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
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
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
12 Steps can take care of the problem

Image courtesy of Gentle-Stress-Relief.com
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)
RESPONSE TIME ANALYSIS (RTA)

Focus on Response Time

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

Identify Wait-Time at every step and rank bottlenecks by user impact.
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
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_AREA
SQL_ID
EXECUTIONS
PARSE_CALLS
BUFFER_GETS
DISK_READS

DBA_OBJECTS
OBJECT_ID
OBJECT_NAME
OBJECT_TYPE

V$SQL_BIND_CAPTURE
SQL_ID
NAME
DATATYPE_STRING
VALUE_STRING

© 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;
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
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')
```
1 hr of time in database. 78% is spent on db file sequential read

Over half of the time is spent on buffer waits
**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 utilizing 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.
IDENTIFY END-TO-END TIME

Accurate End-to-End Response Time Analysis

REQUEST

PlaceOrder

| END USER WEB LAYER | APPLICATION LAYER | DATABASE LAYER |

ProdSelect

| END USER WEB LAYER | APPLICATION LAYER | DATABASE LAYER |

Profile

| END USER WEB LAYER | APPLICATION LAYER | DATABASE LAYER |

WAIT-TIME by STEP

HOURS OF WAIT-TIME

© 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
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;
```

<table>
<thead>
<tr>
<th>Id</th>
<th>Operation &amp; Option</th>
<th>Name</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>SELECT STATEMENT</td>
<td></td>
</tr>
<tr>
<td>1</td>
<td>NESTED LOOPS</td>
<td></td>
</tr>
<tr>
<td>2</td>
<td>TABLE ACCESS BY INDEX ROWID</td>
<td>DEPT</td>
</tr>
<tr>
<td>3</td>
<td>INDEX UNIQUE SCAN</td>
<td>PK_DEPT</td>
</tr>
<tr>
<td>4</td>
<td>TABLE ACCESS FULL</td>
<td>EMP</td>
</tr>
</tbody>
</table>
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
DBMS_XPLAN

Functions in 12c

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

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'))
```
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
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

<table>
<thead>
<tr>
<th>Id</th>
<th>Operation</th>
<th>Name</th>
<th>Rows</th>
<th>Bytes</th>
<th>Cost &lt;&amp;CPU&gt;</th>
<th>Time</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>SELECT STATEMENT</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>1</td>
<td>NESTED LOOPS</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>2</td>
<td>TABLE ACCESS BY INDEX ROWID</td>
<td>DEPT</td>
<td>1503</td>
<td>54108</td>
<td>15</td>
<td>0:00:00:01</td>
</tr>
<tr>
<td>3</td>
<td>INDEX UNIQUE SCAN</td>
<td>PK_DEPT</td>
<td>1</td>
<td>11</td>
<td>2</td>
<td>0:00:00:01</td>
</tr>
<tr>
<td>4</td>
<td>TABLE ACCESS FULL</td>
<td>EMP</td>
<td>1503</td>
<td>37575</td>
<td>13</td>
<td>0:00:00:01</td>
</tr>
</tbody>
</table>

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> 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

<table>
<thead>
<tr>
<th>Id</th>
<th>Operation</th>
<th>Name</th>
<th>Rows</th>
<th>Bytes</th>
<th>Cost (%CPU)</th>
<th>Time</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>SELECT STATEMENT</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>1</td>
<td>NESTED LOOPS</td>
<td></td>
<td>3958</td>
<td>139K</td>
<td>15 &lt;100&gt;</td>
<td>00:00:01</td>
</tr>
<tr>
<td>2</td>
<td>TABLE ACCESS BY INDEX ROWID</td>
<td>DEPT</td>
<td>1</td>
<td>11</td>
<td>2 &lt;0&gt;</td>
<td>00:00:01</td>
</tr>
<tr>
<td>3</td>
<td>INDEX UNIQUE SCAN</td>
<td>PK_DEPT</td>
<td>1</td>
<td></td>
<td>1 &lt;0&gt;</td>
<td>00:00:01</td>
</tr>
<tr>
<td>4</td>
<td>TABLE ACCESS FULL</td>
<td>EMP</td>
<td>3958</td>
<td>98950</td>
<td>13 &lt;0&gt;</td>
<td>00:00:01</td>
</tr>
</tbody>
</table>

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_x5.1st 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
```

<table>
<thead>
<tr>
<th>DEPTNO</th>
<th>COUNT(*)</th>
</tr>
</thead>
<tbody>
<tr>
<td>10</td>
<td>77</td>
</tr>
<tr>
<td>20</td>
<td>1500</td>
</tr>
<tr>
<td>30</td>
<td>478</td>
</tr>
<tr>
<td>40</td>
<td>3958</td>
</tr>
</tbody>
</table>

Optimizer mode: ALL_ROWS

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

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
4. KNOW THE OPTIMIZER FEATURES USED

