What the SQL is Going On Out Here?
About Dan Hotka:

Dan Hotka is a Director of Database Field Operations for Quest Software. He has over 21 years in the computer industry and over 16 years experience with Oracle products. He is an acknowledged Oracle expert with Oracle experience dating back to the Oracle V4.0 days. He has co-authored the popular books Oracle Unleashed, Oracle8 Server Unleashed, Oracle Development Unleashed by SAMS and Special Edition using Oracle8i by Que, is frequently published in trade journals, and regularly speaks at Oracle conferences and user groups around the world. Dan can be reached at dhotka@earthlink.net or dhotka@quest.com.
Our Mission

To enable today’s businesses to achieve 24x7 operation of mission-critical applications
The eBusiness Infrastructure Landscape

Hardware/Network
- HP
- Cisco
- Sun
- EMC
- Veritas
- Legato

Operating System
- CA
- Tivoli
- BMC

Application/Database
- Develop/Deploy
- Manage

Apps
- Quest Software

DB
- Quest Software
The Complete Quest Solution

DEVELOP/DEPLOY

DB Server Development
DB Change Management

Database Performance Management

DB & Application Monitoring
Application Offloading
High Availability

MANAGE
Agenda

- SQL Issues
- Tuning Methodology
- How to find Poorly Performing SQL
- The Oracle Optimizers
- Understanding the Explain Plan
- Oracle Tuning Tools
- SQL Do’s and Don’ts
SQL Issues

- SQL Rage!
  - 20% of SQL is consuming 80% of the resources
  - Poorly performing SQL statements infuriates the staff!
Where Are These SQL Statements Coming From?

- SQL, while easy to understand conceptually, can be difficult to grasp as it relates to performance.
- Biggest problem, non-trained application programmers and/or end-users are expected to deliver highly tuned SQL.
- High demand for new applications & RAD techniques causes sloppiness.
- Not enough time or resources to examine what the SQL is really doing.
Tuning Methodologies
Top-down & Bottom-up

- **Top Down**
  - Reacting to problem SQL
  - End users using 3rd party tools
  - No tuning at the development level
  - No review of code
Tuning Methodologies
Top-down & Bottom-up

- Bottom-up
  - Build it right the first time
  - Know the data and design
  - Quickly evaluate alternatives - rule, first, or all
  - Add hints based on database stats
  - Execute & compare
  - Determine which plans are affected by a database change
Tuning Methodology

- Operating System
- Oracle Database
- App Design
- SQL Code
Tuning Methodology
In Action

- **Advice**
  - Improve the table access for the primary select
    - Use semi-join
    - Add HASH_SJ hint
  - Use different tablespaces for indexes and tables
    - Modify the tablespace of index SCOTT.PK_EMP to something other than USER_DATA
  - Update Statistics
    - Re-analyze table SCOTT.EMP
    - Re-analyze table SCOTT.DEPT
  - Create indexes on foreign keys
    - Create an index on table SCOTT.EMP on column (DEPTNO)
How to find Poorly Performing SQL

- Monitoring
- Scripts
  - Tim Gorman www.sagelogix.com
  - Top Offensive SQL Statements
    - Oracle Professional June 2000
- Tools
- Luck...
How to find Poorly Performing SQL
How to find Poorly Performing SQL

