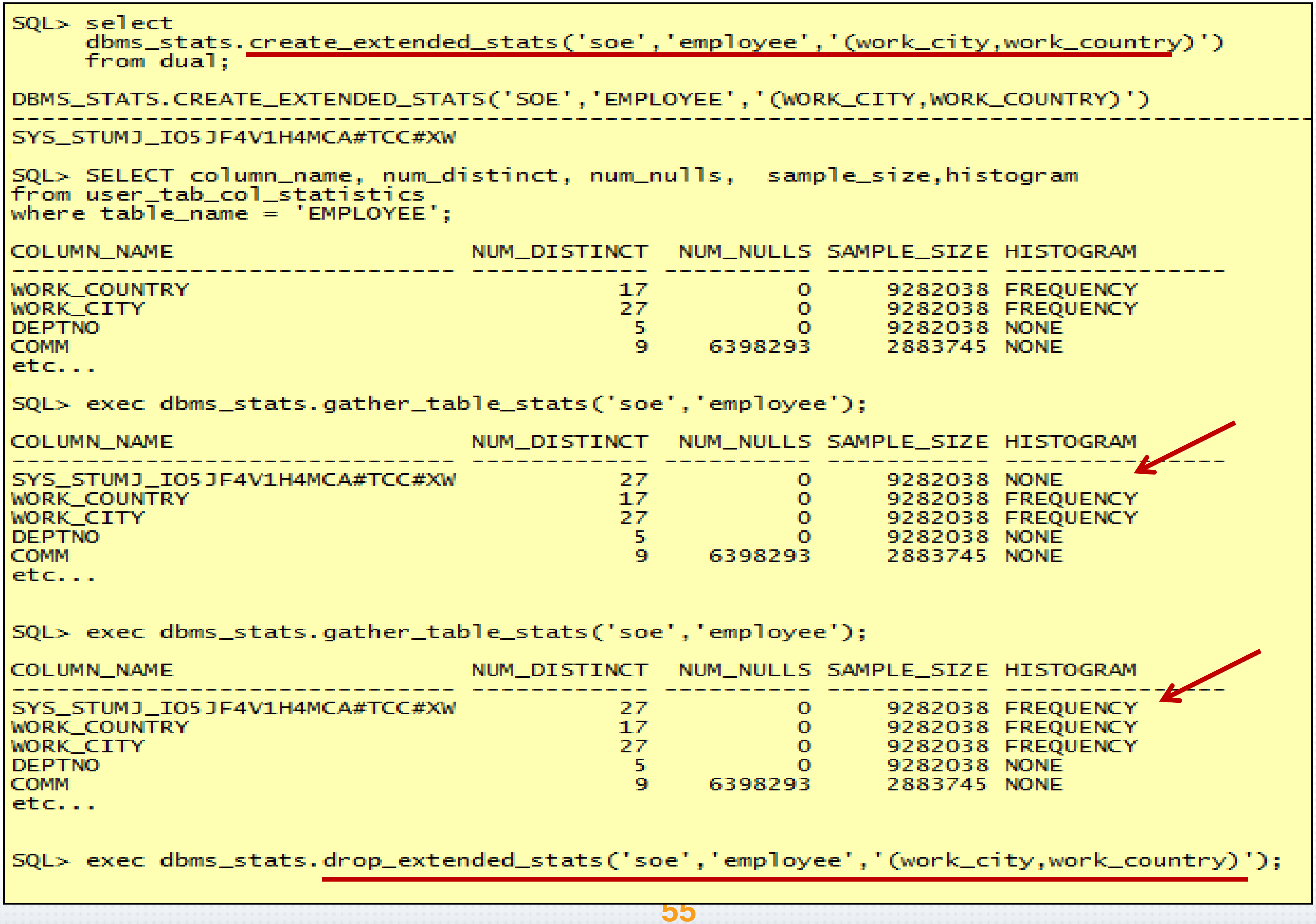


```
SELECT column_name, num_distinct, num_nulls, sample_size,histogram
from user_tab_col_statistics
where table_name = 'EMPLOYEE';

COLUMN_NAME          NUM_DISTINCT  NUM_NULLS SAMPLE_SIZE HISTOGRAM
-------------------- ------------ ---------- ----------- ---------
WORK_COUNTRY                   17          0      185095 NONE
WORK_CITY                      26          0      185095 NONE
DEPTNO                          5          0      185095 NONE
COMM                            9    6372050       57654 NONE
SAL                            35          0      185095 NONE
HIREDATE                       68          0      185095 NONE
MGR                            21      87050      183354 NONE
JOB                             5          0      185095 NONE
ENAME                         100          0      185095 NONE
EMPNO                         103          0      185095 NONE
```

```
SQL> select count(*) from employee where work_city = 'Mu

COUNT(*)
----------
  991232

------------------------------------------------------
SQL_ID  f49skyum0g5cy, child number 0
-------------------------------------
select count(*) from employee where work_city = 'Munich' and
work_country = 'Germany'

Plan hash value: 301197670

------------------------------------------------------------------------------
| Id  | Operation          | Name     | Rows  | Bytes | Cost (%CPU)| Time     |
------------------------------------------------------------------------------
|   0 | SELECT STATEMENT   |          |       |       | 21301 (100)|          |
|   1 |  SORT AGGREGATE    |          |     1 |    18 |            |          |
|*  2 |   TABLE ACCESS FULL| EMPLOYEE |  105K|  1860K| 21301   (2)| 00:00:01 |
------------------------------------------------------------------------------

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

 2 - filter(("WORK_CITY"='Munich' AND "WORK_COUNTRY"='Germany'))
```

© 2015 SOLARWINDS WORLDWIDE, LLC. ALL RIGHTS RESERVED.

EXTENDED STATISTICS



```
SQL> select
     dbms_stats.create_extended_stats('soe','employee','(work_city,work_country)')
     from dual;

DBMS_STATS.CREATE_EXTENDED_STATS('SOE','EMPLOYEE','(WORK_CITY,WORK_COUNTRY)')
------------------------------------------------------------------------------------------
SYS_STUMJ_IO5JF4V1H4MCA#TCC#XW

SQL> SELECT column_name, num_distinct, num_nulls,  sample_size,histogram
from user_tab_col_statistics
where table_name = 'EMPLOYEE';

COLUMN_NAME                    NUM_DISTINCT  NUM_NULLS SAMPLE_SIZE HISTOGRAM
------------------------------ ------------ ---------- ----------- ---------------
WORK_COUNTRY                             17          0     9282038 FREQUENCY
WORK_CITY                                27          0     9282038 FREQUENCY
DEPTNO                                    5          0     9282038 NONE
COMM                                      9    6398293     2883745 NONE
etc...

SQL> exec dbms_stats.gather_table_stats('soe','employee');

COLUMN_NAME                    NUM_DISTINCT  NUM_NULLS SAMPLE_SIZE HISTOGRAM
------------------------------ ------------ ---------- ----------- ---------------
SYS_STUMJ_IO5JF4V1H4MCA#TCC#XW           27          0     9282038 NONE
WORK_COUNTRY                             17          0     9282038 FREQUENCY
WORK_CITY                                27          0     9282038 FREQUENCY
DEPTNO                                    5          0     9282038 NONE
COMM                                      9    6398293     2883745 NONE
etc...


SQL> exec dbms_stats.gather_table_stats('soe','employee');

COLUMN_NAME                    NUM_DISTINCT  NUM_NULLS SAMPLE_SIZE HISTOGRAM
------------------------------ ------------ ---------- ----------- ---------------
SYS_STUMJ_IO5JF4V1H4MCA#TCC#XW           27          0     9282038 FREQUENCY
WORK_COUNTRY                             17          0     9282038 FREQUENCY
WORK_CITY                                27          0     9282038 FREQUENCY
DEPTNO                                    5          0     9282038 NONE
COMM                                      9    6398293     2883745 NONE
etc...


SQL> exec dbms_stats.drop_extended_stats('soe','employee','(work_city,work_country)');
```

9. GATHER RUN-TIME DETAILS

- Get baseline metrics
  - How long does it take now
  - What is acceptable (10 sec, 2 min, 1 hour)
  - Get number of Buffer Gets
    - Measurement to compare against while tuning

- Collect Wait Event Information
  - Locking / Blocking (enq)
  - I/O problem (db file sequential read)
  - Latch contention (latch)
  - Network slowdown (SQL*Net)
  - May be multiple issues
  - All have different resolutions

© 2015 SOLARWINDS WORLDWIDE, LLC. ALL RIGHTS RESERVED.

10. TUNE THE QUERY

- Focus on most expensive operations first
  - Try to reduce high-cost steps
  - Read less rows
- Seeks vs scans—which is more expensive
- Review Join Methods
  - Nested loop
  - Merge Join
  - Hash join
- Use SQL Diagramming
  - To get best Execution Plan

© 2015 SOLARWINDS WORLDWIDE, LLC. ALL RIGHTS RESERVED.

CASE STUDY 1

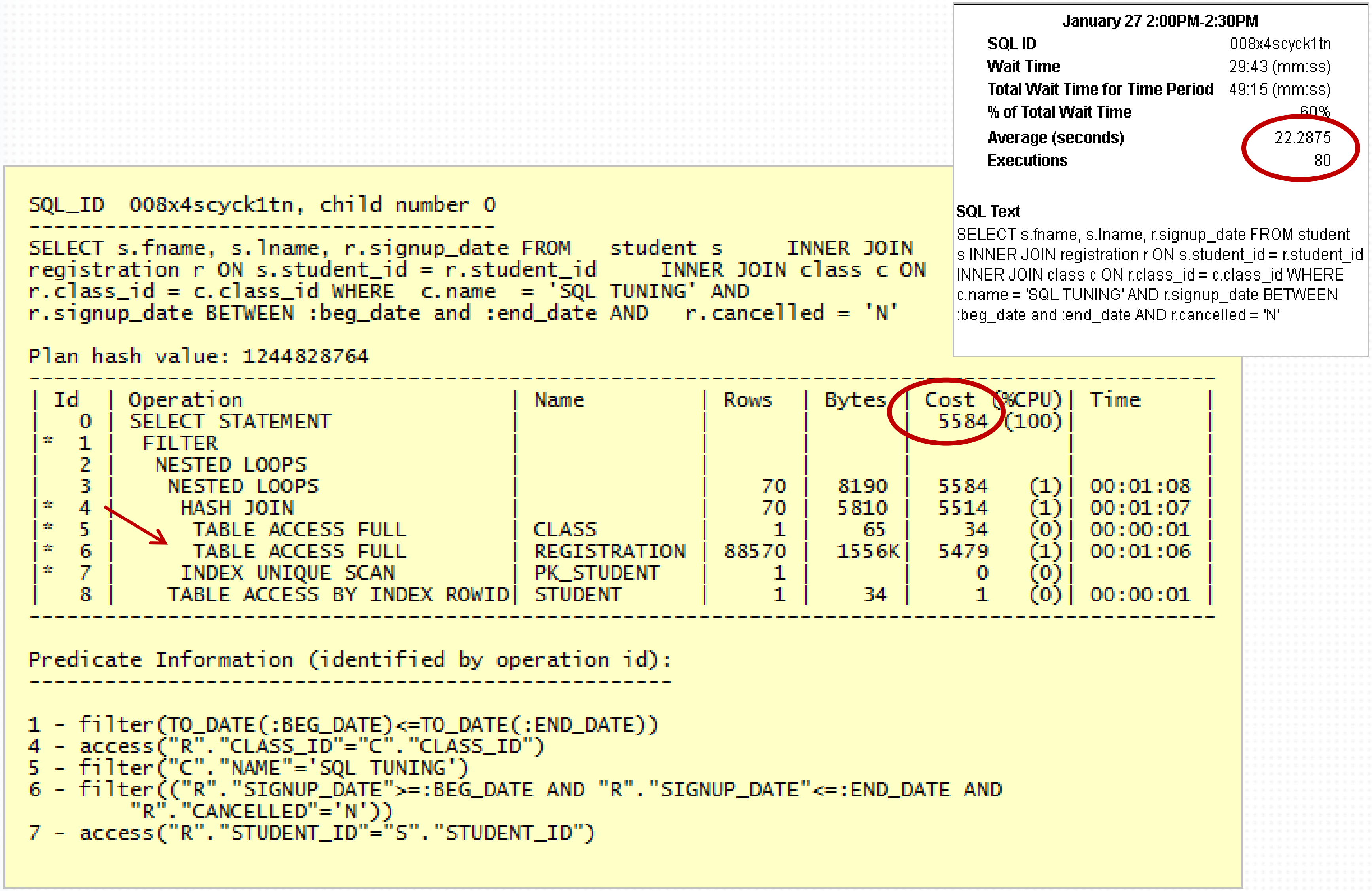
- Who registered yesterday for SQL Tuning

```
SELECT s.fname, s.lname, r.signup_date
FROM   student s
   INNER JOIN registration r ON s.student_id = r.student_id
   INNER JOIN class c ON r.class_id = c.class_id
WHERE  c.name  = 'SQL TUNING'
AND   r.signup_date BETWEEN :beg_date AND :end_date
AND   r.cancelled = 'N'
```

- **Execution Stats – 21,829 Buffer Gets**
- **Execution Time – 22 seconds to execute**
- **Wait Events – Waits 90% direct path read**

© 2015 SOLARWINDS WORLDWIDE, LLC. ALL RIGHTS RESERVED.