» Show parameter optimizer

<table>
<thead>
<tr>
<th>NAME</th>
<th>TYPE</th>
<th>VALUE</th>
</tr>
</thead>
<tbody>
<tr>
<td>optimizer_adaptive_features</td>
<td>boolean</td>
<td>TRUE</td>
</tr>
<tr>
<td>optimizer_adaptive_reporting_only</td>
<td>boolean</td>
<td>FALSE</td>
</tr>
<tr>
<td>optimizer_capture_sql_plan_baselines</td>
<td>boolean</td>
<td>FALSE</td>
</tr>
<tr>
<td>optimizer_dynamic_sampling</td>
<td>integer</td>
<td>2</td>
</tr>
<tr>
<td>optimizer_features_enable</td>
<td>string</td>
<td>12.1.0.1</td>
</tr>
<tr>
<td>optimizer_index_caching</td>
<td>integer</td>
<td>0</td>
</tr>
<tr>
<td>optimizer_index_cost_adj</td>
<td>integer</td>
<td>100</td>
</tr>
<tr>
<td>optimizer_mode</td>
<td>string</td>
<td>ALL_ROWS</td>
</tr>
<tr>
<td>optimizer_secure_view_merging</td>
<td>boolean</td>
<td>TRUE</td>
</tr>
<tr>
<td>optimizer_use_invisible_indexes</td>
<td>boolean</td>
<td>FALSE</td>
</tr>
<tr>
<td>optimizer_use_pending_statistics</td>
<td>boolean</td>
<td>FALSE</td>
</tr>
<tr>
<td>optimizer_use_sql_plan_baselines</td>
<td>boolean</td>
<td>TRUE</td>
</tr>
</tbody>
</table>

» 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

---
- statistics feedback used for this statement
- this is an adaptive plan (rows marked '-' are inactive)
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
HOW THE OPTIMIZER WORKS

Parsed Query (from Parser)

**Query Transformer** – rewrites query to be more efficient

**Transformed Query**

**Estimator** – looks at selectivity, cardinality & cost

**Query + Estimates**

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

Data Dictionary

Schema Definition

Statistics

Init.ora parameter to control behavior: OPTIMIZER_FEATURES_ENABLED

Default Plan sent to Row Source Generator
EXECUTION PLAN USING SPM (11G)

Select * from dba_sql_plans_baselines;

<table>
<thead>
<tr>
<th>SQL_HANDLE</th>
<th>PLAN_NAME</th>
<th>SQL_TEXT</th>
<th>ENA</th>
<th>ACC</th>
<th>FIX</th>
<th>OPTIMIZER_COST</th>
</tr>
</thead>
<tbody>
<tr>
<td>SYS_SQL_547c574e74755d78</td>
<td>SYS_SQL_PLAN_74755d78e1961c6e</td>
<td>select count(*) from orders a, customers YES YES NO 27</td>
<td></td>
<td></td>
<td></td>
<td>19309</td>
</tr>
<tr>
<td>SYS_SQL_9c3e4291df2a9446</td>
<td>SYS_SQL_PLAN_df2a9446ed88afee</td>
<td>SELECT ATTRIBUTE,SCOPE,NUMERIC VALUE,CHA YES YES NO</td>
<td></td>
<td></td>
<td>YES</td>
<td>2</td>
</tr>
<tr>
<td>SYS_SQL_e744325067d2db2f</td>
<td>SYS_SQL_PLAN_67d2db2fed88afee</td>
<td>SELECT CHAR_VALUE FROM SYSTEM.PRODUCT_PR YES YES NO</td>
<td></td>
<td></td>
<td>YES</td>
<td>2</td>
</tr>
</tbody>
</table>

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

<table>
<thead>
<tr>
<th>Id</th>
<th>Operation</th>
<th>Name</th>
<th>Rows</th>
<th>Bytes</th>
<th>Cost (%CPU)</th>
<th>Time</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>SELECT STATEMENT</td>
<td></td>
<td></td>
<td></td>
<td>2 (100)</td>
<td></td>
</tr>
<tr>
<td>1</td>
<td>TABLE ACCESS BY INDEX ROWID</td>
<td>ITEM</td>
<td>1</td>
<td>69</td>
<td>2 (0)</td>
<td>00:00:01</td>
</tr>
<tr>
<td>* 2</td>
<td>INDEX UNIQUE SCAN</td>
<td>ITEM_I1</td>
<td>1</td>
<td></td>
<td>1 (0)</td>
<td>00:00:01</td>
</tr>
</tbody>
</table>

Predicate Information (identified by operation id):

2 - access("I_ID"=:B1)

Note

- SQL plan baseline SQL_PLAN_gsrrup3zurt88e90e4d55 used for this statement
12C ADAPTIVE QUERY OPTIMIZER