### Top Sessions

<table>
<thead>
<tr>
<th>SID</th>
<th>Oracle User</th>
<th>Status</th>
<th>Machine</th>
<th>OS User</th>
<th>Log read/s</th>
<th>Disk read/s</th>
<th>CPU (ms/s)</th>
<th>Logon</th>
</tr>
</thead>
<tbody>
<tr>
<td>2</td>
<td>SYS (DBW0)</td>
<td>ACTIVE</td>
<td>DHOTKA</td>
<td>UNKNOWN</td>
<td>0.00</td>
<td>0.00</td>
<td>0.00</td>
<td>31-MAY-00 20:10:06</td>
</tr>
<tr>
<td>3</td>
<td>SYS (LGWR)</td>
<td>ACTIVE</td>
<td>DHOTKA</td>
<td>UNKNOWN</td>
<td>0.00</td>
<td>0.00</td>
<td>0.00</td>
<td>31-MAY-00 20:10:06</td>
</tr>
<tr>
<td>4</td>
<td>SYS (CKPT)</td>
<td>ACTIVE</td>
<td>DHOTKA</td>
<td>UNKNOWN</td>
<td>0.00</td>
<td>0.00</td>
<td>0.00</td>
<td>31-MAY-00 20:10:06</td>
</tr>
<tr>
<td>5</td>
<td>SYS (SMON)</td>
<td>ACTIVE</td>
<td>DHOTKA</td>
<td>UNKNOWN</td>
<td>0.00</td>
<td>0.00</td>
<td>0.00</td>
<td>31-MAY-00 20:10:06</td>
</tr>
<tr>
<td>6</td>
<td>SYS (RECO)</td>
<td>ACTIVE</td>
<td>DHOTKA</td>
<td>UNKNOWN</td>
<td>0.00</td>
<td>0.00</td>
<td>0.00</td>
<td>31-MAY-00 20:10:06</td>
</tr>
<tr>
<td>7</td>
<td>ORACLE80....</td>
<td>ACTIVE</td>
<td>(SNP0)</td>
<td></td>
<td>0.00</td>
<td>0.00</td>
<td>0.00</td>
<td>31-MAY-00 20:27:46</td>
</tr>
<tr>
<td>8</td>
<td>ORACLE80....</td>
<td>ACTIVE</td>
<td>(SNP1)</td>
<td></td>
<td>0.00</td>
<td>0.00</td>
<td>0.00</td>
<td>31-MAY-00 20:27:46</td>
</tr>
<tr>
<td>9</td>
<td>SCOTT</td>
<td>INACTIVE</td>
<td>SALES</td>
<td>DHOTKA</td>
<td>DHotka</td>
<td>0.00</td>
<td>0.00</td>
<td>31-MAY-00 20:12:40</td>
</tr>
<tr>
<td>14</td>
<td>SCOTT</td>
<td>INACTIVE</td>
<td>SALES</td>
<td>DHOTKA</td>
<td>DHotka</td>
<td>0.00</td>
<td>0.00</td>
<td>31-MAY-00 20:15:37</td>
</tr>
</tbody>
</table>

**Oracle SID:** 14  
**Username:** SCOTT  
**Server:** d:\orant\bin\oracle90.exe  
**CPU utilization (ms):** 240  
**Block gets:** 52  
**Consistent gets:** 502  
**Block changes:** 1  
**Status:** INACTIVE  
**Program:** C:\WINNT\Profiles\All Users\Start Menu\PROGRAMS\Oracle for Windo  
**Waiting for:** SQL*Net message from client driver id=675562835 bytes=1  
**Current SQL:**
```
select ename from emp where deptno in (select deptno from dept where deptno = 10)
```
How to find Poorly Performing SQL

- Diagnoses Source of I/O Bottlenecks:
  - Poor SQL
  - Poor Layout
- Quickly identify disk hot spots
- Drill down to actual SQL statements
- Identify frequently accessed objects
- Soon: History, I/O Trends, Capacity Planning
<table>
<thead>
<tr>
<th>Name</th>
<th>Total</th>
<th>Rate</th>
<th>PeakRate</th>
<th>Type</th>
<th>Owner</th>
<th>Tablespace</th>
<th>Blocks</th>
<th>Extents</th>
</tr>
</thead>
<tbody>
<tr>
<td>TEST_OBJECTS</td>
<td>94321</td>
<td>413.50</td>
<td>586.04</td>
<td>TABLE</td>
<td>SYSTEM</td>
<td>TOOLS</td>
<td>4805</td>
<td>16</td>
</tr>
<tr>
<td>C_T#</td>
<td>12</td>
<td>0.02</td>
<td>1.00</td>
<td>CLUSTER</td>
<td>SYS</td>
<td>SYSTEM</td>
<td>10</td>
<td>2</td>
</tr>
<tr>
<td>I_FILE##_BLOCK##</td>
<td>5</td>
<td>0.02</td>
<td>0.60</td>
<td>INDEX</td>
<td>SYS</td>
<td>SYSTEM</td>
<td>35</td>
<td>4</td>
</tr>
<tr>
<td>C_FILE##_BLOCK##</td>
<td>8</td>
<td>0.01</td>
<td>0.60</td>
<td>CLUSTER</td>
<td>SYS</td>
<td>SYSTEM</td>
<td>200</td>
<td>8</td>
</tr>
<tr>
<td>FILE$</td>
<td>2</td>
<td>0.01</td>
<td>0.20</td>
<td>TABLE</td>
<td>SYS</td>
<td>SYSTEM</td>
<td>5</td>
<td>1</td>
</tr>
</tbody>
</table>
How to find Poorly Performing SQL
How to find Poorly Performing SQL