EXECUTION PLAN



January 27 2:00PM-2:30PM

SQL ID	008x4scyck1tn
Wait Time	29:43 (mm:ss)
Total Wait Time for Time Period	49:15 (mm:ss)
% of Total Wait Time	60%
Average (seconds)	22.2875
Executions	80

SQL Text

SELECT s.fname, s.lname, r.signup_date FROM student s INNER JOIN registration r ON s.student_id = r.student_id INNER JOIN class c ON r.class_id = c.class_id WHERE c.name = 'SQL TUNING' AND r.signup_date BETWEEN :beg_date and :end_date AND r.cancelled = 'N'

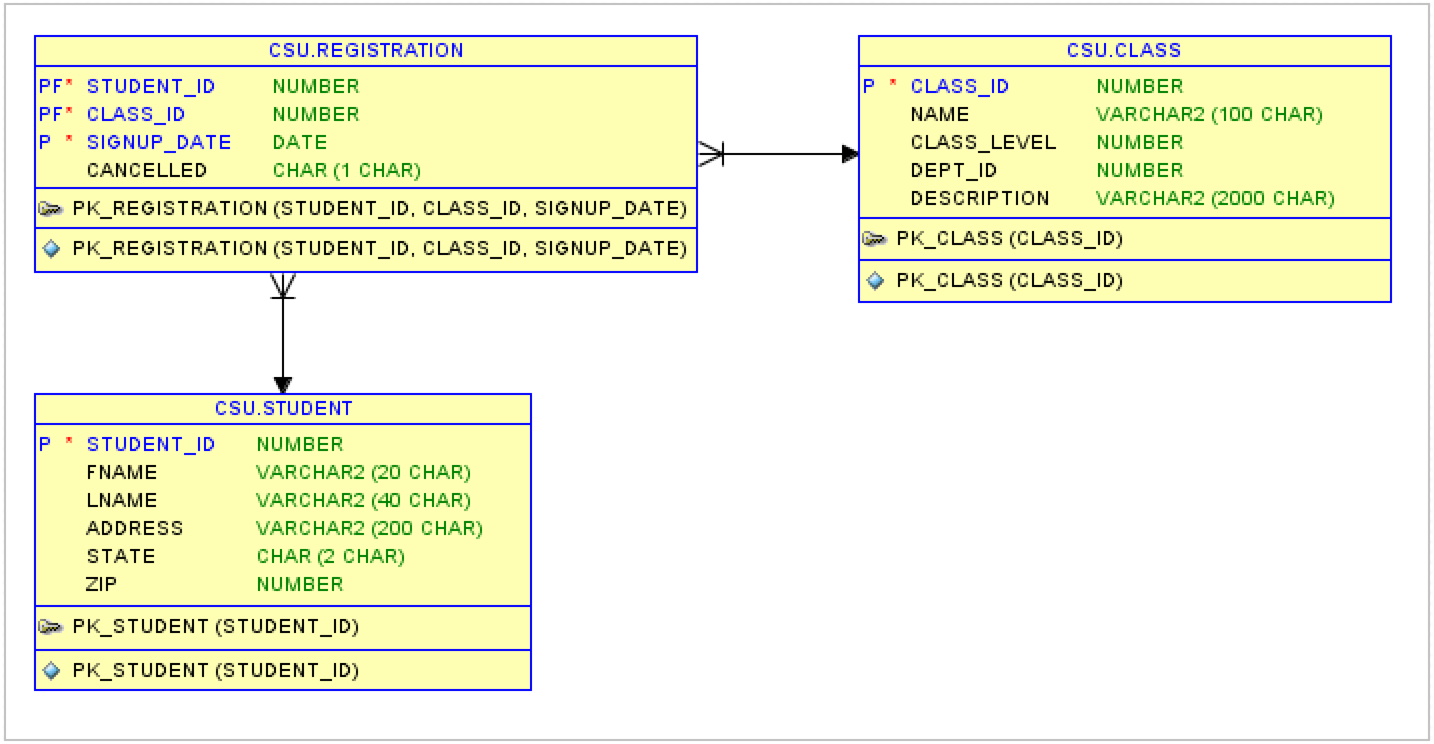
```
SQL_ID  008x4scyck1tn, child number 0
-------------------------------------
SELECT s.fname, s.lname, r.signup_date FROM   student s     INNER JOIN
registration r ON s.student_id = r.student_id     INNER JOIN class c ON
r.class_id = c.class_id WHERE  c.name  = 'SQL TUNING' AND
r.signup_date BETWEEN :beg_date and :end_date AND   r.cancelled = 'N'

Plan hash value: 1244828764

----------------------------------------------------------------------------------------------
| Id  | Operation                     | Name         | Rows  | Bytes | Cost (%CPU)| Time     |
|   0 | SELECT STATEMENT              |              |       |       |  5584 (100)|          |
|*  1 |  FILTER                       |              |       |       |            |          |
|   2 |   NESTED LOOPS                |              |       |       |            |          |
|   3 |    NESTED LOOPS               |              |    70 |  8190 |  5584   (1)| 00:01:08 |
|*  4 |     HASH JOIN                 |              |    70 |  5810 |  5514   (1)| 00:01:07 |
|*  5 |      TABLE ACCESS FULL        | CLASS        |     1 |    65 |    34   (0)| 00:00:01 |
|*  6 |      TABLE ACCESS FULL        | REGISTRATION | 88570 |  1556K|  5479   (1)| 00:01:06 |
|*  7 |     INDEX UNIQUE SCAN         | PK_STUDENT   |     1 |       |     0   (0)|          |
|   8 |    TABLE ACCESS BY INDEX ROWID| STUDENT      |     1 |    34 |     1   (0)| 00:00:01 |
----------------------------------------------------------------------------------------------

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

1 - filter(TO_DATE(:BEG_DATE)<=TO_DATE(:END_DATE))
4 - access("R"."CLASS_ID"="C"."CLASS_ID")
5 - filter("C"."NAME"='SQL TUNING')
6 - filter(("R"."SIGNUP_DATE">=:BEG_DATE AND "R"."SIGNUP_DATE"<=:END_DATE AND
        "R"."CANCELLED"='N'))
7 - access("R"."STUDENT_ID"="S"."STUDENT_ID")
```

© 2015 SOLARWINDS WORLDWIDE, LLC. ALL RIGHTS RESERVED.

RELATIONSHIP DIAGRAM

- FREE - Oracle SQL Developer Data Modeler

http://www.oracle.com/technetwork/developer-tools/datamodeler/sqldevdm31ea-download-515132.html



CSU.REGISTRATION

PF*	STUDENT_ID	NUMBER
PF*	CLASS_ID	NUMBER
P *	SIGNUP_DATE	DATE
	CANCELLED	CHAR (1 CHAR)

🔑 PK_REGISTRATION (STUDENT_ID, CLASS_ID, SIGNUP_DATE)

◆ PK_REGISTRATION (STUDENT_ID, CLASS_ID, SIGNUP_DATE)

CSU.CLASS

P *	CLASS_ID	NUMBER
	NAME	VARCHAR2 (100 CHAR)
	CLASS_LEVEL	NUMBER
	DEPT_ID	NUMBER
	DESCRIPTION	VARCHAR2 (2000 CHAR)

🔑 PK_CLASS (CLASS_ID)

◆ PK_CLASS (CLASS_ID)

CSU.STUDENT

P *	STUDENT_ID	NUMBER
	FNAME	VARCHAR2 (20 CHAR)
	LNAME	VARCHAR2 (40 CHAR)
	ADDRESS	VARCHAR2 (200 CHAR)
	STATE	CHAR (2 CHAR)
	ZIP	NUMBER

🔑 PK_STUDENT (STUDENT_ID)

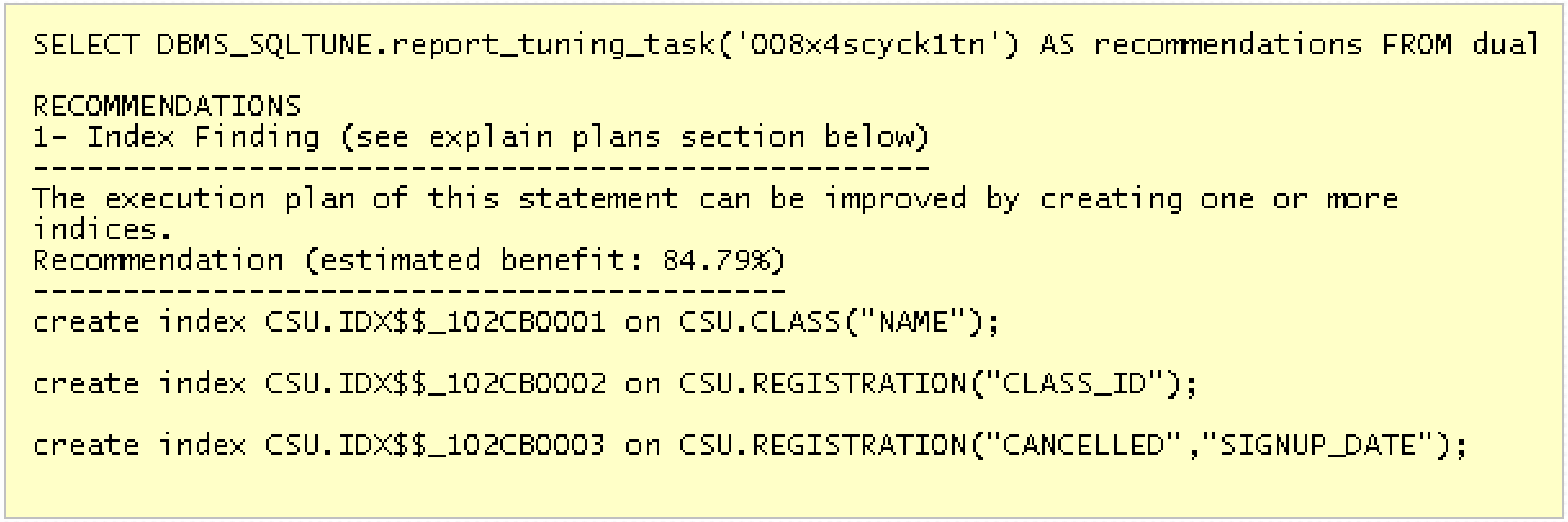
◆ PK_STUDENT (STUDENT_ID)

© 2015 SOLARWINDS WORLDWIDE, LLC. ALL RIGHTS RESERVED.

TUNING ADVISOR

- Recommends – 3 new indexes

```
DECLARE
  l_sql_tune_task_id  VARCHAR2(100);
BEGIN
  l_sql_tune_task_id := DBMS_SQLTUNE.create_tuning_task ( sql_id => '&sql_id',
   scope => DBMS_SQLTUNE.scope_comprehensive, time_limit  => 60,
   task_name => '&sql_id', description => 'Tuning task for class registration query');
  DBMS_OUTPUT.put_line('l_sql_tune_task_id: ' || l_sql_tune_task_id);
END;
/

EXEC DBMS_SQLTUNE.execute_tuning_task(task_name => '&sql_id');
```



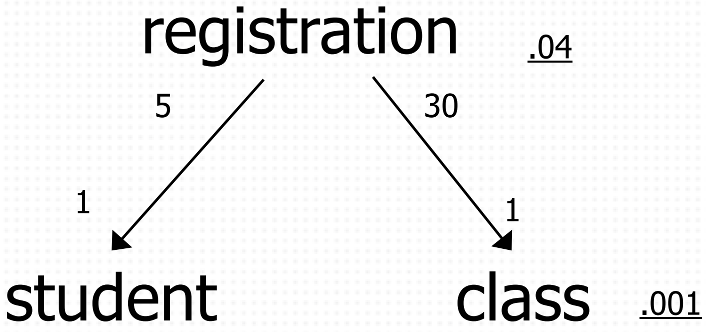
```
SELECT DBMS_SQLTUNE.report_tuning_task('008x4scyck1tn') AS recommendations FROM dual

RECOMMENDATIONS
1- Index Finding (see explain plans section below)
--------------------------------------------------
The execution plan of this statement can be improved by creating one or more
indices.
Recommendation (estimated benefit: 84.79%)
------------------------------------------
create index CSU.IDX$$_102CB0001 on CSU.CLASS("NAME");

create index CSU.IDX$$_102CB0002 on CSU.REGISTRATION("CLASS_ID");

create index CSU.IDX$$_102CB0003 on CSU.REGISTRATION("CANCELLED","SIGNUP_DATE");
```

© 2015 SOLARWINDS WORLDWIDE, LLC. ALL RIGHTS RESERVED.

SQL DIAGRAMMING

» Great Book “SQL Tuning” by Dan Tow

- Great book that teaches SQL Diagramming
- http://www.singingsql.com



```
select count(1) from registration where cancelled = 'N'
and signup_date between '2014-08-10 00:00' and '2014-08-11 00:00'

    64112 / 1783066 = .035956044


select count(1) from class where name = 'SQL TUNING'

    2 / 1,267 = .001
```

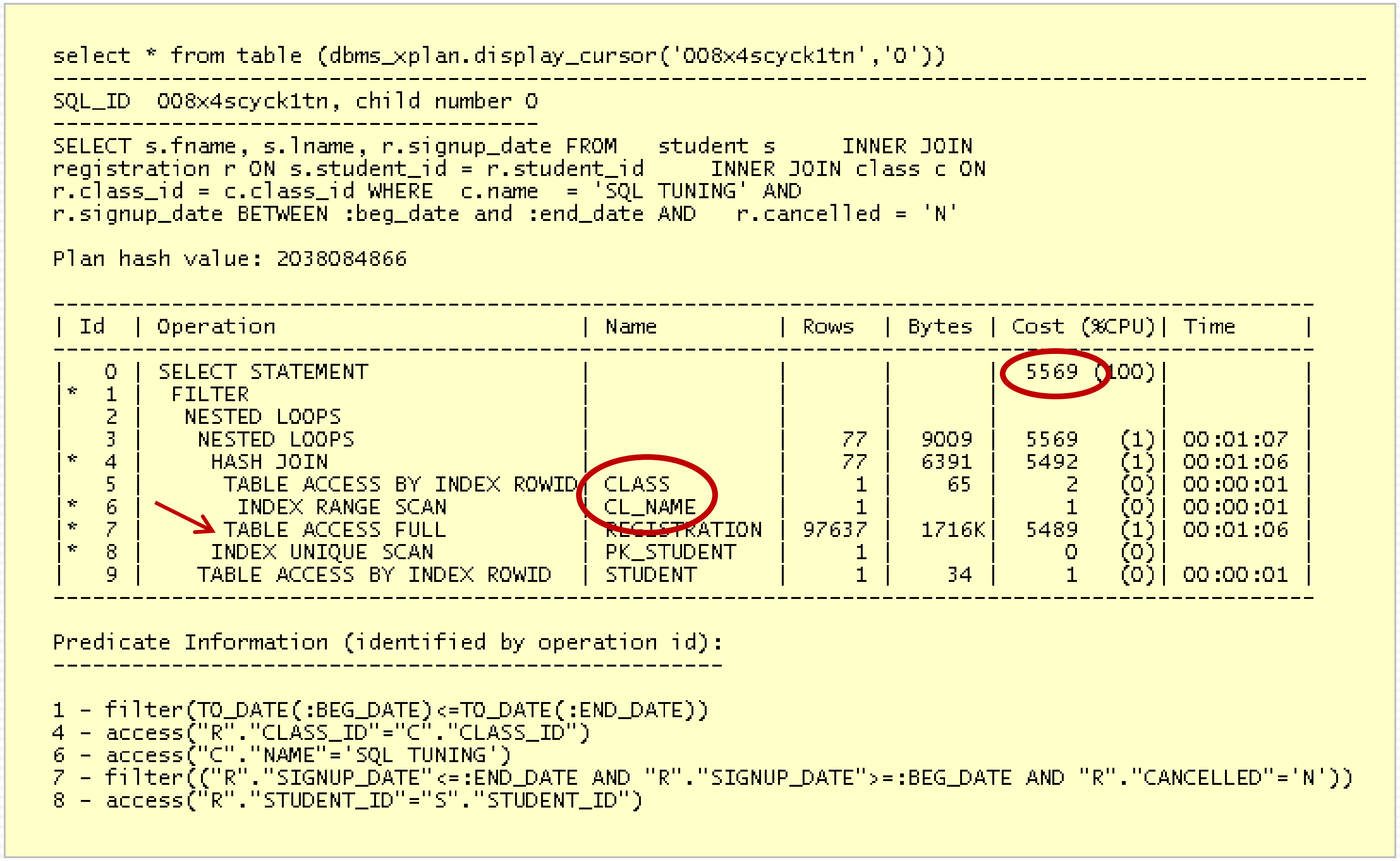
© 2015 SOLARWINDS WORLDWIDE, LLC. ALL RIGHTS RESERVED.

11. RE-RUN THE QUERY

- Make Small Changes
  - Consider adjusting indexes
  - Re-run & check run-time details
  - Compare results with baseline metrics
  - Use ‘buffer gets’ as a key measurement
  - Did you improve it? No? Rinse & Repeat

© 2015 SOLARWINDS WORLDWIDE, LLC. ALL RIGHTS RESERVED.