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

Adaptive Query Optimizer

Adaptive Plans
- Join Methods
- Parallel Distribution

Adaptive Statistics
- Dynamic Statistics
- Automatic Reoptimization
- Sql Plan Directives
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

```sql
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

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

<table>
<thead>
<tr>
<th>Id</th>
<th>Operation</th>
<th>Name</th>
<th>Rows</th>
<th>Bytes</th>
<th>Cost (%CPU)</th>
<th>Time</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>SELECT STATEMENT</td>
<td>HASH JOIN</td>
<td>1895</td>
<td>73905</td>
<td>13184 (100)</td>
<td>00:00:01</td>
</tr>
<tr>
<td>2</td>
<td>TABLE ACCESS FULL ORDER_ITEMS</td>
<td>1895</td>
<td>20845</td>
<td>11862</td>
<td>13184 (3)</td>
<td>00:00:01</td>
</tr>
<tr>
<td>3</td>
<td>TABLE ACCESS FULL PRODUCT</td>
<td>1022K</td>
<td>27M</td>
<td>1314</td>
<td>13184 (2)</td>
<td>00:00:01</td>
</tr>
</tbody>
</table>

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

<table>
<thead>
<tr>
<th>Id</th>
<th>Operation</th>
<th>Name</th>
<th>Rows</th>
<th>Bytes</th>
<th>Cost (%CPU)</th>
<th>Time</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>SELECT STATEMENT</td>
<td>NESTED LOOPS</td>
<td>1895</td>
<td>73905</td>
<td>13184 (100)</td>
<td>00:00:01</td>
</tr>
<tr>
<td>2</td>
<td>NESTED LOOPS</td>
<td>ORDER_ITEMS</td>
<td>1895</td>
<td>20845</td>
<td>11862 (3)</td>
<td>00:00:01</td>
</tr>
<tr>
<td>3</td>
<td>TABLE ACCESS FULL</td>
<td>PRODUCT_IDX</td>
<td>1</td>
<td>28</td>
<td>1314 (2)</td>
<td>00:00:01</td>
</tr>
</tbody>
</table>
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

<table>
<thead>
<tr>
<th>Id</th>
<th>Operation</th>
<th>Name</th>
<th>Rows</th>
<th>Bytes</th>
<th>Cost (%CPU)</th>
<th>Time</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>SELECT STATEMENT</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>1</td>
<td>NESTED LOOPS</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>2</td>
<td>NESTED LOOPS</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>3</td>
<td>TABLE ACCESS FULL</td>
<td>ORDER_ITEMS</td>
<td>1895</td>
<td>73905</td>
<td>13184 (3)</td>
<td>00:00:01</td>
</tr>
<tr>
<td>4</td>
<td>INDEX RANGE SCAN</td>
<td>PRODUCT_IDX</td>
<td>1</td>
<td>20845</td>
<td>11862 (3)</td>
<td>00:00:01</td>
</tr>
<tr>
<td>5</td>
<td>TABLE ACCESS BY INDEX ROWID</td>
<td>PRODUCT</td>
<td>1</td>
<td>28</td>
<td>1314 (2)</td>
<td>00:00:01</td>
</tr>
</tbody>
</table>

Predicate Information (identified by operation id):

3 - filter("0"."UNIT_PRICE"=:B1 AND "0"."QUANTITY">:B2))
4 - access("0"."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

<table>
<thead>
<tr>
<th>Id</th>
<th>Operation</th>
<th>Name</th>
<th>Rows</th>
<th>Bytes</th>
<th>Cost (%CPU)</th>
<th>Time</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>SELECT STATEMENT</td>
<td></td>
<td>323</td>
<td>12597</td>
<td>12468 (3)</td>
<td>00:00:01</td>
</tr>
<tr>
<td>1</td>
<td>NESTED LOOPS</td>
<td></td>
<td>323</td>
<td>12597</td>
<td>12468 (3)</td>
<td>00:00:01</td>
</tr>
<tr>
<td>2</td>
<td>NESTED LOOPS</td>
<td></td>
<td>174</td>
<td>1914</td>
<td>11946 (3)</td>
<td>00:00:01</td>
</tr>
<tr>
<td>3</td>
<td>TABLE ACCESS FULL</td>
<td>ORDER_ITEMS</td>
<td>1</td>
<td>2</td>
<td>3 (0)</td>
<td>00:00:01</td>
</tr>
<tr>
<td>4</td>
<td>INDEX RANGE SCAN</td>
<td>PRODUCT_IDX</td>
<td>2</td>
<td>56</td>
<td>3 (0)</td>
<td>00:00:01</td>
</tr>
<tr>
<td>5</td>
<td>TABLE ACCESS BY INDEX ROWID</td>
<td>PRODUCT</td>
<td>1</td>
<td>2</td>
<td>3 (0)</td>
<td>00:00:01</td>
</tr>
</tbody>
</table>
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

- **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

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

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;
Adapted on first execution
alter session set optimizer_adaptive_reporting_only=FALSE;

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

Plan hash value: 3627148456

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

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)
WHAT CHANGED?

After Reoptimization has occurred

```sql
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

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

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)
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