1: /********************************************************************
2: * File: top_stmt2.sql
3: * Type: SQL*Plus script
4: * Author: Tim Gorman (SageLogix, Inc.)
5: * Date: 04-Oct-99
6: *
7: * Description:
8: * DDL script to create the TOP_STMT2 stored procedure.
9: *
10: * Modifications:
11: *******************************************************************/
12:
13: create or replace procedure top_stmt2
14: ( 
15: in_top_count in number default 20,
16: in_max_disk_reads in number default 10000,
17: in_max_buffer_gets in number default 100000
18: ) is
19: --
20: cursor get_top_stmts(in_dr in number, in_bg in number)
21: is
22: select /*+ rule */
23: substr(sql_text, 1, 60) sql_text,
24: min(address) address,
25: sum(abs(disk_reads)) disk_reads,
26: sum(abs(buffer_gets)) buffer_gets,
27: sum(abs sorts) sorts,
28: sum(abs executions) executions,
29: sum(abs loads) loads,
30: count(*) cnt,
31: ((sum(abs(disk_reads)) *100)+sum(abs(buffer_gets))))/1000
32: factor
33: from sys.v_$sqlarea
34: group by substr(sql_text, 1, 60)
35: having sum(abs(disk_reads)) > in_dr
36: and sum(abs(buffer_gets)) > in_bg
37: order by factor desc:
**Tuning in Production: Recon**

- **Historical 24x7 information**
  - From a minute to a full week--intuitive navigation
  - Detailed info on SQL, users, programs and clients
  - Detailed resource consumption and resource waits
  - Analysis graphs and direct entry into SQL tuning

- **Live Diagnostics**
  - What sessions are active/inactive and what are they doing NOW
  - Trace session with full SQL history and detailed stats per SQL

- **Zero Oracle overhead, negligible server overhead**
Historical Analysis and Diagnostics

(a) View workload breakdown for last hour, day, week...

(b) Select interesting time frame

(c) Select Grid view
(d) View the SQL statements using resources at 1:39PM…

(e) Or check the programs that were running at the time…
Historical Analysis and Diagnostics: Drilldowns

Easily identify the statement (or program) waiting for locks

The same is available for CPU, I/O, Network, Latches and Other Waits
Live Diagnostics

Sessions, SQL currently executed, resource consumption and colored thresholds...

Available indicators

Current workload distribution

No Oracle connection, Zero Overhead
Live Diagnostics Drilldowns

SQL statements executed by session

Comprehensive statistics for the session

Breakdown of session activity, refreshed every second
OnWire’s Non-Intrusive Solution
HostRecorder Collector

- Implemented as a Kernel Streams driver
  - NO Overhead
    - No context switching
    - No expensive User mode processing
    - No polling
    - No contention
    - No agents
  - 10 minute install (< 25K size)
  - No network overhead
OnWire 3.1 Features

- HostRecorder
- Built-in Forms End-User Transactions
- Concurrent Manager Support
- Advanced Locking Diagnostics
- Support for Web and PLSQL Table Arrays
- Normalized Performance Metrics
- Identify and classify resource utilization
- Full Lifecycle Tuning Support w/SQLAB
Oracle Optimizer gives you choices:

- **Rule-based**
  - Based on a set of rules (Index existence, SQL coding)
  - Does NOT consider object statistics

- **Cost-based**
  - Uses object statistics (from ANALYZE command)
  - User has more control in tuning
  - 2 goals:
    - First row: response time for interactive apps (OLTP)
    - All rows: throughput for batch processing (DSS)
  - Makes Assumptions

- **Hints**
  - Cost Based ‘suggestions’, not always used by Oracle!
  - Can be specified in combinations
The Oracle Optimizers
Selecting a Mode

- **Database Level**
  - init.ora OPTIMIZER_MODE parameter
  - RULE, COST, or CHOOSE* (default)

- **Session Level**
  - ALTER SESSION SET OPTIMIZER_GOAL= < >
    - < > = RULE, FIRST_ROWS, ALL_ROWS, CHOOSE

- **SQL Level**
  - HINT RULE, FIRST_ROWS, ALL_ROWS
The Oracle Optimizers
Rule-based Optimizer