NEW EXECUTION PLAN



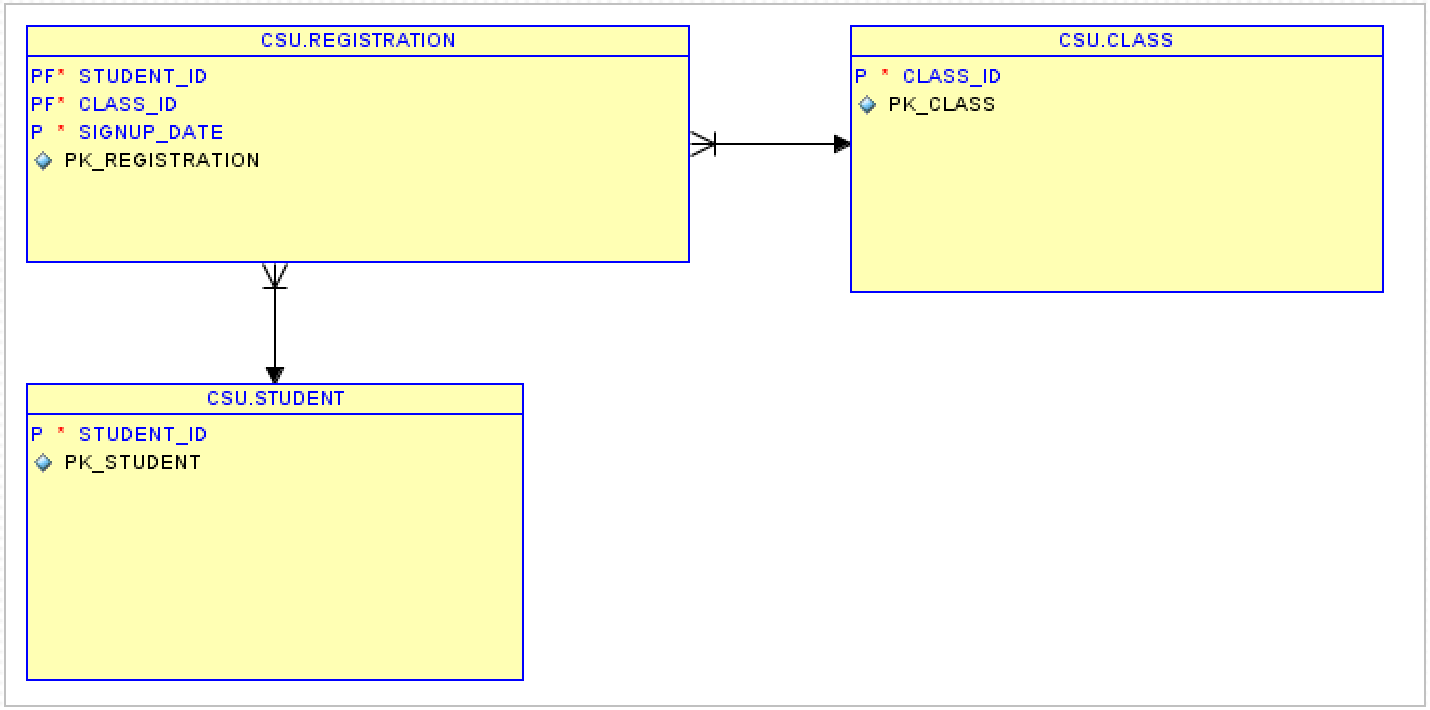
```
select * from table (dbms_xplan.display_cursor('008x4scyck1tn','0'))
-----------------------------------------------------------------------------------------------------
SQL_ID  008x4scyck1tn, child number 0
-------------------------------------
SELECT s.fname, s.lname, r.signup_date FROM   student s     INNER JOIN
registration r ON s.student_id = r.student_id     INNER JOIN class c ON
r.class_id = c.class_id WHERE  c.name  = 'SQL TUNING' AND
r.signup_date BETWEEN :beg_date and :end_date AND   r.cancelled = 'N'

Plan hash value: 2038084866

------------------------------------------------------------------------------------------------
| Id  | Operation                       | Name         | Rows  | Bytes | Cost (%CPU)| Time     |
------------------------------------------------------------------------------------------------
|   0 | SELECT STATEMENT                |              |       |       |  5569 (100)|          |
|*  1 |  FILTER                         |              |       |       |            |          |
|   2 |   NESTED LOOPS                  |              |       |       |            |          |
|   3 |    NESTED LOOPS                 |              |    77 |  9009 |  5569   (1)| 00:01:07 |
|*  4 |     HASH JOIN                   |              |    77 |  6391 |  5492   (1)| 00:01:06 |
|   5 |      TABLE ACCESS BY INDEX ROWID| CLASS        |     1 |    65 |     2   (0)| 00:00:01 |
|*  6 |       INDEX RANGE SCAN          | CL_NAME      |     1 |       |     1   (0)| 00:00:01 |
|*  7 |      TABLE ACCESS FULL          | REGISTRATION | 97637 |  1716K|  5489   (1)| 00:01:06 |
|*  8 |     INDEX UNIQUE SCAN           | PK_STUDENT   |     1 |       |     0   (0)|          |
|   9 |    TABLE ACCESS BY INDEX ROWID  | STUDENT      |     1 |    34 |     1   (0)| 00:00:01 |
------------------------------------------------------------------------------------------------

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

1 - filter(TO_DATE(:BEG_DATE)<=TO_DATE(:END_DATE))
4 - access("R"."CLASS_ID"="C"."CLASS_ID")
6 - access("C"."NAME"='SQL TUNING')
7 - filter(("R"."SIGNUP_DATE"<=:END_DATE AND "R"."SIGNUP_DATE">=:BEG_DATE AND "R"."CANCELLED"='N'))
8 - access("R"."STUDENT_ID"="S"."STUDENT_ID")
```

- Execution Stats – 20,348 buffer gets
- Why is a full table scan still occurring on REGISTRATION?

© 2015 SOLARWINDS WORLDWIDE, LLC. ALL RIGHTS RESERVED.

REVIEW INDEX ORDER

- CLASS_ID not left leading in index

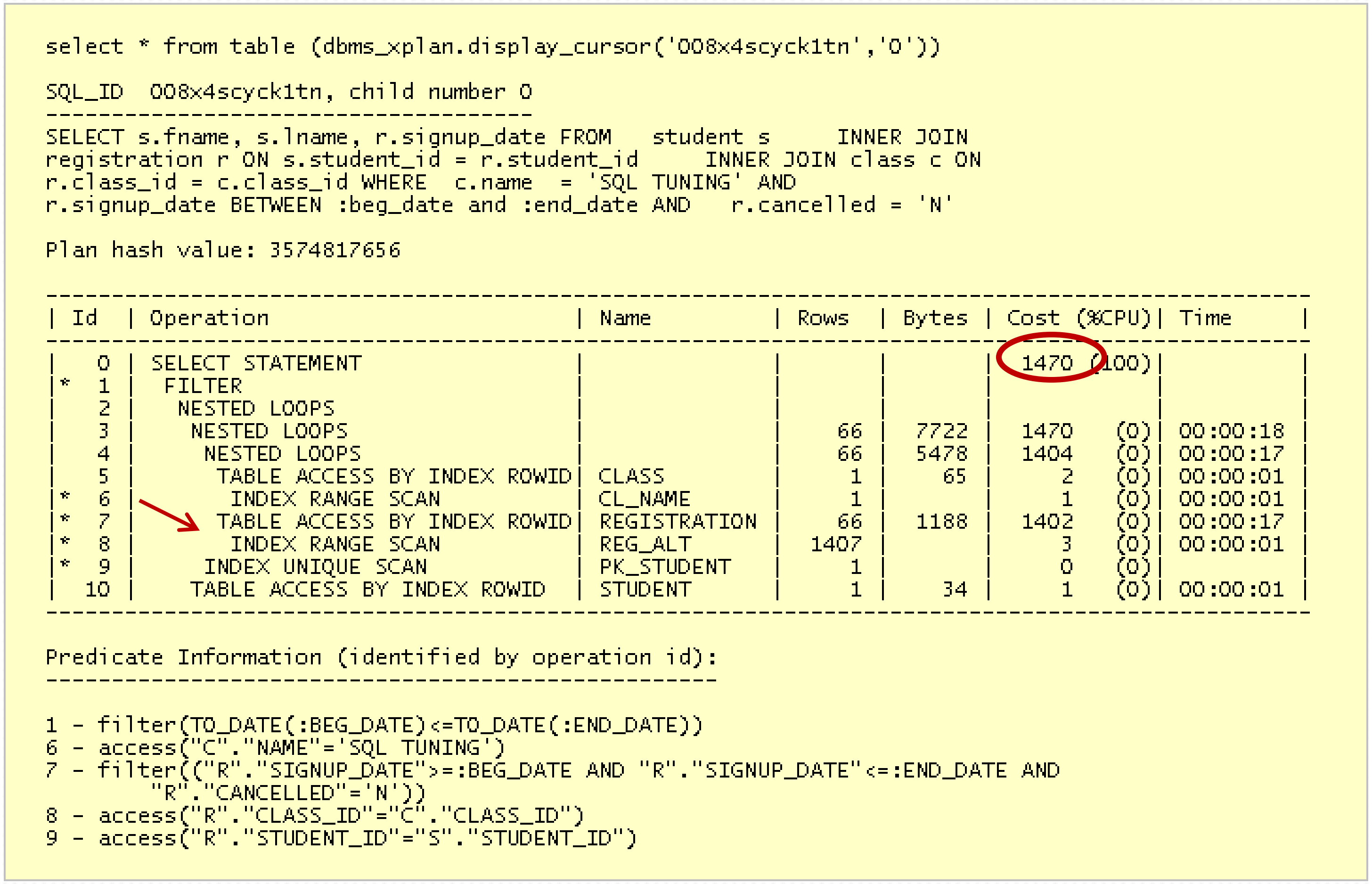


- Execution Stats – 20,348 buffer gets
- Twice the work to use Primary Key Index on REGISTRATION

© 2015 SOLARWINDS WORLDWIDE, LLC. ALL RIGHTS RESERVED.

NEW EXECUTION PLAN

» CREATE INDEX reg_alt ON registration(class_id);



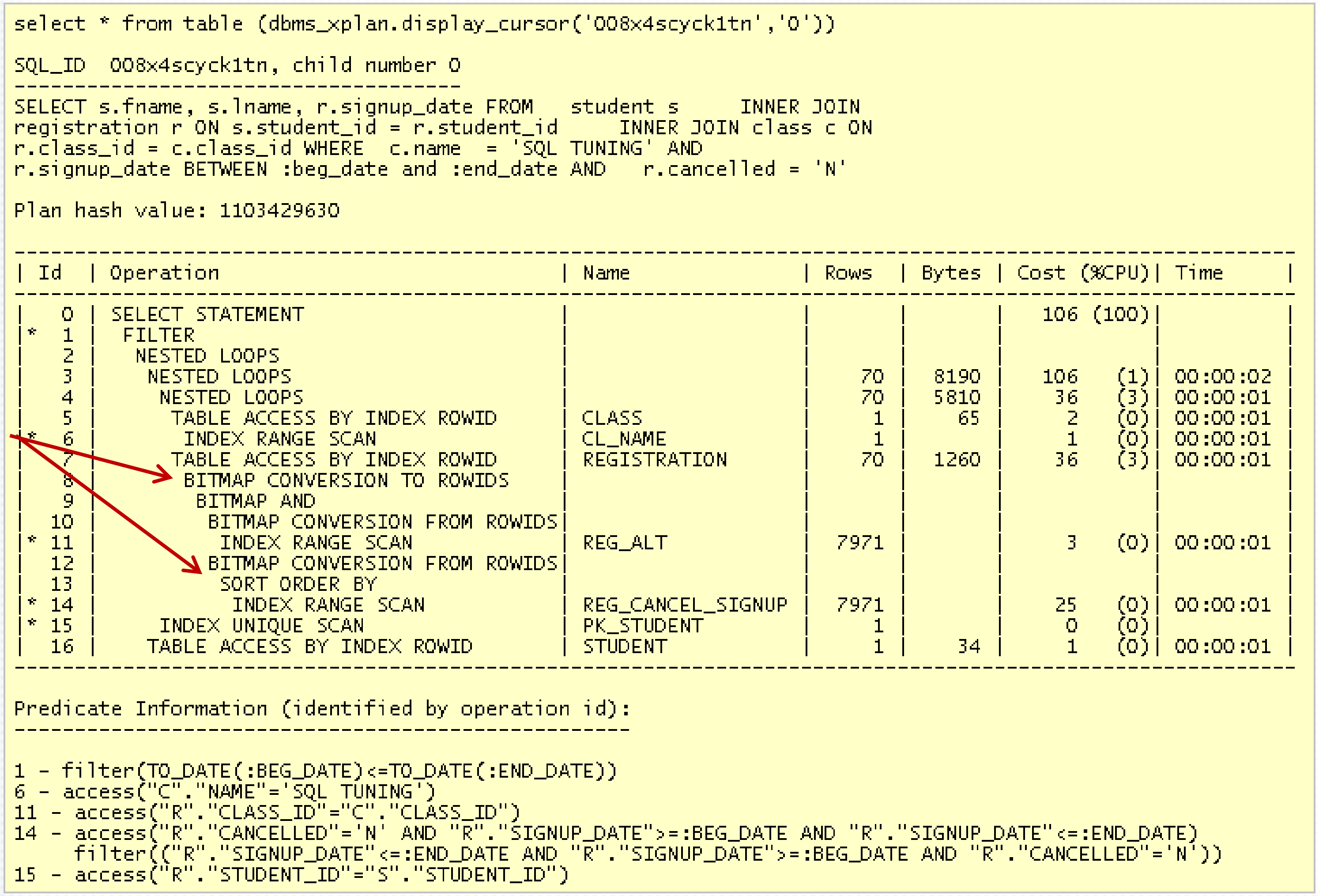
```
select * from table (dbms_xplan.display_cursor('008x4scyck1tn','0'))

SQL_ID  008x4scyck1tn, child number 0
-------------------------------------
SELECT s.fname, s.lname, r.signup_date FROM   student s     INNER JOIN
registration r ON s.student_id = r.student_id     INNER JOIN class c ON
r.class_id = c.class_id WHERE  c.name  = 'SQL TUNING' AND
r.signup_date BETWEEN :beg_date and :end_date AND   r.cancelled = 'N'

Plan hash value: 3574817656

------------------------------------------------------------------------------------------------
| Id  | Operation                          | Name           | Rows  | Bytes | Cost (%CPU)| Time     |
------------------------------------------------------------------------------------------------
|   0 | SELECT STATEMENT                   |                |       |       |  1470 (100)|          |
|*  1 |  FILTER                            |                |       |       |            |          |
|   2 |   NESTED LOOPS                     |                |       |       |            |          |
|   3 |    NESTED LOOPS                    |                |    66 |  7722 |  1470   (0)| 00:00:18 |
|   4 |     NESTED LOOPS                   |                |    66 |  5478 |  1404   (0)| 00:00:17 |
|   5 |      TABLE ACCESS BY INDEX ROWID| CLASS          |     1 |    65 |     2   (0)| 00:00:01 |
|*  6 |       INDEX RANGE SCAN             | CL_NAME        |     1 |       |     1   (0)| 00:00:01 |
|*  7 |      TABLE ACCESS BY INDEX ROWID| REGISTRATION   |    66 |  1188 |  1402   (0)| 00:00:17 |
|*  8 |       INDEX RANGE SCAN             | REG_ALT        |  1407 |       |     3   (0)| 00:00:01 |
|*  9 |     INDEX UNIQUE SCAN              | PK_STUDENT     |     1 |       |     0   (0)|          |
|  10 |    TABLE ACCESS BY INDEX ROWID     | STUDENT        |     1 |    34 |     1   (0)| 00:00:01 |
------------------------------------------------------------------------------------------------

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

1 - filter(TO_DATE(:BEG_DATE)<=TO_DATE(:END_DATE))
6 - access("C"."NAME"='SQL TUNING')
7 - filter(("R"."SIGNUP_DATE">=:BEG_DATE AND "R"."SIGNUP_DATE"<=:END_DATE AND
        "R"."CANCELLED"='N'))
8 - access("R"."CLASS_ID"="C"."CLASS_ID")
9 - access("R"."STUDENT_ID"="S"."STUDENT_ID")
```

» Execution Stats – 3000 Buffer Gets / Average Execs - .008 Secs

© 2015 SOLARWINDS WORLDWIDE, LLC. ALL RIGHTS RESERVED.