» Use TuningStats.sql
  - OracleTuningStats.sql

» Run it for expensive data access targets
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;
```

<table>
<thead>
<tr>
<th>COLUMN_NAME</th>
<th>NUM_DISTINCT</th>
<th>NUM_NULLS</th>
<th>NUM_BUCKETS</th>
<th>DENSITY</th>
<th>SAMPLE_SIZE</th>
</tr>
</thead>
<tbody>
<tr>
<td>COMM</td>
<td>1534</td>
<td>4</td>
<td>0</td>
<td>1.00065189</td>
<td>1583</td>
</tr>
<tr>
<td>DEPTNO</td>
<td>4</td>
<td>0</td>
<td>0</td>
<td>1.25</td>
<td>6013</td>
</tr>
<tr>
<td>EMPNO</td>
<td>6013</td>
<td>0</td>
<td>0</td>
<td>1.00166306</td>
<td>6013</td>
</tr>
<tr>
<td>ENAME</td>
<td>6013</td>
<td>0</td>
<td>0</td>
<td>1.00166306</td>
<td>6013</td>
</tr>
<tr>
<td>HIREDATE</td>
<td>88</td>
<td>0</td>
<td>0</td>
<td>1.01136363</td>
<td>6013</td>
</tr>
<tr>
<td>JOB</td>
<td>22</td>
<td>0</td>
<td>0</td>
<td>1.04545445</td>
<td>6013</td>
</tr>
<tr>
<td>MGR</td>
<td>6</td>
<td>6000</td>
<td>0</td>
<td>1.16666667</td>
<td>6013</td>
</tr>
<tr>
<td>SAL</td>
<td>6000</td>
<td>0</td>
<td>0</td>
<td>1.00166667</td>
<td>6013</td>
</tr>
</tbody>
</table>

Would an index on EMP.DEPTNO increase performance?
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')

<table>
<thead>
<tr>
<th>COLUMN_NAME</th>
<th>NUM_DISTINCT</th>
<th>NUM_NULLS</th>
<th>NUM_buckets</th>
<th>DENSITY</th>
<th>SAMPLE_SIZE</th>
<th>HISTOGRAM</th>
</tr>
</thead>
<tbody>
<tr>
<td>COMM</td>
<td>1534</td>
<td>4430</td>
<td>1</td>
<td>0.00065189</td>
<td>1582</td>
<td>NONE</td>
</tr>
<tr>
<td>DEPTNO</td>
<td>4</td>
<td>0</td>
<td>1</td>
<td>0.00083153</td>
<td>6013</td>
<td>FREQUENCY</td>
</tr>
<tr>
<td>EMPNO</td>
<td>6013</td>
<td>0</td>
<td>4</td>
<td>0.00166306</td>
<td>6013</td>
<td>NONE</td>
</tr>
<tr>
<td>ENAME</td>
<td>6013</td>
<td>0</td>
<td>254</td>
<td>0.000166306</td>
<td>6013</td>
<td>HEIGHT_BALANCED</td>
</tr>
<tr>
<td>HIREDATE</td>
<td>88</td>
<td>0</td>
<td>22</td>
<td>0.011363636</td>
<td>6013</td>
<td>NONE</td>
</tr>
<tr>
<td>JOB</td>
<td>22</td>
<td>0</td>
<td>22</td>
<td>0.00083153</td>
<td>6013</td>
<td>FREQUENCY</td>
</tr>
<tr>
<td>MGR</td>
<td>6</td>
<td>6000</td>
<td>1</td>
<td>0.00166667</td>
<td>13</td>
<td>NONE</td>
</tr>
<tr>
<td>SAL</td>
<td>6000</td>
<td>0</td>
<td>1</td>
<td>0.00166667</td>
<td>6013</td>
<td>NONE</td>
</tr>
</tbody>
</table>

exec dbms_stats.gather_table_stats( ownname => 'SCOTT', tabname => 'EMP', method_opt=>'FOR COLUMNS deptno SIZE 2');

<table>
<thead>
<tr>
<th>COLUMN_NAME</th>
<th>NUM_DISTINCT</th>
<th>NUM_NULLS</th>
<th>NUM_buckets</th>
<th>DENSITY</th>
<th>SAMPLE_SIZE</th>
<th>HISTOGRAM</th>
</tr>
</thead>
<tbody>
<tr>
<td>COMM</td>
<td>1534</td>
<td>4430</td>
<td>1</td>
<td>0.00065189</td>
<td>1582</td>
<td>NONE</td>
</tr>
<tr>
<td>DEPTNO</td>
<td>4</td>
<td>0</td>
<td>1</td>
<td>0.00083153</td>
<td>6013</td>
<td>FREQUENCY</td>
</tr>
<tr>
<td>EMPNO</td>
<td>6013</td>
<td>0</td>
<td>2</td>
<td>0.00166306</td>
<td>6013</td>
<td>NONE</td>
</tr>
<tr>
<td>ENAME</td>
<td>6013</td>
<td>0</td>
<td>254</td>
<td>0.000166306</td>
<td>6013</td>
<td>HEIGHT_BALANCED</td>
</tr>
<tr>
<td>HIREDATE</td>
<td>88</td>
<td>0</td>
<td>22</td>
<td>0.011363636</td>
<td>6013</td>
<td>NONE</td>
</tr>
<tr>
<td>JOB</td>
<td>22</td>
<td>0</td>
<td>22</td>
<td>0.00083153</td>
<td>6013</td>
<td>FREQUENCY</td>
</tr>
<tr>
<td>MGR</td>
<td>6</td>
<td>6000</td>
<td>1</td>
<td>0.00166667</td>
<td>13</td>
<td>NONE</td>
</tr>
<tr>
<td>SAL</td>
<td>6000</td>
<td>0</td>
<td>1</td>
<td>0.00166667</td>
<td>6013</td>
<td>NONE</td>
</tr>
</tbody>
</table>

New in 12c – Top Frequency / Hybrid (Height Balanced going away)
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';
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
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
Data Type Issues - db file scattered read

```sql
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=1l7)
    TABLE ACCESS (BY INDEX ROWID) OF 'DATA_OUT' (TABLE) (Cost=1 Card=1 Bytes=1l7)
    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));
```

<table>
<thead>
<tr>
<th>Id</th>
<th>Operation</th>
<th>Name</th>
<th>Rows</th>
<th>Bytes</th>
<th>Cost (%CPU)</th>
<th>Time</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>SELECT STATEMENT</td>
<td></td>
<td></td>
<td></td>
<td>370 (100)</td>
<td></td>
</tr>
<tr>
<td>/*</td>
<td>TABLE ACCESS FULL</td>
<td>DATA_OUT</td>
<td>1</td>
<td>117</td>
<td>370 (4)</td>
<td>00:00:05</td>
</tr>
</tbody>
</table>

Predicate Information (identified by operation id):

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

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

PLAN_TABLE_OUTPUT

SQL_ID cdgndknbhfc0cq, 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

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

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)
CHANGE THE BASELINE

```sql
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 0h9tjuds1bgas6, 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