<table>
<thead>
<tr>
<th>Rank</th>
<th>Where Clause Rule</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>ROWID = constant</td>
</tr>
<tr>
<td>2</td>
<td>unique indexed column = constant</td>
</tr>
<tr>
<td>3</td>
<td>entire unique concatenated index = constant</td>
</tr>
<tr>
<td>4</td>
<td>entire cluster key = cluster key of object in same cluster</td>
</tr>
<tr>
<td>5</td>
<td>entire cluster key = constant</td>
</tr>
<tr>
<td>6</td>
<td>entire nonunique concatenated index = constant</td>
</tr>
<tr>
<td>7</td>
<td>nonunique index = constant</td>
</tr>
<tr>
<td>8</td>
<td>entire noncompressed concatenated index &gt;= constant</td>
</tr>
<tr>
<td>9</td>
<td>entire compressed concatenated index &gt;= constant</td>
</tr>
<tr>
<td>10</td>
<td>partial but leading columns of noncompressed concatenated index</td>
</tr>
<tr>
<td>11</td>
<td>partial but leading columns of compressed concatenated index</td>
</tr>
<tr>
<td>12</td>
<td>unique indexed column using the SQL statement BETWEEN or LIKE options</td>
</tr>
<tr>
<td>13</td>
<td>nonunique indexed column using the SQL statement BETWEEN or LIKE options</td>
</tr>
<tr>
<td>14</td>
<td>unique indexed column &lt; or &gt; constant</td>
</tr>
<tr>
<td>15</td>
<td>nonunique indexed column &lt; or &gt; constant</td>
</tr>
<tr>
<td>16</td>
<td>sort/merge</td>
</tr>
<tr>
<td>17</td>
<td>MAX or MIN SQL statement functions on indexed column</td>
</tr>
<tr>
<td>18</td>
<td>ORDER BY entire index</td>
</tr>
<tr>
<td>19</td>
<td>full table scans</td>
</tr>
</tbody>
</table>
The Oracle Optimizers
Cost-based Optimizer

Oracle hints include:

/*+ALL_ROWS*/ Optimize SQL for best throughput
/*+AND_EQUAL*/ Use index merging on specified tables
/*+CLUSTER*/ Use a cluster scan for a specified table
/*+COST*/ Use cost-based optimizer always
/*+FIRST_ROWS*/ Optimize SQL for best response times
/*+FULL*/ Use a full-table scan
/*+HASH*/ Use a hash-search method
/*+INDEX*/ Force the use of a specified index
/*+STAR*/ Force Star join, between a large table with
concatenated keys and smaller tables
/*+ORDERED*/ Use the from clause join sequence
/*+ROWID*/ Use ROWID access method
/*+USE_MERGE*/ Use sort merge join technique
/*+USE_NL*/ Use nested loop join technique
/*+USE_NOCACHE*/ Don’t put the data in the buffers
/*+USE_HASH*/ Use a hash join
/*+USE_CONCAT*/ Use multiple indexes for or conditions

& MORE
Helpful INIT.ORA Parameters

- Database File Multi Block Read Count
  - Both rule and cost
- Hash Multi Block IO Count
- Hash Join Enabled
- Hash Area Size
  - Cost only
- Sort Area Size
  - Both rule and cost
Helpful 8i INIT.ORA settings

- Optimizer Index Caching
- Optimizer Index Cost Adjustment
  - adjusts for low hit ratio assumption
Index Usage

- **Rule Optimizer**
  - Follows rules
  - Lower clustering factor, better rule selection
  - No functions in index
  - Use a function to NOT use an index

- **Cost Optimizer**
  - Follows stats
  - 8i supports function-based indexes
  - can use HINTS

Where `comm * 1.1 > 1000` vs

Where `comm > 1000/1.1`
Clustering Factor

- Is the relationship between the number of index blocks vs the number of related data blocks
- Low is good.
- Lower Clustering Factors:
  - Sort data into index order prior to loading
  - Use primary key (and in sorted order again)
The Oracle Optimizers
Cost-based Optimizer
Cost-based Assumptions

- Can use Cost Optimizer even if no stats
- Assumes even data distribution
- Assumes certain row counts
- Assumes low buffer hit ratio
- Assumes lots of users
Cost Optimizer Statistics

- Analyze <object> compute statistics
  - Collects:
    - Number of blocks
    - Number of rows
    - Indexes:
      - Granularity of Indexes (clustering factor)
      - Distinct values
      - Number of blocks per leaf
    - Histograms - if uneven data distribution
Understanding Explain Plans

- Is a Necessity to Tuning SQL Statements
- Shows you the choices made by either optimizer
- Can be difficult to interpret
- Indenting and tools greatly aid!
Understanding Explain Plans
Driving Table

- **Rule**
  - Based on existence of indexes or LAST table in FROM clause or first NESTED SELECT

- **COST - FIRST ROWS**
  - Based on unique indexes
  - Tries to avoid Full Scans and Sorts

- **COST - ALL ROWS**
  - Based on total rows returned

- Cost - Ordered Hint
Understanding Explain Plans Syntax

SQL>
SQL> EXPLAIN PLAN FOR
  2  select ename
  3  from emp
  4  where deptno in (select deptno from dept where deptno = 10);

Explained.