TUNING ADVISOR SUGGESTED INDEX

- CREATE INDEX reg_cancel_signup ON registration(cancelled,signup_date);



```
select * from table (dbms_xplan.display_cursor('008x4scyck1tn','0'))

SQL_ID  008x4scyck1tn, child number 0
-------------------------------------
SELECT s.fname, s.lname, r.signup_date FROM   student s     INNER JOIN
registration r ON s.student_id = r.student_id     INNER JOIN class c ON
r.class_id = c.class_id WHERE  c.name  = 'SQL TUNING' AND
r.signup_date BETWEEN :beg_date and :end_date AND   r.cancelled = 'N'

Plan hash value: 1103429630

----------------------------------------------------------------------------------------------------------
| Id  | Operation                             | Name               | Rows  | Bytes | Cost (%CPU)| Time     |
----------------------------------------------------------------------------------------------------------
|   0 | SELECT STATEMENT                      |                    |       |       |   106 (100)|          |
|*  1 |  FILTER                               |                    |       |       |            |          |
|   2 |   NESTED LOOPS                        |                    |       |       |            |          |
|   3 |    NESTED LOOPS                       |                    |    70 |  8190 |   106   (1)| 00:00:02 |
|   4 |     NESTED LOOPS                      |                    |    70 |  5810 |    36   (3)| 00:00:01 |
|   5 |      TABLE ACCESS BY INDEX ROWID      | CLASS              |     1 |    65 |     2   (0)| 00:00:01 |
|*  6 |       INDEX RANGE SCAN                | CL_NAME            |     1 |       |     1   (0)| 00:00:01 |
|   7 |      TABLE ACCESS BY INDEX ROWID      | REGISTRATION       |    70 |  1260 |    36   (3)| 00:00:01 |
|   8 |       BITMAP CONVERSION TO ROWIDS     |                    |       |       |            |          |
|   9 |        BITMAP AND                     |                    |       |       |            |          |
|  10 |         BITMAP CONVERSION FROM ROWIDS|                    |       |       |            |          |
|* 11 |          INDEX RANGE SCAN             | REG_ALT            |  7971 |       |     3   (0)| 00:00:01 |
|  12 |         BITMAP CONVERSION FROM ROWIDS|                    |       |       |            |          |
|  13 |          SORT ORDER BY                |                    |       |       |            |          |
|* 14 |           INDEX RANGE SCAN            | REG_CANCEL_SIGNUP  |  7971 |       |    25   (0)| 00:00:01 |
|* 15 |     INDEX UNIQUE SCAN                 | PK_STUDENT         |     1 |       |     0   (0)|          |
|  16 |    TABLE ACCESS BY INDEX ROWID        | STUDENT            |     1 |    34 |     1   (0)| 00:00:01 |
----------------------------------------------------------------------------------------------------------

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

1 - filter(TO_DATE(:BEG_DATE)<=TO_DATE(:END_DATE))
6 - access("C"."NAME"='SQL TUNING')
11 - access("R"."CLASS_ID"="C"."CLASS_ID")
14 - access("R"."CANCELLED"='N' AND "R"."SIGNUP_DATE">=:BEG_DATE AND "R"."SIGNUP_DATE"<=:END_DATE)
     filter(("R"."SIGNUP_DATE"<=:END_DATE AND "R"."SIGNUP_DATE">=:BEG_DATE AND "R"."CANCELLED"='N'))
15 - access("R"."STUDENT_ID"="S"."STUDENT_ID")
```

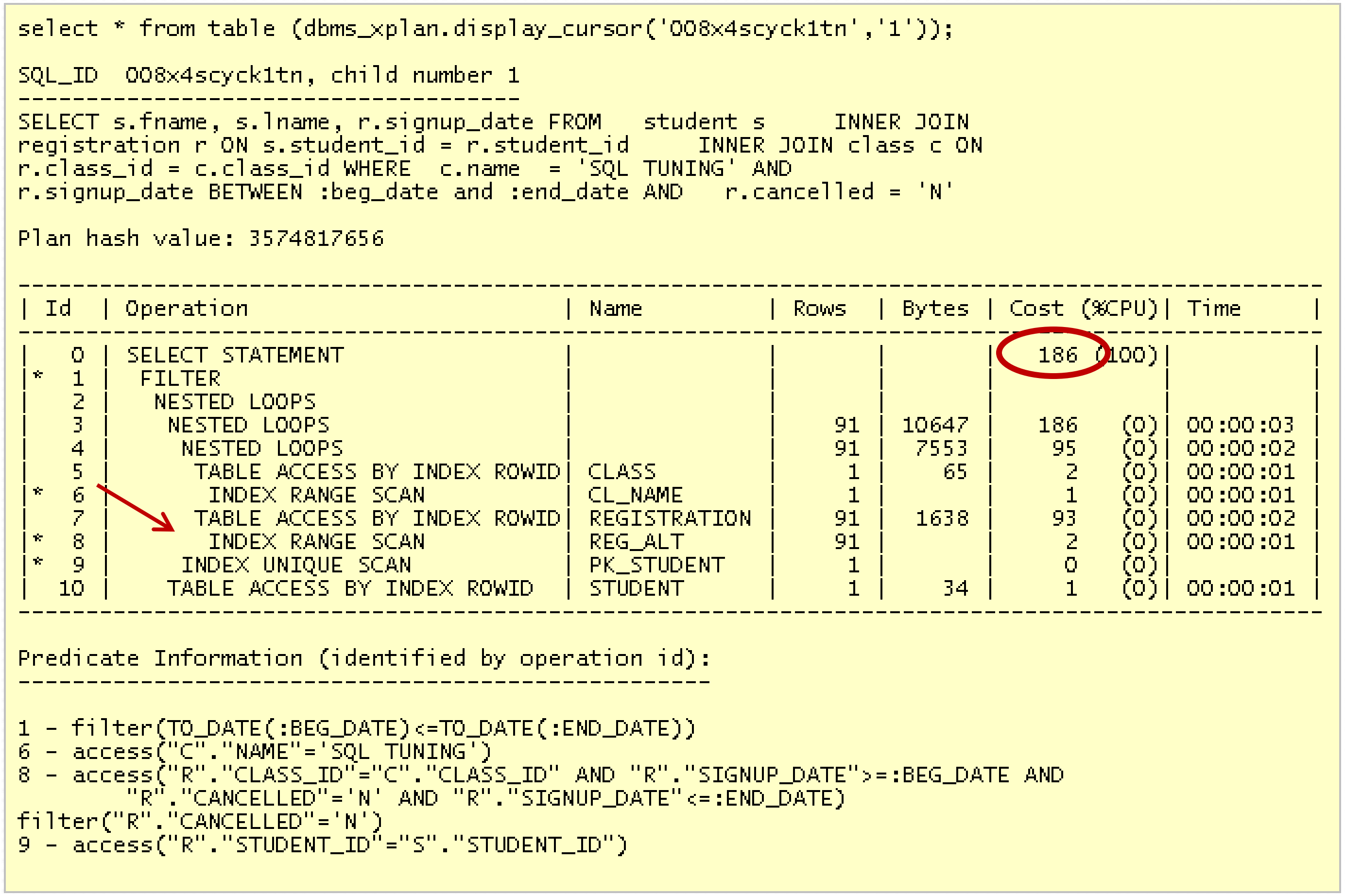
Execution Stats:
1107 Buffer Gets

Avg Executions:
0.14 Secs

© 2015 SOLARWINDS WORLDWIDE, LLC. ALL RIGHTS RESERVED.

BETTER EXECUTION PLAN

CREATE INDEX reg_alt ON registration(class_id,signup_date, cancelled);



```
select * from table (dbms_xplan.display_cursor('008x4scyck1tn','1'));

SQL_ID  008x4scyck1tn, child number 1
-------------------------------------
SELECT s.fname, s.lname, r.signup_date FROM   student s     INNER JOIN
registration r ON s.student_id = r.student_id     INNER JOIN class c ON
r.class_id = c.class_id WHERE  c.name  = 'SQL TUNING' AND
r.signup_date BETWEEN :beg_date and :end_date AND   r.cancelled = 'N'

Plan hash value: 3574817656

-------------------------------------------------------------------------------------------------
| Id  | Operation                          | Name          | Rows   | Bytes  | Cost (%CPU)| Time     |
-------------------------------------------------------------------------------------------------
|   0 | SELECT STATEMENT                   |               |        |        |   186 (100)|          |
|*  1 |  FILTER                            |               |        |        |            |          |
|   2 |   NESTED LOOPS                     |               |        |        |            |          |
|   3 |    NESTED LOOPS                    |               |     91 |  10647 |   186   (0)| 00:00:03 |
|   4 |     NESTED LOOPS                   |               |     91 |   7553 |    95   (0)| 00:00:02 |
|   5 |      TABLE ACCESS BY INDEX ROWID| CLASS            |      1 |     65 |     2   (0)| 00:00:01 |
|*  6 |       INDEX RANGE SCAN             | CL_NAME       |      1 |        |     1   (0)| 00:00:01 |
|   7 |      TABLE ACCESS BY INDEX ROWID| REGISTRATION     |     91 |   1638 |    93   (0)| 00:00:02 |
|*  8 |       INDEX RANGE SCAN             | REG_ALT       |     91 |        |     2   (0)| 00:00:01 |
|*  9 |     INDEX UNIQUE SCAN              | PK_STUDENT    |      1 |        |     0   (0)|          |
|  10 |    TABLE ACCESS BY INDEX ROWID     | STUDENT       |      1 |     34 |     1   (0)| 00:00:01 |
-------------------------------------------------------------------------------------------------

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

1 - filter(TO_DATE(:BEG_DATE)<=TO_DATE(:END_DATE))
6 - access("C"."NAME"='SQL TUNING')
8 - access("R"."CLASS_ID"="C"."CLASS_ID" AND "R"."SIGNUP_DATE">=:BEG_DATE AND
        "R"."CANCELLED"='N' AND "R"."SIGNUP_DATE"<=:END_DATE)
filter("R"."CANCELLED"='N')
9 - access("R"."STUDENT_ID"="S"."STUDENT_ID")
```

- Execution Stats – 445 Buffer Gets / Average Execs - .002 Secs

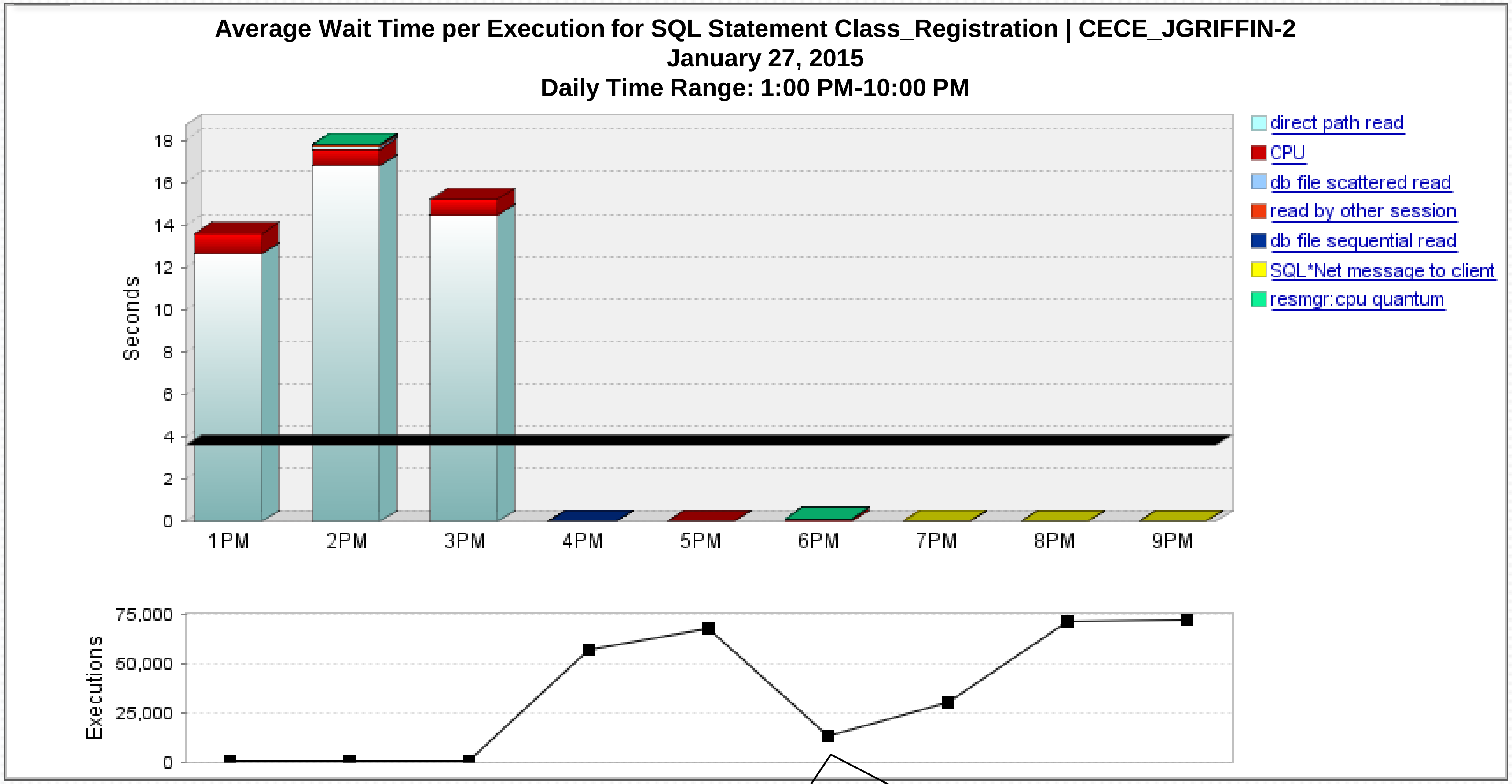
© 2015 SOLARWINDS WORLDWIDE, LLC. ALL RIGHTS RESERVED.

12. MONITOR YOUR TUNING RESULTS

- Monitor the improvement
  - Be able to prove that tuning made a difference
  - Take new metric measurements
  - Compare them to initial readings
  - Brag about the improvements – no one else will
- Monitor for next tuning opportunity
  - Tuning is iterative
  - There is always room for improvement
  - Make sure you tune things that make a difference
- Shameless Product Pitch - DPA

© 2015 SOLARWINDS WORLDWIDE, LLC. ALL RIGHTS RESERVED.

PERFORMANCE IMPROVED?



Average Wait Time per Execution for SQL Statement Class_Registration | CECE_JGRIFFIN-2
January 27, 2015
Daily Time Range: 1:00 PM-10:00 PM

reg_canceled_signup index

© 2015 SOLARWINDS WORLDWIDE, LLC. ALL RIGHTS RESERVED.