<table>
<thead>
<tr>
<th>Id</th>
<th>Operation</th>
<th>Name</th>
<th>Rows</th>
<th>Bytes</th>
<th>Cost (%CPU)</th>
<th>Time</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>SELECT STATEMENT</td>
<td></td>
<td></td>
<td></td>
<td>10238 (100)</td>
<td>00:00:01</td>
</tr>
<tr>
<td>1</td>
<td>NESTED LOOPS</td>
<td></td>
<td></td>
<td></td>
<td>10238 (100)</td>
<td>00:00:01</td>
</tr>
<tr>
<td>2</td>
<td>NESTED LOOPS</td>
<td></td>
<td></td>
<td></td>
<td>10238 (100)</td>
<td>00:00:01</td>
</tr>
<tr>
<td>* 3</td>
<td>TABLE ACCESS BY INDEX ROWID BATCHED</td>
<td>ORDER_ITEMS</td>
<td>1</td>
<td>33</td>
<td>10238 (1)</td>
<td>00:00:01</td>
</tr>
<tr>
<td>* 4</td>
<td>INDEX RANGE SCAN</td>
<td>OI_PRODUCT_ID</td>
<td>11354</td>
<td>28</td>
<td>3 (0)</td>
<td>00:00:01</td>
</tr>
<tr>
<td>* 5</td>
<td>INDEX RANGE SCAN</td>
<td>PRODUCT_PRODUCT_ID</td>
<td>1</td>
<td>2</td>
<td>3 (0)</td>
<td>00:00:01</td>
</tr>
<tr>
<td>6</td>
<td>TABLE ACCESS BY INDEX ROWID</td>
<td>PRODUCT</td>
<td>1</td>
<td>22</td>
<td>3 (0)</td>
<td>00:00:01</td>
</tr>
</tbody>
</table>
```
### CHANGE THE BASELINE

**SQL**

```sql
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-yyyy HH24:MI') last_executed
  3 from dba_sql_plan_baselines where creator = 'SOE'
  4 order by 1;

<table>
<thead>
<tr>
<th>PLAN_NAME</th>
<th>SQL_TEXT</th>
<th>ENA ACC FIX</th>
<th>OPTIMIZER_COST</th>
<th>LAST_EXECUTED</th>
</tr>
</thead>
<tbody>
<tr>
<td>SQL_PLAN_dqqrmfgazp9rp4dcad05d</td>
<td>select /* jg */ p.product_name</td>
<td>YES</td>
<td>YES</td>
<td>NO</td>
</tr>
</tbody>
</table>

**PL/SQL**

```sql
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.

### Additional SQL Queries

**SQL**

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

<table>
<thead>
<tr>
<th>SQL_ID</th>
<th>CHILD_NUMBER</th>
<th>PLAN_HASH_VALUE</th>
<th>SQL_FULLTEXT</th>
</tr>
</thead>
<tbody>
<tr>
<td>12zj3utbrq3kb</td>
<td>0</td>
<td>3021036780</td>
<td>select /* jg */ p.product_name from order_items o, product p where o.unit_price</td>
</tr>
<tr>
<td>0h9tjus1bgas6</td>
<td>0</td>
<td>3794610757</td>
<td>select /*+ USE_NL(p) <em>/ /</em> jg */ p.product_name from order_items o, product p wh</td>
</tr>
</tbody>
</table>

**SQL**

```sql
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**

```sql
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-yyyy HH24:MI') last_executed
  3 from dba_sql_plan_baselines where creator = 'SOE'
  4 order by 1;
```

<table>
<thead>
<tr>
<th>SQL_HANDLE</th>
<th>PLAN_NAME</th>
<th>SQL_TEXT</th>
<th>ENA ACC FIX</th>
</tr>
</thead>
<tbody>
<tr>
<td>SQL_db5af373d5faa6f5</td>
<td>SQL_PLAN_dqqrmfgazp9rp4dcad05d</td>
<td>select /* jg */ p.product_name</td>
<td>NO</td>
</tr>
<tr>
<td>SQL_db5af373d5faa6f5</td>
<td>SQL_PLAN_dqqrmfgazp9rp2f36d8b</td>
<td>select /* jg */ p.product_name</td>
<td>YES</td>
</tr>
</tbody>
</table>
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’
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
12C PARALLEL DISTRIBUTION

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

```sql
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

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