SQL> SELECT operation, options, object_name, id, parent_id
  2  from plan_table;

<table>
<thead>
<tr>
<th>OPERATION</th>
<th>OPTIONS</th>
<th>OBJECT_NAME</th>
<th>ID</th>
<th>PARENT_ID</th>
</tr>
</thead>
<tbody>
<tr>
<td>SELECT STATEMENT</td>
<td></td>
<td></td>
<td>0</td>
<td></td>
</tr>
<tr>
<td>NESTED LOOPS</td>
<td></td>
<td></td>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td>INDEX</td>
<td>UNIQUE SCAN</td>
<td>PK_DEPT</td>
<td>2</td>
<td>1</td>
</tr>
<tr>
<td>TABLE ACCESS</td>
<td>FULL</td>
<td>EMP</td>
<td>3</td>
<td>1</td>
</tr>
</tbody>
</table>
## Explain Symbol Description

<table>
<thead>
<tr>
<th>Symbol</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>AND-EQUAL</td>
<td>Index values will be used to join rows.</td>
</tr>
<tr>
<td>CONCATENATION</td>
<td>SQL statement UNION command.</td>
</tr>
<tr>
<td>FILTER</td>
<td>FILTERs apply ‘other criteria’ in the query to further qualify the matching rows. The ‘other criteria’ include correlated subqueries, and HAVING clause.</td>
</tr>
<tr>
<td>FIRST ROW</td>
<td>SQL statement will be processed via a cursor.</td>
</tr>
<tr>
<td>FOR UPDATE</td>
<td>SQL statement clause ‘for update of’ placed row level locks on affected rows.</td>
</tr>
<tr>
<td>INDEX (UNIQUE)</td>
<td>SQL statement utilized a unique index to search for a specific value.</td>
</tr>
<tr>
<td>INDEX (RANGE SCAN)</td>
<td>SQL statement contains a nonequality or BETWEEN condition.</td>
</tr>
<tr>
<td>HASH JOIN</td>
<td>SQL statement initiated a hash-join operation.</td>
</tr>
<tr>
<td>MERGE JOIN</td>
<td>SQL statement references two or more tables, sorting the two result sets being joined over the join columns and then merging the results via the join columns.</td>
</tr>
<tr>
<td>NESTED LOOPS</td>
<td>This operation is one form of joining tables. One row is retrieved from the row source identified by the first (inner) operation, and then joined to all matching rows in the other table (outer).</td>
</tr>
<tr>
<td>NONUNIQUE INDEX (RANGE SCAN)</td>
<td>The RANGE SCAN option indicates that ORACLE expects to return multiple matches (ROWIDs) from the index search</td>
</tr>
<tr>
<td>PARTITION (CONCATENATED)</td>
<td>SQL statement will access a partitioned object and merge the retrieved rows from the accessed partitions.</td>
</tr>
<tr>
<td>PARTITION (SINGLE)</td>
<td>SQL statement will access a single partition.</td>
</tr>
<tr>
<td>PARTITION (EMPTY)</td>
<td>The SQL statement makes reference to an empty partition.</td>
</tr>
<tr>
<td>SORT (ORDER BY)</td>
<td>SQL statement contains an ORDER BY SQL command.</td>
</tr>
<tr>
<td>SORT (AGREGATE)</td>
<td>SQL statement initiated a sort to resolve a MIN or MAX function.</td>
</tr>
<tr>
<td>SORT (GROUP BY)</td>
<td>SQL statement contains a GROUP BY SQL command.</td>
</tr>
<tr>
<td>TABLE ACCESS (FULL)</td>
<td>All rows are retrieved from the table without using an index.</td>
</tr>
<tr>
<td>TABLE ACCESS (BY ROWID)</td>
<td>A row is retrieved based on ROWID</td>
</tr>
<tr>
<td>TABLE ACCESS (CLUSTER)</td>
<td>A row is retrieved from a table that is part of a cluster.</td>
</tr>
<tr>
<td>UNION</td>
<td>SQL statement contains a DISTINCT SQL command.</td>
</tr>
</tbody>
</table>
Understanding Explain Plans