CASE STUDY 2

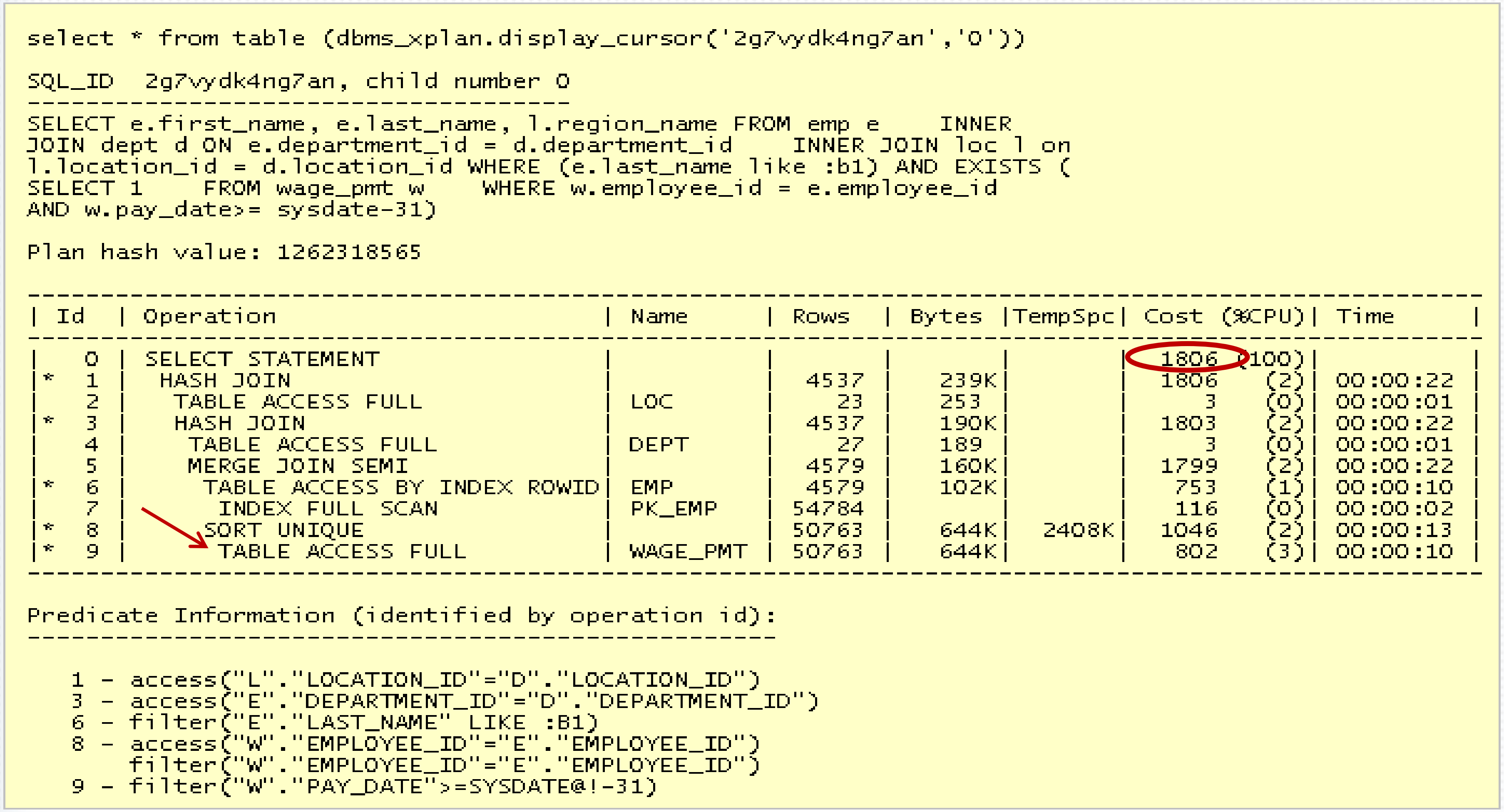
- Current paychecks for specific employees

```
SELECT e.first_name, e.last_name, l.region_name
FROM emp e
  INNER JOIN dept d ON e.department_id = d.department_id
  INNER JOIN loc l on l.location_id = d.location_id
WHERE (e.last_name like :b1)
AND EXISTS (
  SELECT 1
  FROM wage_pmt w
  WHERE w.employee_id = e.employee_id
  AND w.pay_date>= sysdate-31);
```

- **Execution Stats - 3,890 Buffer Gets**
- **Average Execution - .31 seconds**
- **Resource - 99% CPU**

© 2015 SOLARWINDS WORLDWIDE, LLC. ALL RIGHTS RESERVED.

EXECUTION PLAN

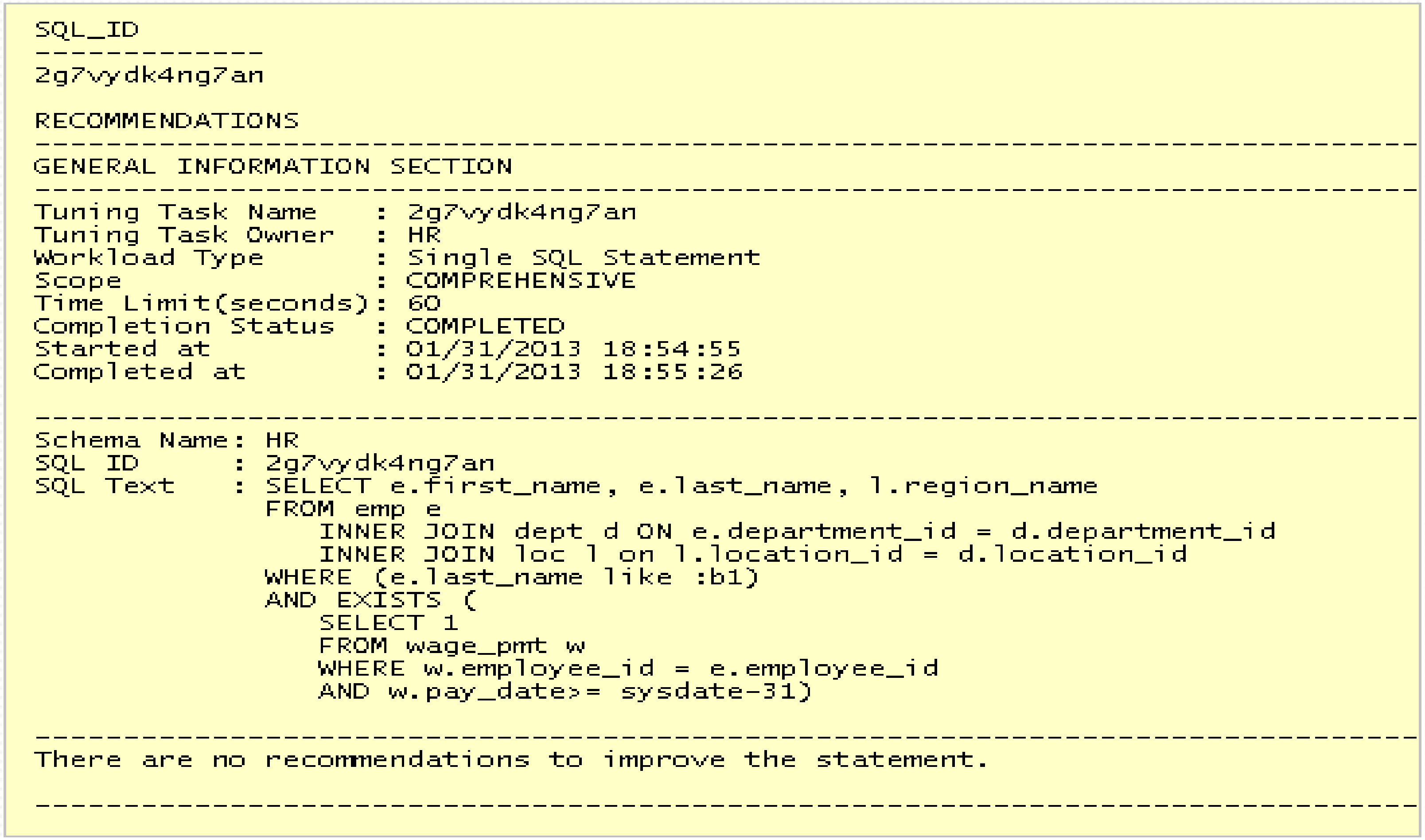


```
select * from table (dbms_xplan.display_cursor('2g7vydk4ng7an','0'))

SQL_ID  2g7vydk4ng7an, child number 0
-------------------------------------
SELECT e.first_name, e.last_name, l.region_name FROM emp e    INNER
JOIN dept d ON e.department_id = d.department_id    INNER JOIN loc l on
l.location_id = d.location_id WHERE (e.last_name like :b1) AND EXISTS (
SELECT 1    FROM wage_pmt w    WHERE w.employee_id = e.employee_id
AND w.pay_date>= sysdate-31)

Plan hash value: 1262318565

-----------------------------------------------------------------------------------------------------
| Id  | Operation                       | Name       | Rows  | Bytes |TempSpc| Cost (%CPU)| Time     |
-----------------------------------------------------------------------------------------------------
|   0 | SELECT STATEMENT                |            |       |       |       |  1806 (100)|          |
|*  1 |  HASH JOIN                      |            |  4537 |  239K|       |  1806   (2)| 00:00:22 |
|   2 |   TABLE ACCESS FULL             | LOC        |    23 |  253 |       |     3   (0)| 00:00:01 |
|*  3 |   HASH JOIN                     |            |  4537 |  190K|       |  1803   (2)| 00:00:22 |
|   4 |    TABLE ACCESS FULL            | DEPT       |    27 |  189 |       |     3   (0)| 00:00:01 |
|   5 |    MERGE JOIN SEMI              |            |  4579 |  160K|       |  1799   (2)| 00:00:22 |
|*  6 |     TABLE ACCESS BY INDEX ROWID| EMP        |  4579 |  102K|       |   753   (1)| 00:00:10 |
|   7 |      INDEX FULL SCAN            | PK_EMP     | 54784 |       |       |   116   (0)| 00:00:02 |
|*  8 |     SORT UNIQUE                 |            | 50763 |  644K|  2408K|  1046   (2)| 00:00:13 |
|*  9 |      TABLE ACCESS FULL          | WAGE_PMT   | 50763 |  644K|       |   802   (3)| 00:00:10 |
-----------------------------------------------------------------------------------------------------

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

   1 - access("L"."LOCATION_ID"="D"."LOCATION_ID")
   3 - access("E"."DEPARTMENT_ID"="D"."DEPARTMENT_ID")
   6 - filter("E"."LAST_NAME" LIKE :B1)
   8 - access("W"."EMPLOYEE_ID"="E"."EMPLOYEE_ID")
       filter("W"."EMPLOYEE_ID"="E"."EMPLOYEE_ID")
   9 - filter("W"."PAY_DATE">=SYSDATE@!-31)
```

© 2015 SOLARWINDS WORLDWIDE, LLC. ALL RIGHTS RESERVED.

TUNING ADVISOR

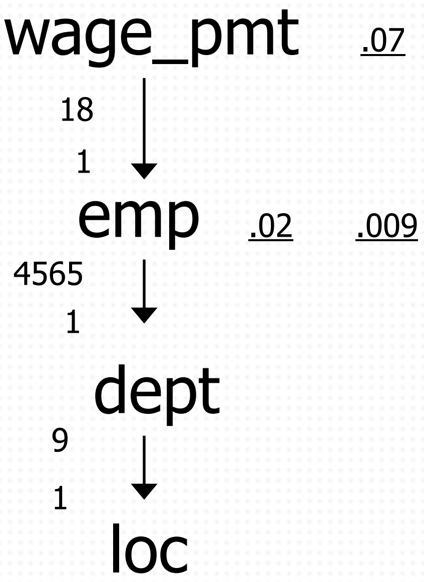
- **No recommendations?**



```
SQL_ID
-------------
2g7vydk4ng7an

RECOMMENDATIONS
-------------------------------------------------------------------------------
GENERAL INFORMATION SECTION
-------------------------------------------------------------------------------
Tuning Task Name    : 2g7vydk4ng7an
Tuning Task Owner   : HR
Workload Type       : Single SQL Statement
Scope               : COMPREHENSIVE
Time Limit(seconds): 60
Completion Status   : COMPLETED
Started at          : 01/31/2013 18:54:55
Completed at        : 01/31/2013 18:55:26

-------------------------------------------------------------------------------
Schema Name: HR
SQL ID     : 2g7vydk4ng7an
SQL Text   : SELECT e.first_name, e.last_name, l.region_name
             FROM emp e
                INNER JOIN dept d ON e.department_id = d.department_id
                INNER JOIN loc l on l.location_id = d.location_id
             WHERE (e.last_name like :b1)
             AND EXISTS (
                SELECT 1
                FROM wage_pmt w
                WHERE w.employee_id = e.employee_id
                AND w.pay_date>= sysdate-31)

-------------------------------------------------------------------------------
There are no recommendations to improve the statement.

-------------------------------------------------------------------------------
```

© 2015 SOLARWINDS WORLDWIDE, LLC. ALL RIGHTS RESERVED.

SQL DIAGRAMMING



wage_pmt .07

18

1

emp .02 .009

4565

1

dept

9

1

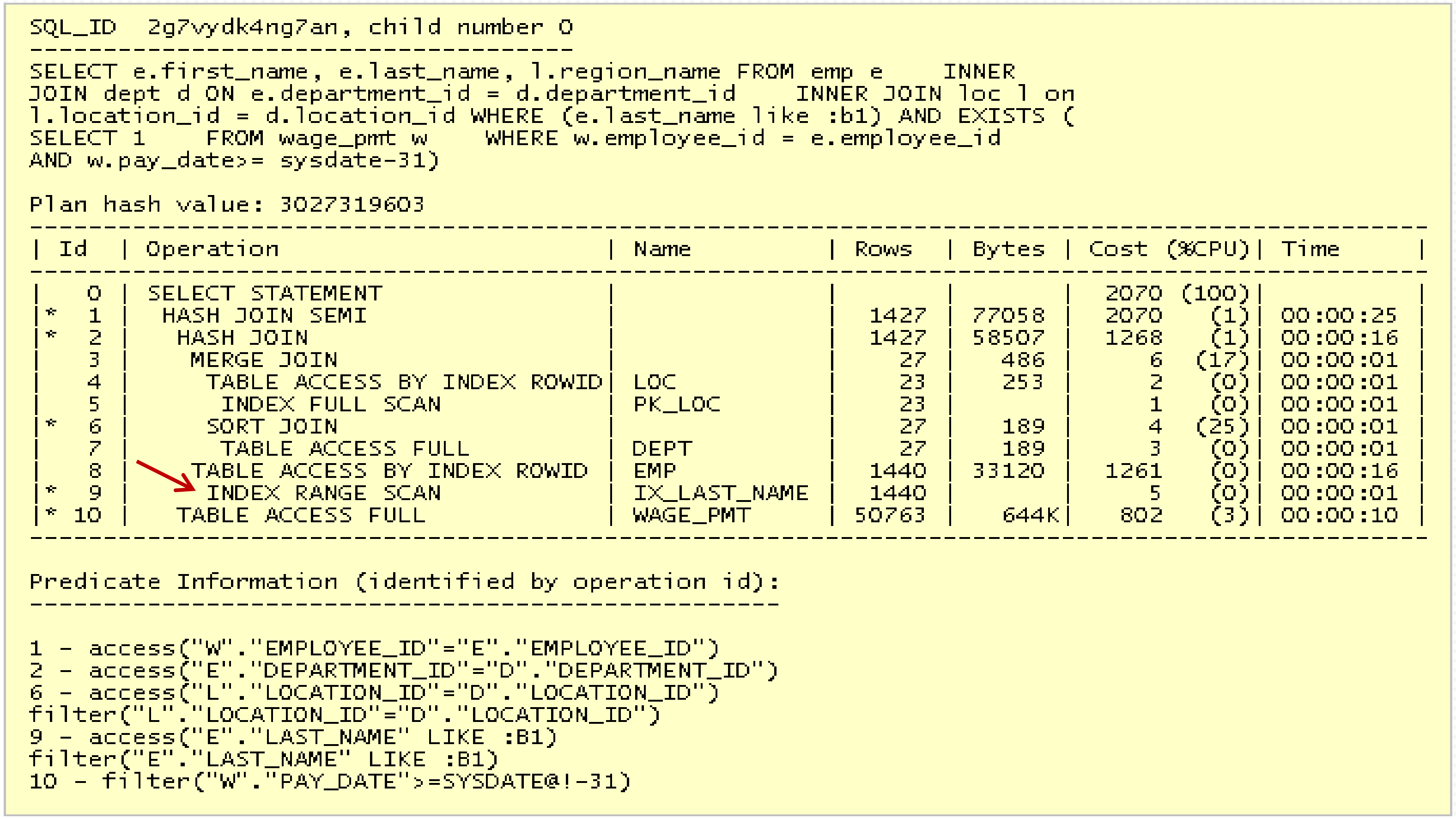
loc

```
select count(1) from wage_pmt
where pay_date >= sysdate - 31

54,784 / 821,760 = .066

select max(cnt), min(cnt)
from (select last_name, count(1) cnt from emp group by last_name)

1,024 / 54,784 = .018 – max
512   /  54,784 = .009 – min
```

© 2015 SOLARWINDS WORLDWIDE, LLC. ALL RIGHTS RESERVED.