```sql
ALTER SESSION SET PARALLEL DEGREE POLICY = 'ADAPTIVE';
```

```sql
SELECT * FROM table(DBMS_XPLAN.DISPLAY_CURSOR('13sprg5vbk8d9', '2', FORMAT='ALLSTATS_LAST'));
```

<table>
<thead>
<tr>
<th>Id</th>
<th>Operation</th>
<th>Name</th>
<th>Starts</th>
<th>E-Rows</th>
<th>A-Rows</th>
<th>A-Time</th>
<th>Buffers</th>
<th>Reads</th>
<th>OMem</th>
<th>1Mem</th>
<th>Used-Mem</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>SELECT STATEMENT</td>
<td></td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td></td>
</tr>
<tr>
<td>1</td>
<td>PX COORDINATOR</td>
<td></td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td></td>
</tr>
<tr>
<td>2</td>
<td>PX SEND QC (RANDOM)</td>
<td></td>
<td>0</td>
<td>4966K</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td></td>
</tr>
<tr>
<td>3</td>
<td>HASH JOIN</td>
<td>:TQ10001</td>
<td>0</td>
<td>4966K</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td></td>
</tr>
<tr>
<td>4</td>
<td>PX RECEIVE</td>
<td></td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td></td>
</tr>
<tr>
<td>5</td>
<td>PX SEND BROADCAST</td>
<td>:TQ10000</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td></td>
</tr>
<tr>
<td>6</td>
<td>NESTED LOOPS</td>
<td></td>
<td>1</td>
<td>525</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td></td>
</tr>
<tr>
<td>7</td>
<td>NESTED LOOPS</td>
<td></td>
<td>1</td>
<td>3786</td>
<td>525</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td></td>
</tr>
<tr>
<td>8</td>
<td>NESTED LOOPS</td>
<td></td>
<td>1</td>
<td>2743</td>
<td>385</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td></td>
</tr>
<tr>
<td>9</td>
<td>PX BLOCK ITERATOR</td>
<td></td>
<td>1</td>
<td>201</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td></td>
</tr>
<tr>
<td>10</td>
<td>TABLE ACCESS FULL</td>
<td>CUSTOMERS</td>
<td>17</td>
<td>1182</td>
<td>201</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td></td>
</tr>
<tr>
<td>11</td>
<td>TABLE ACCESS BY INDEX ROWID BATCHED</td>
<td>ORDERS</td>
<td>201</td>
<td>2</td>
<td>385</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td></td>
</tr>
<tr>
<td>12</td>
<td>INDEX RANGE SCAN</td>
<td>ORD_CUSTOMER_IDX</td>
<td>201</td>
<td>2</td>
<td>385</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td></td>
</tr>
<tr>
<td>13</td>
<td>INDEX RANGE SCAN</td>
<td>ORDER_ITEMS_IDX</td>
<td>385</td>
<td>3</td>
<td>525</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td></td>
</tr>
<tr>
<td>14</td>
<td>TABLE ACCESS BY INDEX ROWID</td>
<td>ORDER_ITEMS</td>
<td>25</td>
<td>525</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td></td>
</tr>
<tr>
<td>15</td>
<td>PX BLOCK ITERATOR</td>
<td></td>
<td>0</td>
<td>131K</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td></td>
</tr>
<tr>
<td>16</td>
<td>TABLE ACCESS FULL</td>
<td>PRODUCT</td>
<td>0</td>
<td>131K</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td></td>
</tr>
</tbody>
</table>

Predicate Information (identified by operation id):

- access(‘I’,”PRODUCT_ID=”’”PRODUCT_ID”’)
- access(‘I’,”CUSTOMER_ID”’”CUSTOMER_ID”’)
- access(‘C’,”CUST_FIRST_NAME”’”joshua”’ OR ”CUST_LAST_NAME”’”modena”’)
- access(‘C’,”CUST_FIRST_NAME”’”joshua”’ OR ”CUST_LAST_NAME”’”modena”’)
- access(‘C’,”CUST_LAST_NAME”’”modena”’)
- access(‘C’,”CUST_LAST_NAME”’”modena”’)

Note:
- dynamic statistics used: dynamic sampling (level-AUTO)
- automatic DDP: Computed Degree of Parallelism is & because of degree limit
  - parallel scans affinitized
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_EXTENDED_STATS
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;
SQL PLAN DIRECTIVES

No Statistics on Employee table so Optimizer uses Directive

```sql
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

<table>
<thead>
<tr>
<th>Id</th>
<th>Operation</th>
<th>Name</th>
<th>Rows</th>
<th>Bytes</th>
<th>Cost (%CPU)</th>
<th>Time</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>SELECT STATEMENT</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>1</td>
<td>SORT AGGREGATE</td>
<td></td>
<td>1</td>
<td>104</td>
<td>21207 (100)</td>
<td>00:00:01</td>
</tr>
<tr>
<td>2</td>
<td>TABLE ACCESS FULL</td>
<td>EMPLOYEE</td>
<td>991K</td>
<td>98M</td>
<td>21207 (2)</td>
<td>00:00:01</td>
</tr>
</tbody>
</table>

Predicate Information (identified by operation id):

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

Note
-----
- statistics feedback used for this statement
```
SQL PLAN DIRECTIVES

With Bad Statistics on Employee table

```
SQL> select count(*) from employee where work_city = 'Munich'
COUNT(*)
---------
991232

SQL> select count(*) from employee where work_city = 'Munich' and work_country = 'Germany'
Plan hash value: 301197670

<table>
<thead>
<tr>
<th>Id</th>
<th>Operation</th>
<th>Name</th>
<th>Rows</th>
<th>Bytes</th>
<th>Cost (%CPU)</th>
<th>Time</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>SELECT STATEMENT</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>1</td>
<td>SORT AGGREGATE</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>*</td>
<td>TABLE ACCESS FULL</td>
<td>EMPLOYEE</td>
<td>105K</td>
<td>1860K</td>
<td>21301 (100)</td>
<td>00:00:01</td>
</tr>
</tbody>
</table>

Predicate Information (identified by operation id):
2 - filter(("WORK_CITY"='Munich' AND "WORK_COUNTRY"='Germany'))
```
EXTENDED STATISTICS

```sql
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';

<table>
<thead>
<tr>
<th>COLUMN_NAME</th>
<th>NUM_DISTINCT</th>
<th>NUM_NULLS</th>
<th>SAMPLE_SIZE</th>
<th>HISTOGRAM</th>
</tr>
</thead>
<tbody>
<tr>
<td>WORK_COUNTRY</td>
<td>17</td>
<td>0</td>
<td>9282038</td>
<td>FREQUENCY</td>
</tr>
<tr>
<td>WORK_CITY</td>
<td>27</td>
<td>0</td>
<td>9282038</td>
<td>FREQUENCY</td>
</tr>
<tr>
<td>DEPTNO</td>
<td>5</td>
<td>0</td>
<td>9282038</td>
<td>NONE</td>
</tr>
<tr>
<td>COMM</td>
<td>9</td>
<td>6398293</td>
<td>2883745</td>
<td>NONE</td>
</tr>
<tr>
<td>etc...</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