Syntax

- Nested Loop Join
  - driving table
  - Default order (rule)

- Merge Scan Join
  - sort & match

- Hash Join (7.3)
  - Full scans with no sorts
  - Join column to row address
Understanding Explain Plans

- Nested Loop Join
  - small portion accessed from a large table & joined from a small portion of the second table

- Merge Scan Join
  - large portion of rows are being joined

- Hash Join
  - large portion with a lot of memory
Understanding Explain Plans

### CPU Time (sec) - Parse & Execute

<table>
<thead>
<tr>
<th>Percentage</th>
<th>Nested Loop</th>
<th>Merge Join</th>
<th>Hash Join</th>
</tr>
</thead>
<tbody>
<tr>
<td>100</td>
<td>103.26</td>
<td>65.63</td>
<td>25.06</td>
</tr>
<tr>
<td>50</td>
<td>79.26</td>
<td>54.34</td>
<td>19.16</td>
</tr>
<tr>
<td>25</td>
<td>39.01</td>
<td>45.85</td>
<td>12.40</td>
</tr>
<tr>
<td>10</td>
<td>15.22</td>
<td>42.26</td>
<td>8.99</td>
</tr>
<tr>
<td>5</td>
<td>8.10</td>
<td>34.33</td>
<td>8.50</td>
</tr>
<tr>
<td>3</td>
<td>4.54</td>
<td>34.25</td>
<td>7.95</td>
</tr>
<tr>
<td>2</td>
<td>3.15</td>
<td>33.47</td>
<td>7.63</td>
</tr>
<tr>
<td>1</td>
<td>1.74</td>
<td>33.59</td>
<td>7.60</td>
</tr>
</tbody>
</table>
Understanding Explain Plans

Tips

- Use bind variables
  - Helps SQL parsing/processing

- Merging Indexes
  - Oracle will process up to 5
  - Use AND-EQUAL hint to select and limit

- Nested Selects vs Joins vs UNION ALL
  - Use UNION ALL syntax where possible
  - Cost optimizer might change NS to Joins
Oracle Tuning Tools

- Explain Table
- TKPROF
- GUI Tools
  - SQL Navigator
  - TOAD
  - OEM SQL Analyze
  - SQLab
Oracle Tuning Tools
Explain Plan

SQL>
SQL> EXPLAIN PLAN FOR
2  select ename
3   from emp
4  where deptno in (select deptno from dept where deptno = 10);

Explained.

SQL> SELECT operation, options, object_name, id, parent_id
2  from plan_table;

<table>
<thead>
<tr>
<th>OPERATION</th>
<th>OPTIONS</th>
<th>OBJECT_NAME</th>
<th>ID</th>
<th>PARENT_ID</th>
</tr>
</thead>
<tbody>
<tr>
<td>SELECT STATEMENT</td>
<td></td>
<td></td>
<td>0</td>
<td></td>
</tr>
<tr>
<td>NESTED LOOPS</td>
<td></td>
<td></td>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td>INDEX</td>
<td>UNIQUE SCAN</td>
<td>PK_DEPT</td>
<td>2</td>
<td>1</td>
</tr>
<tr>
<td>TABLE ACCESS</td>
<td>FULL</td>
<td>EMP</td>
<td>3</td>
<td>1</td>
</tr>
</tbody>
</table>
Oracle Tuning Tools
TKPROF

```sql
select * from tutorial.cur_emp_status
```

<table>
<thead>
<tr>
<th>call</th>
<th>count</th>
<th>cpu</th>
<th>elapsed</th>
<th>disk</th>
<th>query</th>
<th>current</th>
<th>rows</th>
</tr>
</thead>
<tbody>
<tr>
<td>Parse</td>
<td>1</td>
<td>0.02</td>
<td>0.00</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>Execute</td>
<td>1</td>
<td>0.00</td>
<td>0.00</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>Fetch</td>
<td>2</td>
<td>0.23</td>
<td>0.25</td>
<td>11</td>
<td>1051</td>
<td>3</td>
<td>15</td>
</tr>
</tbody>
</table>

| total  | 4     | 0.25 | 0.27    | 11   | 1051  | 3       | 15   |
## Oracle Tuning Tools - TKPROF