NEW EXECUTION PLAN

» CREATE INDEX ix_last_name ON emp(last_name);



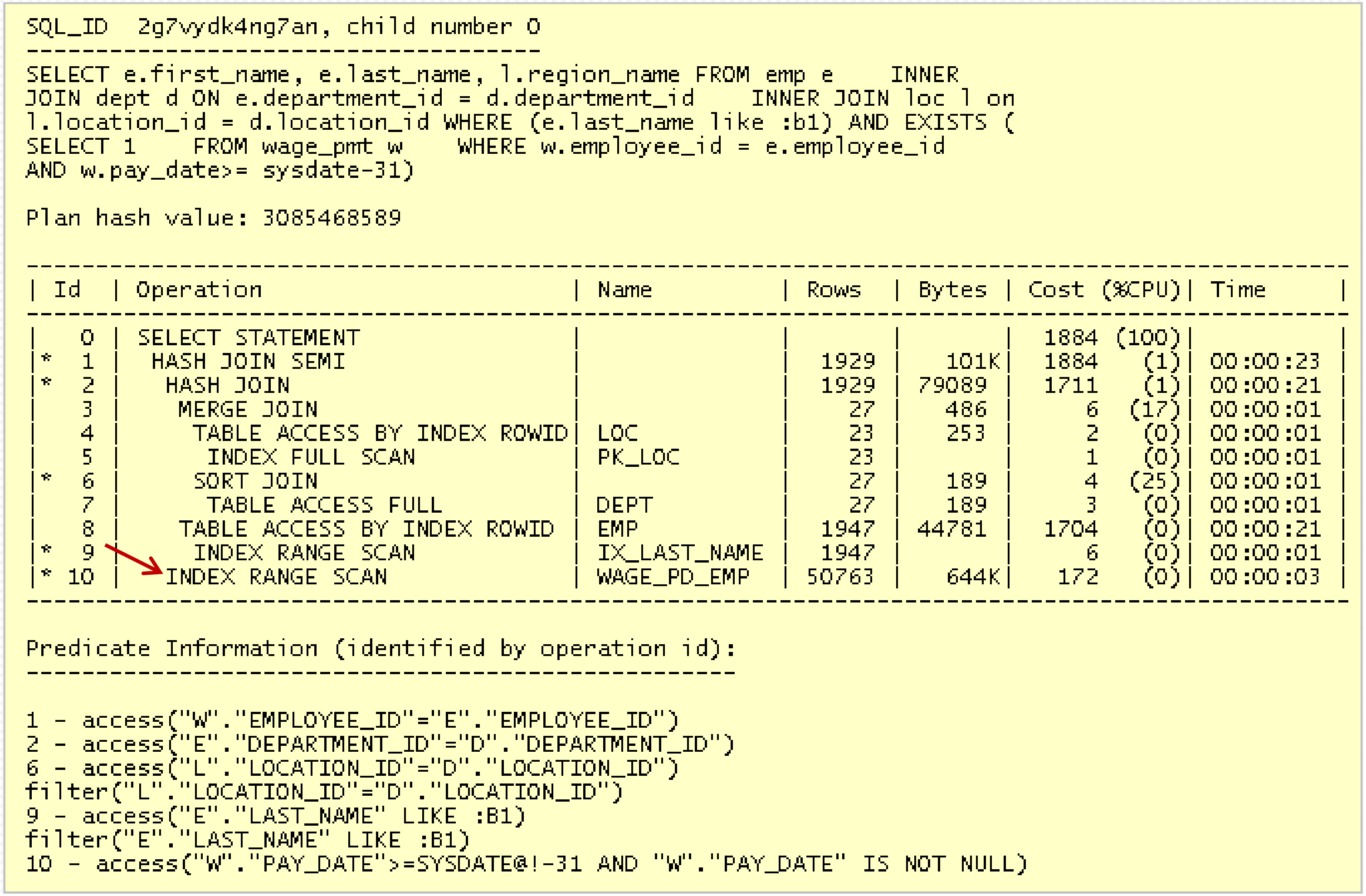
```
SQL_ID  2g7vydk4ng7an, child number 0
-------------------------------------
SELECT e.first_name, e.last_name, l.region_name FROM emp e    INNER
JOIN dept d ON e.department_id = d.department_id    INNER JOIN loc l on
l.location_id = d.location_id WHERE (e.last_name like :b1) AND EXISTS (
SELECT 1    FROM wage_pmt w    WHERE w.employee_id = e.employee_id
AND w.pay_date>= sysdate-31)

Plan hash value: 3027319603

-----------------------------------------------------------------------------------------------
| Id  | Operation                     | Name         | Rows  | Bytes | Cost (%CPU)| Time     |
-----------------------------------------------------------------------------------------------
|   0 | SELECT STATEMENT              |              |       |       |  2070 (100)|          |
|*  1 |  HASH JOIN SEMI               |              |  1427 | 77058 |  2070   (1)| 00:00:25 |
|*  2 |   HASH JOIN                   |              |  1427 | 58507 |  1268   (1)| 00:00:16 |
|   3 |    MERGE JOIN                 |              |    27 |   486 |     6  (17)| 00:00:01 |
|   4 |     TABLE ACCESS BY INDEX ROWID| LOC          |    23 |   253 |     2   (0)| 00:00:01 |
|   5 |      INDEX FULL SCAN          | PK_LOC       |    23 |       |     1   (0)| 00:00:01 |
|*  6 |     SORT JOIN                 |              |    27 |   189 |     4  (25)| 00:00:01 |
|   7 |      TABLE ACCESS FULL        | DEPT         |    27 |   189 |     3   (0)| 00:00:01 |
|   8 |    TABLE ACCESS BY INDEX ROWID | EMP          |  1440 | 33120 |  1261   (0)| 00:00:16 |
|*  9 |     INDEX RANGE SCAN          | IX_LAST_NAME |  1440 |       |     5   (0)| 00:00:01 |
|* 10 |   TABLE ACCESS FULL           | WAGE_PMT     | 50763 |   644K|   802   (3)| 00:00:10 |
-----------------------------------------------------------------------------------------------

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

1 - access("W"."EMPLOYEE_ID"="E"."EMPLOYEE_ID")
2 - access("E"."DEPARTMENT_ID"="D"."DEPARTMENT_ID")
6 - access("L"."LOCATION_ID"="D"."LOCATION_ID")
filter("L"."LOCATION_ID"="D"."LOCATION_ID")
9 - access("E"."LAST_NAME" LIKE :B1)
filter("E"."LAST_NAME" LIKE :B1)
10 - filter("W"."PAY_DATE">=SYSDATE@!-31)
```

» **Execution Stats – 1105 Buffer Gets / Average Execs - .06 Secs**

© 2015 SOLARWINDS WORLDWIDE, LLC. ALL RIGHTS RESERVED.

NEW EXECUTION PLAN

» CREATE INDEX wp_pd_emp ON wage_pmt(employee_id,pay_date);

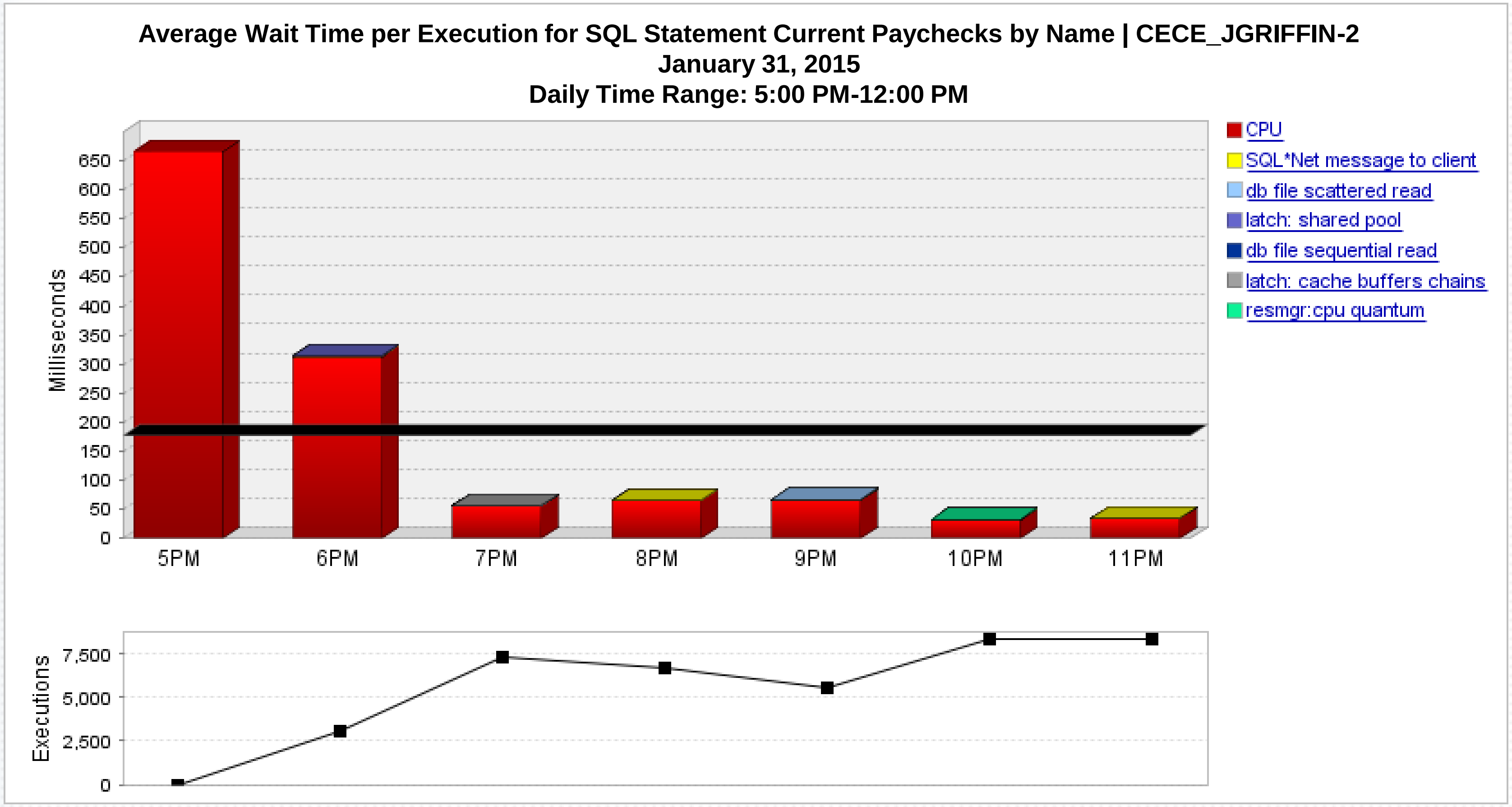


```
SQL_ID  2g7vydk4ng7an, child number 0
-------------------------------------
SELECT e.first_name, e.last_name, l.region_name FROM emp e    INNER
JOIN dept d ON e.department_id = d.department_id    INNER JOIN loc l on
l.location_id = d.location_id WHERE (e.last_name like :b1) AND EXISTS (
SELECT 1    FROM wage_pmt w    WHERE w.employee_id = e.employee_id
AND w.pay_date>= sysdate-31)

Plan hash value: 3085468589

-----------------------------------------------------------------------------------------------
| Id  | Operation                     | Name          | Rows  | Bytes | Cost (%CPU)| Time     |
-----------------------------------------------------------------------------------------------
|   0 | SELECT STATEMENT              |               |       |       |  1884 (100)|          |
|*  1 |  HASH JOIN SEMI               |               |  1929 |   101K|  1884   (1)| 00:00:23 |
|*  2 |   HASH JOIN                   |               |  1929 | 79089 |  1711   (1)| 00:00:21 |
|   3 |    MERGE JOIN                 |               |    27 |   486 |     6  (17)| 00:00:01 |
|   4 |     TABLE ACCESS BY INDEX ROWID| LOC           |    23 |   253 |     2   (0)| 00:00:01 |
|   5 |      INDEX FULL SCAN          | PK_LOC        |    23 |       |     1   (0)| 00:00:01 |
|*  6 |     SORT JOIN                 |               |    27 |   189 |     4  (25)| 00:00:01 |
|   7 |      TABLE ACCESS FULL        | DEPT          |    27 |   189 |     3   (0)| 00:00:01 |
|   8 |    TABLE ACCESS BY INDEX ROWID | EMP           |  1947 | 44781 |  1704   (0)| 00:00:21 |
|*  9 |     INDEX RANGE SCAN          | IX_LAST_NAME  |  1947 |       |     6   (0)| 00:00:01 |
|* 10 |   INDEX RANGE SCAN            | WAGE_PD_EMP   | 50763 |   644K|   172   (0)| 00:00:03 |
-----------------------------------------------------------------------------------------------

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

1 - access("W"."EMPLOYEE_ID"="E"."EMPLOYEE_ID")
2 - access("E"."DEPARTMENT_ID"="D"."DEPARTMENT_ID")
6 - access("L"."LOCATION_ID"="D"."LOCATION_ID")
filter("L"."LOCATION_ID"="D"."LOCATION_ID")
9 - access("E"."LAST_NAME" LIKE :B1)
filter("E"."LAST_NAME" LIKE :B1)
10 - access("W"."PAY_DATE">=SYSDATE@!-31 AND "W"."PAY_DATE" IS NOT NULL)
```

- **Execution Stats – 695 Buffer Gets / Average Execs - .03 Secs**

© 2015 SOLARWINDS WORLDWIDE, LLC. ALL RIGHTS RESERVED.

IMPROVED PERFORMANCE?