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

<table>
<thead>
<tr>
<th>COLUMN_NAME</th>
<th>NUM_DISTINCT</th>
<th>NUM_NULLS</th>
<th>SAMPLE_SIZE</th>
<th>HISTOGRAM</th>
</tr>
</thead>
<tbody>
<tr>
<td>SYS_STUMJ_IO5JF4V1H4MCA#TCC#XW</td>
<td>27</td>
<td>0</td>
<td>9282038</td>
<td>NONE</td>
</tr>
<tr>
<td>WORK_COUNTRY</td>
<td>17</td>
<td>0</td>
<td>9282038</td>
<td>FREQUENCY</td>
</tr>
<tr>
<td>WORK_CITY</td>
<td>27</td>
<td>0</td>
<td>9282038</td>
<td>FREQUENCY</td>
</tr>
<tr>
<td>DEPTNO</td>
<td>5</td>
<td>0</td>
<td>9282038</td>
<td>NONE</td>
</tr>
<tr>
<td>COMM</td>
<td>9</td>
<td>6398293</td>
<td>2883745</td>
<td>NONE</td>
</tr>
<tr>
<td>etc...</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

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

<table>
<thead>
<tr>
<th>COLUMN_NAME</th>
<th>NUM_DISTINCT</th>
<th>NUM_NULLS</th>
<th>SAMPLE_SIZE</th>
<th>HISTOGRAM</th>
</tr>
</thead>
<tbody>
<tr>
<td>SYS_STUMJ_IO5JF4V1H4MCA#TCC#XW</td>
<td>27</td>
<td>0</td>
<td>9282038</td>
<td>FREQUENCY</td>
</tr>
<tr>
<td>WORK_COUNTRY</td>
<td>17</td>
<td>0</td>
<td>9282038</td>
<td>FREQUENCY</td>
</tr>
<tr>
<td>WORK_CITY</td>
<td>27</td>
<td>0</td>
<td>9282038</td>
<td>FREQUENCY</td>
</tr>
<tr>
<td>DEPTNO</td>
<td>5</td>
<td>0</td>
<td>9282038</td>
<td>NONE</td>
</tr>
<tr>
<td>COMM</td>
<td>9</td>
<td>6398293</td>
<td>2883745</td>
<td>NONE</td>
</tr>
<tr>
<td>etc...</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

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
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
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
EXECUTION PLAN

```
PLAN hash value: 1244828764

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

Predicate Information (identified by operation id):
1 - filter(TO_DATE(:BE:DATE)<=TO_DATE(:END_DATE))
4 - access("R"."CLASS_ID"="C"."CLASS_ID")
5 - filter("C"."NAME"='SQL TUNING')
6 - filter("R"."SIGNUP_DATE">=BEGIN_DATE AND "R"."SIGNUP_DATE"<=END_DATE AND "R"."CANCELLATION"='N')
7 - access("R"."STUDENT_ID"='S"."STUDENT_ID")
```

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'
```
FREE - Oracle SQL Developer Data Modeler
Recommendations – 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");
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
```
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
NEW EXECUTION PLAN

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

<table>
<thead>
<tr>
<th>Id</th>
<th>Operation</th>
<th>Name</th>
<th>Rows</th>
<th>Bytes</th>
<th>Cost (%CPU)</th>
<th>Time</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>SELECT STATEMENT</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>*</td>
<td>FILTER</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>1</td>
<td>NESTED LOOPS</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>2</td>
<td>HASH JOIN</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>*</td>
<td>TABLE ACCESS BY INDEX ROWID</td>
<td>CLASS</td>
<td>77</td>
<td>9009</td>
<td>5569 (100)</td>
<td>00:01:07</td>
</tr>
<tr>
<td>*</td>
<td>INDEX RANGE SCAN</td>
<td>CL_NAME</td>
<td>77</td>
<td>6391</td>
<td>5492 (1)</td>
<td>00:01:06</td>
</tr>
<tr>
<td>*</td>
<td>TABLE ACCESS FULL</td>
<td>REGISTRATION</td>
<td>97637</td>
<td>1716K</td>
<td>5489 (1)</td>
<td>00:01:06</td>
</tr>
<tr>
<td>*</td>
<td>INDEX UNIQUE SCAN</td>
<td>PK_STUDENT</td>
<td>1</td>
<td>65</td>
<td>2 (0)</td>
<td>00:00:01</td>
</tr>
<tr>
<td>*</td>
<td>TABLE ACCESS BY INDEX ROWID</td>
<td>STUDENT</td>
<td>1</td>
<td>34</td>
<td>1 (0)</td>
<td>00:00:01</td>
</tr>
</tbody>
</table>

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?
CLASS_ID not left leading in index