<table>
<thead>
<tr>
<th>Rows</th>
<th>Execution Plan</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td><code>SELECT STATEMENT</code> HINT: CHOOSE</td>
</tr>
<tr>
<td>137</td>
<td><code>FILTER</code></td>
</tr>
<tr>
<td>137</td>
<td><code>NESTED LOOPS</code></td>
</tr>
<tr>
<td>137</td>
<td><code>NESTED LOOPS</code></td>
</tr>
<tr>
<td>137</td>
<td><code>NESTED LOOPS</code></td>
</tr>
<tr>
<td>137</td>
<td><code>NESTED LOOPS</code></td>
</tr>
<tr>
<td>49</td>
<td><code>NESTED LOOPS</code></td>
</tr>
<tr>
<td>23</td>
<td><code>NESTED LOOPS</code></td>
</tr>
<tr>
<td>15</td>
<td><code>TABLE ACCESS</code> HINT: ANALYZED (FULL) OF 'EMPLOYEES'</td>
</tr>
<tr>
<td>23</td>
<td><code>TABLE ACCESS</code> HINT: ANALYZED (CLUSTER) OF 'SAL_HISTORY'</td>
</tr>
<tr>
<td>137</td>
<td><code>TABLE ACCESS</code> HINT: ANALYZED (BY ROWID) OF 'JOB_CODES'</td>
</tr>
<tr>
<td>137</td>
<td><code>INDEX</code> HINT: ANALYZED (UNIQUE SCAN) OF 'I_JOBS' (UNIQUE)</td>
</tr>
<tr>
<td>137</td>
<td><code>TABLE ACCESS</code> HINT: ANALYZED (BY ROWID) OF 'DEPARTMENTS'</td>
</tr>
<tr>
<td>137</td>
<td><code>INDEX</code> HINT: ANALYZED (UNIQUE SCAN) OF 'I_DEPTS' (UNIQUE)</td>
</tr>
<tr>
<td>23</td>
<td><code>SORT (AGGREGATE)</code></td>
</tr>
<tr>
<td>23</td>
<td><code>TABLE ACCESS</code> HINT: ANALYZED (CLUSTER) OF 'DEPT_HISTORY'</td>
</tr>
<tr>
<td>15</td>
<td><code>INDEX</code> HINT: ANALYZED (UNIQUE SCAN) OF 'I_EMP_EMPNO' (CLUSTER)</td>
</tr>
<tr>
<td>23</td>
<td><code>SORT (AGGREGATE)</code></td>
</tr>
<tr>
<td>23</td>
<td><code>TABLE ACCESS</code> HINT: ANALYZED (CLUSTER) OF 'JOB_HISTORY'</td>
</tr>
<tr>
<td>15</td>
<td><code>INDEX</code> HINT: ANALYZED (UNIQUE SCAN) OF 'I_EMP_EMPNO' (CLUSTER)</td>
</tr>
</tbody>
</table>
Oracle Tools - SQLab XPert
Oracle Tools - SQLab XPert
Oracle Tools - SQLab XPert

- **Advice**
  - Improve the table access for the primary select
  - Use semi-join
    - Add HASH SJ hint
  - Use different tablespaces for indexes and tables
    - Modify the tablespace of index SCOTT.PK_EMP to something other than USER_DATA
  - Update Statistics
    - Re-analyze table SCOTT.EMP
    - Re-analyze table SCOTT.DEPT
  - Create indexes on foreign keys
    - Create an index on table SCOTT.EMP on column (DEPTNO)
### Oracle Tools - SQLab Xpert

**Compare Plans**

<table>
<thead>
<tr>
<th>Statement View</th>
<th>Comparison View</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>5: Add Join Criteria (Trace Statistics)</strong></td>
<td><strong>3: First_Rows Hint (Trace Statistics)</strong></td>
</tr>
</tbody>
</table>

**5: Add Join Criteria (Trace Statistics)**

- **SELECT STATEMENT HINT: ALL_ROWS (Revised Plan)**
- **SORT ORDER BY**
  - Cost: 6
- **Rows Processed:** 0
- **SUM (q.qty * q.VALUE) desc**
- **SORT GROUP BY**
  - Cost: 6
- **Rows Processed:** 0
- **p.supplier_id**
- **HASH JOIN**
  - Cost: 3
- **Rows Processed:** 0
- **ow_order_lines to ow_parts o.part_id = p.supplier_part_id**
- **TABLE ACCESS IMPACT ow_ORDER_LINES**
  - Cost: 1
  - **Rows Processed:** 38
  - **Tablespace:** SYSTEM
  - **INDEX RANGE SCAN IMPACT:l_o_qty**
    - Cost: 1
    - **Rows Processed:** 39
    - **Tablespace:** SYSTEM
  - **TABLE ACCESS IMPACT ow_PARTS**
    - **Blocks:**
      - Cost: 0
      - **Rows Processed:** 10