Average Wait Time per Execution for SQL Statement Current Paychecks by Name | CECE_JGRIFFIN-2
January 31, 2015
Daily Time Range: 5:00 PM-12:00 PM

- Execution Stats – 695 Buffer Gets / Average Execs - .03 Secs

© 2015 SOLARWINDS WORLDWIDE, LLC. ALL RIGHTS RESERVED.

CASE STUDY 3

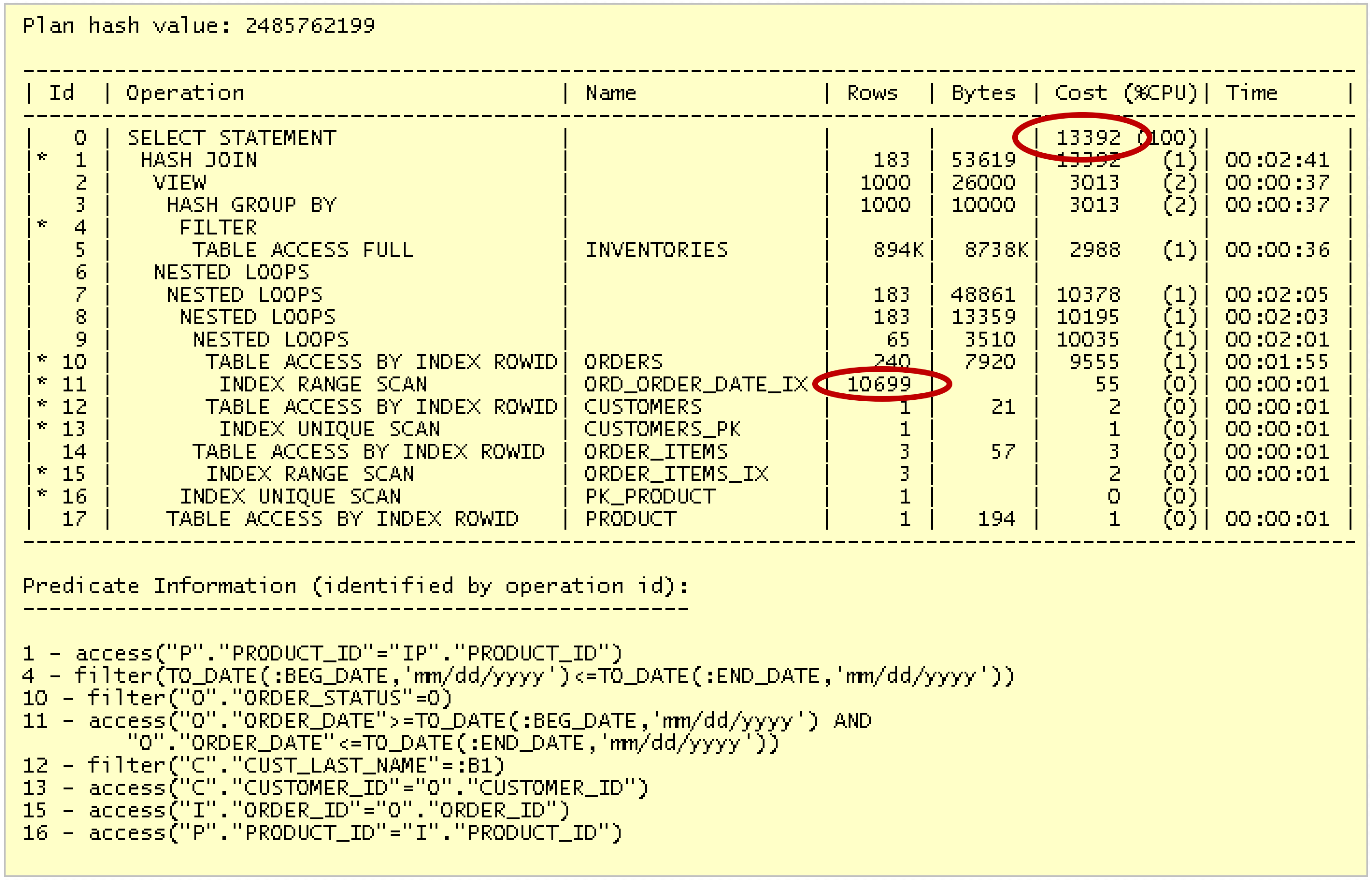
» Inventory lookup for New Orders by Customer

```
SELECT c.cust_first_name, c.cust_last_name, o.order_date, o.order_status,
               o.order_mode, i.line_item_id, p.product_description,
               i.unit_price * i.quantity total_price, quantity quantity_ordered, ip.total_on_hand
FROM orders o, order_Items i, customers c, product p,
          (SELECT product_id, sum(quantity_on_hand) total_on_hand
           FROM inventories
          GROUP BY product_id) ip
WHERE i.order_id = o.order_id AND c.customer_id = o.customer_id
AND p.product_id = i.product_id AND p.product_id = ip.product_id
AND c.cust_last_name = :B1
AND o.order_status = 0
AND o.order_date BETWEEN to_date(:BEG_DATE,'mm/dd/yyyy')
                                    AND to_date(:END_DATE,'mm/dd/yyyy')
```

» **Execution Stats: 73,392 Buffer Gets**

© 2015 SOLARWINDS WORLDWIDE, LLC. ALL RIGHTS RESERVED.

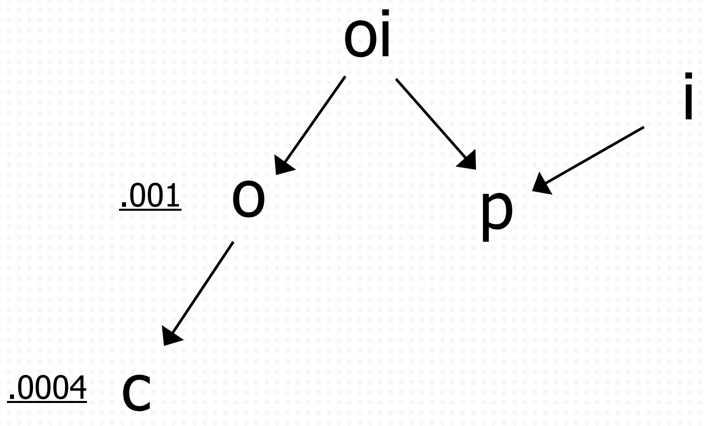
EXECUTION PLAN



```
Plan hash value: 2485762199

-----------------------------------------------------------------------------------------------------
| Id  | Operation                      | Name               | Rows  | Bytes | Cost (%CPU)| Time     |
-----------------------------------------------------------------------------------------------------
|   0 | SELECT STATEMENT               |                    |       |       | 13392 (100)|          |
|*  1 |  HASH JOIN                     |                    |   183 | 53619 | 13392   (1)| 00:02:41 |
|   2 |   VIEW                         |                    |  1000 | 26000 |  3013   (2)| 00:00:37 |
|   3 |    HASH GROUP BY               |                    |  1000 | 10000 |  3013   (2)| 00:00:37 |
|*  4 |     FILTER                     |                    |       |       |            |          |
|   5 |      TABLE ACCESS FULL         | INVENTORIES        |   894K|  8738K|  2988   (1)| 00:00:36 |
|   6 |   NESTED LOOPS                 |                    |       |       |            |          |
|   7 |    NESTED LOOPS                |                    |   183 | 48861 | 10378   (1)| 00:02:05 |
|   8 |     NESTED LOOPS               |                    |   183 | 13359 | 10195   (1)| 00:02:03 |
|   9 |      NESTED LOOPS              |                    |    65 |  3510 | 10035   (1)| 00:02:01 |
|* 10 |       TABLE ACCESS BY INDEX ROWID| ORDERS           |   240 |  7920 |  9555   (1)| 00:01:55 |
|* 11 |        INDEX RANGE SCAN        | ORD_ORDER_DATE_IX  | 10699 |       |    55   (0)| 00:00:01 |
|* 12 |       TABLE ACCESS BY INDEX ROWID| CUSTOMERS        |     1 |    21 |     2   (0)| 00:00:01 |
|* 13 |        INDEX UNIQUE SCAN       | CUSTOMERS_PK       |     1 |       |     1   (0)| 00:00:01 |
|  14 |      TABLE ACCESS BY INDEX ROWID | ORDER_ITEMS      |     3 |    57 |     3   (0)| 00:00:01 |
|* 15 |       INDEX RANGE SCAN         | ORDER_ITEMS_IX     |     3 |       |     2   (0)| 00:00:01 |
|* 16 |     INDEX UNIQUE SCAN          | PK_PRODUCT         |     1 |       |     0   (0)|          |
|  17 |    TABLE ACCESS BY INDEX ROWID | PRODUCT            |     1 |   194 |     1   (0)| 00:00:01 |
-----------------------------------------------------------------------------------------------------

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

1 - access("P"."PRODUCT_ID"="IP"."PRODUCT_ID")
4 - filter(TO_DATE(:BEG_DATE,'mm/dd/yyyy')<=TO_DATE(:END_DATE,'mm/dd/yyyy'))
10 - filter("O"."ORDER_STATUS"=0)
11 - access("O"."ORDER_DATE">=TO_DATE(:BEG_DATE,'mm/dd/yyyy') AND
           "O"."ORDER_DATE"<=TO_DATE(:END_DATE,'mm/dd/yyyy'))
12 - filter("C"."CUST_LAST_NAME"=:B1)
13 - access("C"."CUSTOMER_ID"="O"."CUSTOMER_ID")
15 - access("I"."ORDER_ID"="O"."ORDER_ID")
16 - access("P"."PRODUCT_ID"="I"."PRODUCT_ID")
```

© 2015 SOLARWINDS WORLDWIDE, LLC. ALL RIGHTS RESERVED.

SQL DIAGRAMMING

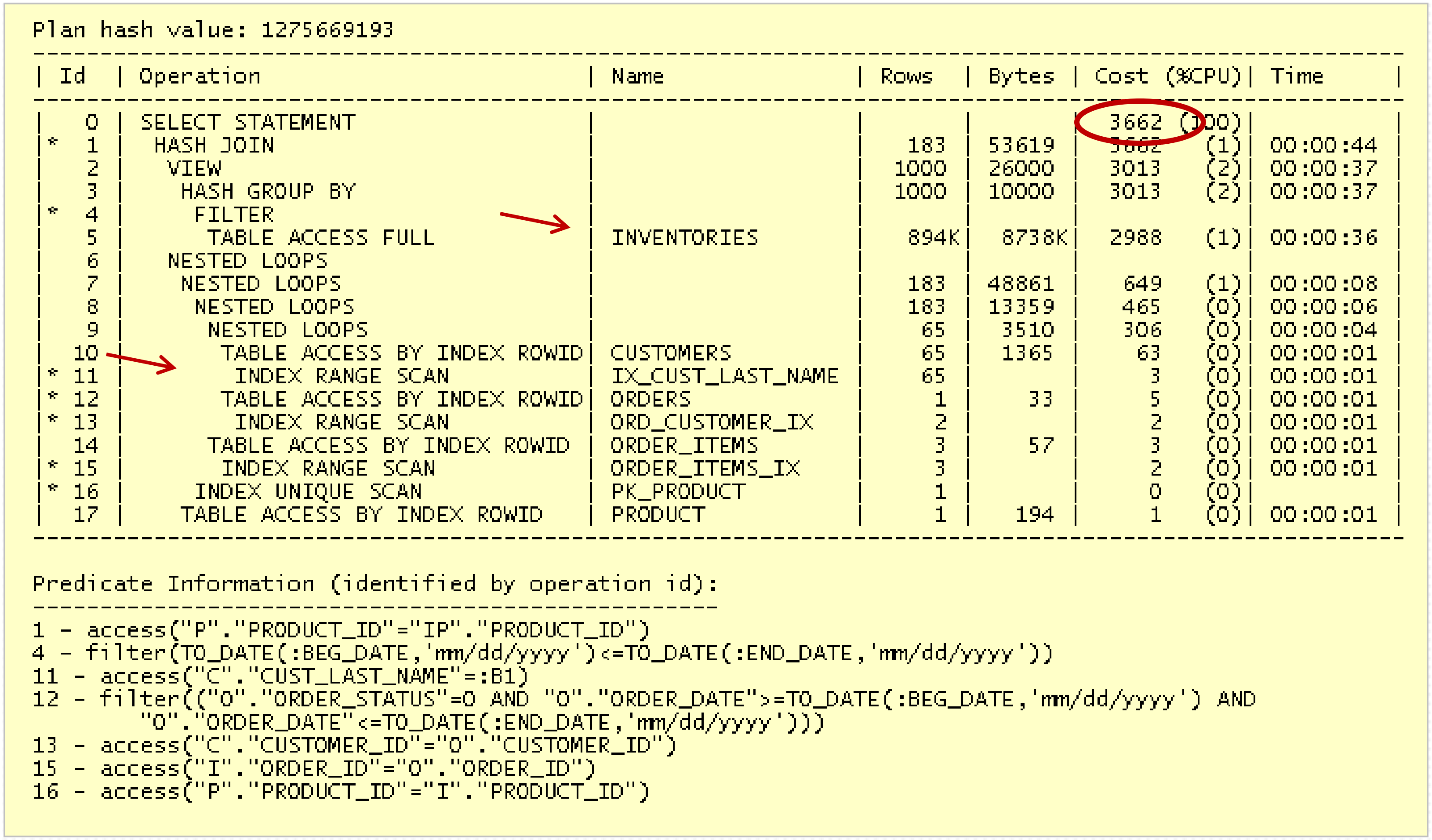


```
SELECT COUNT(1) FROM customer WHERE cust_last_name LIKE 'SMI%'

2054 / 5812142 =.00035


SELECT COUNT(1) FROM orders
WHERE order_status = 0
AND order_date BETWEEN TO_DATE(:BEG_DATE,'mm/dd/yyyy')
AND TO_DATE(:END_DATE,'mm/dd/yyyy`

8767 / 7399600 = .0011
```

© 2015 SOLARWINDS WORLDWIDE, LLC. ALL RIGHTS RESERVED.