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

» CREATE INDEX reg_alt ON registration(class_id);

```sql
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

<table>
<thead>
<tr>
<th>Id</th>
<th>Operation</th>
<th>Name</th>
<th>Rows</th>
<th>Bytes</th>
<th>Cost (%CPU)</th>
<th>Time</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>SELECT STATEMENT</td>
<td>FILTER</td>
<td></td>
<td></td>
<td>1470 (100)</td>
<td></td>
</tr>
<tr>
<td>1</td>
<td>NESTED LOOPS</td>
<td>FILTER</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>3</td>
<td>NESTED LOOPS</td>
<td>FILTER</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>4</td>
<td>NESTED LOOPS</td>
<td>FILTER</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>5</td>
<td>TABLE ACCESS BY INDEX ROWID</td>
<td>CLASS</td>
<td>66</td>
<td>7722</td>
<td>1470 (0)</td>
<td>0:00:18</td>
</tr>
<tr>
<td>6</td>
<td>INDEX RANGE SCAN</td>
<td>CLASS</td>
<td>66</td>
<td>5478</td>
<td>1404 (0)</td>
<td>0:00:17</td>
</tr>
<tr>
<td>7</td>
<td>TABLE ACCESS BY INDEX ROWID</td>
<td>CL_NAME</td>
<td>1</td>
<td>65</td>
<td>2 (0)</td>
<td>0:00:01</td>
</tr>
<tr>
<td>8</td>
<td>INDEX RANGE SCAN</td>
<td>CL_NAME</td>
<td>1</td>
<td>65</td>
<td>2 (0)</td>
<td>0:00:01</td>
</tr>
<tr>
<td>9</td>
<td>INDEX UNIQUE SCAN</td>
<td>REG_ALT</td>
<td>66</td>
<td>1188</td>
<td>1402 (0)</td>
<td>0:00:17</td>
</tr>
<tr>
<td>10</td>
<td>TABLE ACCESS BY INDEX ROWID</td>
<td>STUDENT</td>
<td>1407</td>
<td></td>
<td>3 (0)</td>
<td>0:00:01</td>
</tr>
<tr>
<td>11</td>
<td>TABLE ACCESS BY INDEX ROWID</td>
<td>STUDENT</td>
<td>1</td>
<td>34</td>
<td>1 (0)</td>
<td>0:00:01</td>
</tr>
</tbody>
</table>

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'>=BEGIN_DATE AND 'R'.'SIGNUP_DATE'<=END_DATE AND 'R'.'CANCELLED'='N')
8 - access('P'.'CLASS_ID'='C'.'CLASS_ID')
9 - access('R'.'STUDENT_ID'='S'.'STUDENT_ID')

» Execution Stats – 3000 Buffer Gets / Average Execs - .008 Secs
CREATE INDEX reg_cancel_signup ON registration(cancelled,signup_date);

Execution Stats:
1107 Buffer Gets
Avg Executions: 0.14 Secs
 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.sign-up-date BETWEEN :beg_date and :end_date AND r.cancelled = 'N'

Plan hash value: 3574817656

<table>
<thead>
<tr>
<th>Id</th>
<th>Operation</th>
<th>Name</th>
<th>Rows</th>
<th>Bytes</th>
<th>Cost (%CPU)</th>
<th>Time</th>
</tr>
</thead>
<tbody>
<tr>
<td>*</td>
<td>SELECT STATEMENT</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>1</td>
<td>FILTER</td>
<td></td>
<td></td>
<td></td>
<td>186 (100)</td>
<td></td>
</tr>
<tr>
<td>2</td>
<td>NESTED LOOPS</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>3</td>
<td>NESTED LOOPS</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>4</td>
<td>NESTED LOOPS</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>5</td>
<td>TABLE ACCESS BY INDEX ROWID</td>
<td>CLASS</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>6</td>
<td>INDEX RANGE SCAN</td>
<td>CL_NAME</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>7</td>
<td>TABLE ACCESS BY INDEX ROWID</td>
<td>REGISTRATION</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>8</td>
<td>INDEX RANGE SCAN</td>
<td>REG_ALT</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>9</td>
<td>INDEX UNIQUE SCAN</td>
<td>PK_STUDENT</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>10</td>
<td>TABLE ACCESS BY INDEX ROWID</td>
<td>STUDENT</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

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
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
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
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
### EXECUTION PLAN

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

**SQL_ID** 2g7vydk4ng7an, child number 0

---

```sql
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

<table>
<thead>
<tr>
<th>Id</th>
<th>Operation</th>
<th>Name</th>
<th>Rows</th>
<th>Bytes</th>
<th>TempSp</th>
<th>Cost (%)</th>
<th>Time</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>SELECT STATEMENT</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>1806 (100)</td>
<td></td>
</tr>
<tr>
<td>* 1</td>
<td>HASH JOIN</td>
<td>LOC</td>
<td>4537</td>
<td>239K</td>
<td>1806</td>
<td>(2)</td>
<td>00:00:22</td>
</tr>
<tr>
<td>* 2</td>
<td>TABLE ACCESS FULL</td>
<td>DEPT</td>
<td>27</td>
<td>189K</td>
<td>1799</td>
<td>(2)</td>
<td>00:00:22</td>
</tr>
<tr>
<td>* 3</td>
<td>HASH JOIN</td>
<td>EMP</td>
<td>4579</td>
<td>102K</td>
<td>753</td>
<td>(1)</td>
<td>00:00:10</td>
</tr>
<tr>
<td>* 4</td>
<td>TABLE ACCESS FULL</td>
<td>PK_EMP</td>
<td>54784</td>
<td>2408K</td>
<td>1046</td>
<td>(2)</td>
<td>00:00:13</td>
</tr>
<tr>
<td>* 5</td>
<td>MERGE JOIN SEMI</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>802</td>
<td>(3)</td>
</tr>
<tr>
<td>* 6</td>
<td>TABLE ACCESS BY INDEX ROWID</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>802</td>
<td>(3)</td>
</tr>
</tbody>
</table>

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)
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.
```sql
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
```
**NEW EXECUTION PLAN**

» CREATE INDEX ix_last_name ON emp(last_name);

```sql
SQL_ID 2g7\vd4ng7a, 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

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

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
NEW EXECUTION PLAN

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

```sql
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

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

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

Plan hash value: 2485762199

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

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"=:81)
13 - access("C","CUSTOMER_ID"="0","CUSTOMER_ID")
15 - access("I","ORDER_ID"="0","ORDER_ID")
16 - access("P","PRODUCT_ID"="I","PRODUCT_ID")
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
NEW EXECUTION PLAN

CREATE INDEX ix_cust_last_name ON customers (cust_last_name);

Plan hash value: 1275669193

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

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"."CUSTOM_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
**BEST EXECUTION PLAN**

» CREATE INDEX ix_product ON inventories (product_id);

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

Predicate Information (identified by operation id):

```
6 - access("C","CUST_LAST_NAME"=":B1")
7 - filter("O","ORDER_STATUS"=O 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**
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
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!