NEW EXECUTION PLAN

» CREATE INDEX ix_cust_last_name ON customers (cust_last_name);



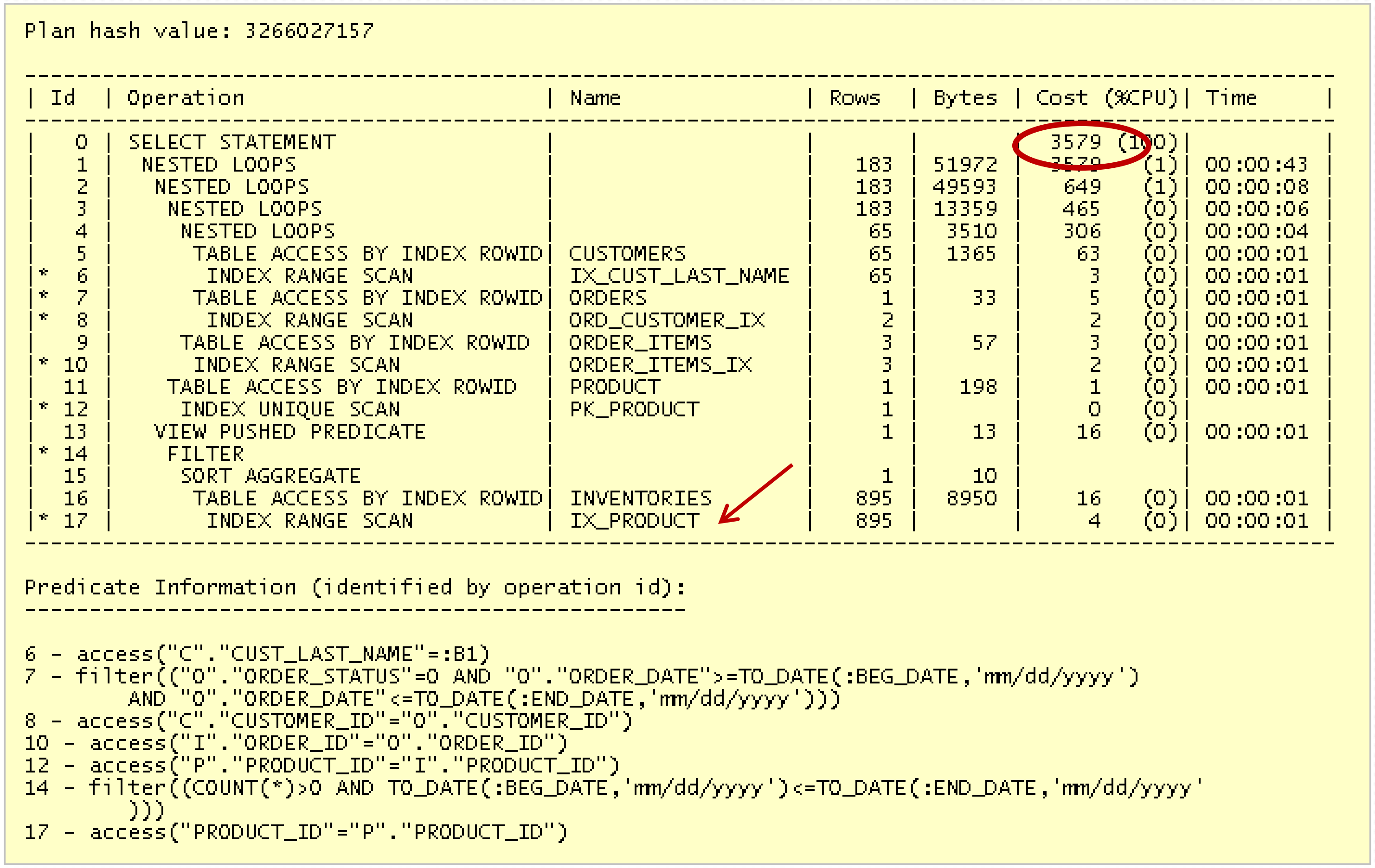
```
Plan hash value: 1275669193

-------------------------------------------------------------------------------------------------------
| Id  | Operation                         | Name               | Rows  | Bytes | Cost (%CPU)| Time     |
-------------------------------------------------------------------------------------------------------
|   0 | SELECT STATEMENT                  |                    |       |       |  3662 (100)|          |
|*  1 |  HASH JOIN                        |                    |   183 | 53619 |  3662   (1)| 00:00:44 |
|   2 |   VIEW                            |                    |  1000 | 26000 |  3013   (2)| 00:00:37 |
|   3 |    HASH GROUP BY                  |                    |  1000 | 10000 |  3013   (2)| 00:00:37 |
|*  4 |     FILTER                        |                    |       |       |            |          |
|   5 |      TABLE ACCESS FULL            | INVENTORIES        |  894K|  8738K|  2988   (1)| 00:00:36 |
|   6 |   NESTED LOOPS                    |                    |       |       |            |          |
|   7 |    NESTED LOOPS                   |                    |   183 | 48861 |   649   (1)| 00:00:08 |
|   8 |     NESTED LOOPS                  |                    |   183 | 13359 |   465   (0)| 00:00:06 |
|   9 |      NESTED LOOPS                 |                    |    65 |  3510 |   306   (0)| 00:00:04 |
|  10 |       TABLE ACCESS BY INDEX ROWID| CUSTOMERS          |    65 |  1365 |    63   (0)| 00:00:01 |
|* 11 |        INDEX RANGE SCAN           | IX_CUST_LAST_NAME  |    65 |       |     3   (0)| 00:00:01 |
|* 12 |       TABLE ACCESS BY INDEX ROWID| ORDERS             |     1 |    33 |     5   (0)| 00:00:01 |
|* 13 |        INDEX RANGE SCAN           | ORD_CUSTOMER_IX    |     2 |       |     2   (0)| 00:00:01 |
|  14 |      TABLE ACCESS BY INDEX ROWID  | ORDER_ITEMS        |     3 |    57 |     3   (0)| 00:00:01 |
|* 15 |       INDEX RANGE SCAN            | ORDER_ITEMS_IX     |     3 |       |     2   (0)| 00:00:01 |
|* 16 |     INDEX UNIQUE SCAN             | PK_PRODUCT         |     1 |       |     0   (0)|          |
|  17 |    TABLE ACCESS BY INDEX ROWID    | PRODUCT            |     1 |   194 |     1   (0)| 00:00:01 |
-------------------------------------------------------------------------------------------------------

Predicate Information (identified by operation id):
---------------------------------------------------
1 - access("P"."PRODUCT_ID"="IP"."PRODUCT_ID")
4 - filter(TO_DATE(:BEG_DATE,'mm/dd/yyyy')<=TO_DATE(:END_DATE,'mm/dd/yyyy'))
11 - access("C"."CUST_LAST_NAME"=:B1)
12 - filter(("O"."ORDER_STATUS"=0 AND "O"."ORDER_DATE">=TO_DATE(:BEG_DATE,'mm/dd/yyyy') AND
        "O"."ORDER_DATE"<=TO_DATE(:END_DATE,'mm/dd/yyyy')))
13 - access("C"."CUSTOMER_ID"="O"."CUSTOMER_ID")
15 - access("I"."ORDER_ID"="O"."ORDER_ID")
16 - access("P"."PRODUCT_ID"="I"."PRODUCT_ID")
```

- **Execution Stats – 11,182 Buffer Gets**

© 2015 SOLARWINDS WORLDWIDE, LLC. ALL RIGHTS RESERVED.

BEST EXECUTION PLAN

» CREATE INDEX ix_product ON inventories (product_id);



```
Plan hash value: 3266027157

-----------------------------------------------------------------------------------------------------
| Id  | Operation                          | Name               | Rows  | Bytes | Cost (%CPU)| Time     |
-----------------------------------------------------------------------------------------------------
|   0 | SELECT STATEMENT                   |                    |       |       |  3579 (100)|          |
|   1 |  NESTED LOOPS                      |                    |   183 | 51972 |  3579   (1)| 00:00:43 |
|   2 |   NESTED LOOPS                     |                    |   183 | 49593 |   649   (1)| 00:00:08 |
|   3 |    NESTED LOOPS                    |                    |   183 | 13359 |   465   (0)| 00:00:06 |
|   4 |     NESTED LOOPS                   |                    |    65 |  3510 |   306   (0)| 00:00:04 |
|   5 |      TABLE ACCESS BY INDEX ROWID| CUSTOMERS          |    65 |  1365 |    63   (0)| 00:00:01 |
|*  6 |       INDEX RANGE SCAN             | IX_CUST_LAST_NAME  |    65 |       |     3   (0)| 00:00:01 |
|*  7 |      TABLE ACCESS BY INDEX ROWID| ORDERS             |     1 |    33 |     5   (0)| 00:00:01 |
|*  8 |       INDEX RANGE SCAN             | ORD_CUSTOMER_IX    |     2 |       |     2   (0)| 00:00:01 |
|   9 |     TABLE ACCESS BY INDEX ROWID   | ORDER_ITEMS        |     3 |    57 |     3   (0)| 00:00:01 |
|* 10 |      INDEX RANGE SCAN              | ORDER_ITEMS_IX     |     3 |       |     2   (0)| 00:00:01 |
|  11 |    TABLE ACCESS BY INDEX ROWID    | PRODUCT            |     1 |   198 |     1   (0)| 00:00:01 |
|* 12 |     INDEX UNIQUE SCAN              | PK_PRODUCT         |     1 |       |     0   (0)|          |
|  13 |   VIEW PUSHED PREDICATE            |                    |     1 |    13 |    16   (0)| 00:00:01 |
|* 14 |    FILTER                          |                    |       |       |            |          |
|  15 |     SORT AGGREGATE                 |                    |     1 |    10 |            |          |
|  16 |      TABLE ACCESS BY INDEX ROWID| INVENTORIES        |   895 |  8950 |    16   (0)| 00:00:01 |
|* 17 |       INDEX RANGE SCAN             | IX_PRODUCT         |   895 |       |     4   (0)| 00:00:01 |
-----------------------------------------------------------------------------------------------------

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

6 - access("C"."CUST_LAST_NAME"=:B1)
7 - filter(("O"."ORDER_STATUS"=0 AND "O"."ORDER_DATE">=TO_DATE(:BEG_DATE,'mm/dd/yyyy')
        AND "O"."ORDER_DATE"<=TO_DATE(:END_DATE,'mm/dd/yyyy')))
8 - access("C"."CUSTOMER_ID"="O"."CUSTOMER_ID")
10 - access("I"."ORDER_ID"="O"."ORDER_ID")
12 - access("P"."PRODUCT_ID"="I"."PRODUCT_ID")
14 - filter((COUNT(*)>0 AND TO_DATE(:BEG_DATE,'mm/dd/yyyy')<=TO_DATE(:END_DATE,'mm/dd/yyyy'
        )))
17 - access("PRODUCT_ID"="P"."PRODUCT_ID")
```

- **Execution Stats – 262 Buffer Gets**

© 2015 SOLARWINDS WORLDWIDE, LLC. ALL RIGHTS RESERVED.

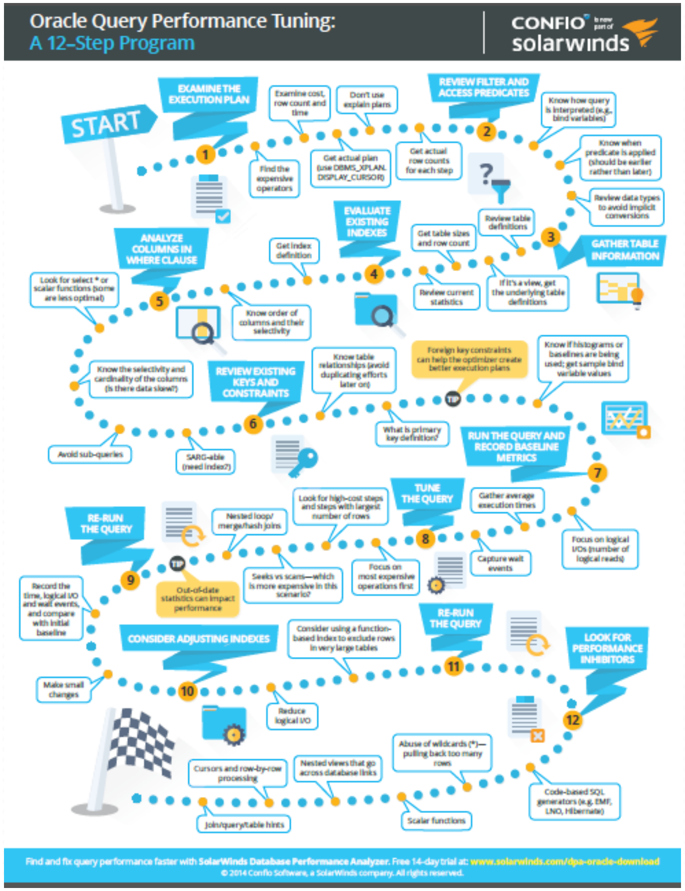
SUMMARY OF THE 12 STEP PROGRAM

1. Find Which SQL to Tune
2. Get Execution Plan
3. Examine the Execution Plan
4. Know the Optimizer Features used
5. Get Table & Column Info
6. Review Indexes & Constraints
7. Can’t Change the Query
8. Engineer out the Stupid
9. Gather Run-Time Details
10. Tune the Query
11. Re-Run the Query
12. Monitor to Check Tuning Results

[A 12 Step Program for Cats]

Download Poster at:

[12_Step_Tuning]



Oracle Query Performance Tuning:
A 12–Step Program

CONFIO is now part of solarwinds

START

1 EXAMINE THE EXECUTION PLAN

Find the expensive operators

Examine cost, row count and time

Get actual plan (use DBMS_XPLAN. DISPLAY_CURSOR)

Don't use explain plans

Get actual row counts for each step

2 REVIEW FILTER AND ACCESS PREDICATES

Know how query is interpreted (e.g., bind variables)

Know when predicate is applied (should be earlier rather than later)

Review data types to avoid implicit conversions

3 GATHER TABLE INFORMATION

Review table definitions

If it's a view, get the underlying table definitions

Get table sizes and row count

Review current statistics

4 EVALUATE EXISTING INDEXES

Get index definition

Know order of columns and their selectivity

5 ANALYZE COLUMNS IN WHERE CLAUSE

Look for select * or scalar functions (some are less optimal)

Know the selectivity and cardinality of the columns (is there data skew?)

Avoid sub-queries

SARG-able (need index?)

6 REVIEW EXISTING KEYS AND CONSTRAINTS

Know table relationships (avoid duplicating efforts later on)

What is primary key definition?

TIP Foreign key constraints can help the optimizer create better execution plans

Know if histograms or baselines are being used; get sample bind variable values

7 RUN THE QUERY AND RECORD BASELINE METRICS

Focus on logical I/Os (number of logical reads)

Gather average execution times

Capture wait events

8 TUNE THE QUERY

Focus on most expensive operations first

Look for high-cost steps and steps with largest number of rows

Seeks vs scans—which is more expensive in this scenario?

Nested loop/ merge/hash joins

TIP Out-of-date statistics can impact performance

9 RE-RUN THE QUERY

Record the time, logical I/O and wait events, and compare with initial baseline

Make small changes

10 CONSIDER ADJUSTING INDEXES

Reduce logical I/O

Consider using a function-based index to exclude rows in very large tables

11 RE-RUN THE QUERY

12 LOOK FOR PERFORMANCE INHIBITORS

Code-based SQL generators (e.g. EMF, LNO, Hibernate)

Abuse of wildcards (*)—pulling back too many rows

Scalar functions

Nested views that go across database links

Cursors and row-by-row processing

Join/query/table hints

Find and fix query performance faster with **SolarWinds Database Performance Analyzer.** Free 14-day trial at: www.solarwinds.com/dpa-oracle-download
© 2014 Confio Software, a SolarWinds company. All rights reserved.

© 2015 SOLARWINDS WORLDWIDE, LLC. ALL RIGHTS RESERVED.

RESOLVE PERFORMANCE ISSUES QUICKLY—FREE TRIAL

- Try ***Database Performance Analyzer*** FREE for 14 days
- Improve root cause of slow performance
  - Quickly identify root cause of issues that impact end-user response time
  - See historical trends over days, months, and years
  - Understand impact of VMware® performance
  - Agentless architecture with no dependence on Oracle Packs, installs in minutes

www.solarwinds.com/dpa-download/

Q & A

THANK YOU!

The SOLARWINDS and SOLARWINDS & Design marks are the exclusive property of SolarWinds Worldwide, LLC, are registered with the U.S. Patent and Trademark Office, and may be registered or pending registration in other countries. All other SolarWinds trademarks, service marks, and logos may be common law marks, registered or pending registration in the United States or in other countries. All other trademarks mentioned herein are used for identification purposes only and may be or are trademarks or registered trademarks of their respective companies.

© 2014 SOLARWINDS WORLDWIDE, LLC. ALL RIGHTS RESERVED.