**3: First_Rows Hint (Trace Statistics)**

- **SELECT STATEMENT HINT: FIRST_ROWS**
- **SORT ORDER BY**
  - Est. Rows: 4 Cost: 6
  - **Rows Processed:** 1
  - **SUM (q.qty * q.VALUE) desc**
  - **SORT GROUP BY**
    - Est. Rows: 4 Cost: 6
    - **Rows Processed:** 1
    - **p.supplier_id**
  - **HASH JOIN**
    - Est. Rows: 4 Cost: 3
    - **Rows Processed:** 17
    - **ow_order_lines to ow_parts o.part_id = p.supplier_part_id**
  - **TABLE ACCESS IMPACT ow_ORDER_LINES**
    - **Rows Processed:** 38
    - **Tablespace:** SYSTEM
    - **INDEX RANGE SCAN IMPACT:l_o_qty**
      - **Blocks:**
        - Est. Rows: 0 Cost: 1
        - **Rows Processed:** 10
        - **Tablespace:** SYSTEM
Oracle Tools - SQLab XPert
Compare Performance
Oracle Tools - Plan Stability

- Stored Outlines
  - Is a method to guarantee that a certain Execution Plan will be used for a particular SQL statement
  - Oracle can specifically create Stored Outlines -or- it can create one for each SQL statement presented
    - CREATE OR REPLACE <outline> FOR CATEGORY <category> ON <sql statement>;
    - system/session parameter CREATE_STORED_OUTLINES = <TRUE, FALSE, ‘category name’ NOOVERIDE>
  - Must have ‘CREATE ANY OUTLINE’ permissions
Oracle Tools - Plan Stability

Oracle will use a Stored Outline unless:

- system/session parameter
  USE_STORED_OUTLINES = False
- mismatch on SQL text including Hints

How it works:

- Oracle uses OL$ and OL$HINTS tables
  - see also USER_OUTLINES, USER_OUTLINE_HINTS
- Stored indefinitely unless explicitly removed
- system/session parameter
  USE_STORED_OUTLINES = TRUE or <category>
Oracle Tools - Plan Stability

- Stored Outline Management
  - Packages DROP_UNUSED, DROP_BY_CAT, UPDATE_BY_CAT
  - Moving Outlines:
    - EXP OUTLN/OUTLN FILE = <file name> TABLES = 'OL$
      'OL$HINTS' SILENT=Y [WHERE CATEGORY=<category>]
    - IMP OUTLN/OUTLN FILE=<file name> TABLES = 'OL$
      'OL$HINT' IGNORE=Y SILENT=Y
    - see p 7-32 Oracle8i Tuning Guide
Avoid using the HAVING clause.
  - Use WHERE clause
  - The HAVING statement filters selected rows only after all of the rows have been retrieved.

Use the NOT EXISTS statement in place of a NOT IN statement.

Use joins in place of EXISTS.

Use EXISTS in place of DISTINCT.
SQL Do’s and Don’ts

- **AVOID** doing calculations on indexed WHERE columns
  - the optimizer will use a full-table scan
    - Oracle8i has new function-based index feature
- Depending on the types of SQL statement issued, think about using a concatenated index.
- Avoid using **NOT** on indexed columns (precludes using an index).
- Use **WHERE** instead of **ORDER BY** when an index is used.
- **Bind variables Vs. constants**
Summary

- Try to use good database design throughout all applications
- Monitor
- Understand your options
  - Optimizer Modes
  - Join conditions
- Use GUI Tuning Tools
- Don’t Let SQL RAGE Happen to you!
Questions?

Dan Hotka is a Director of Database Field Operations for Quest Software. He has over 21 years in the computer industry and over 16 years experience with Oracle products. He is an acknowledged Oracle expert with Oracle experience dating back to the Oracle V4.0 days. He has just completed Oracle8i from Scratch by Que and has co-authored the popular books Oracle Unleashed, Oracle8 Server Unleashed, Oracle Development Unleashed by SAMS and Special Edition using Oracle8/8i by Que, is frequently published in Oracle Professional by Pinnacle Publications, and regularly speaks at Oracle conferences and user groups around the world. Dan can be reached at dhotka@earthlink.net or dhotka@quest.com.

Bibliography:

Tim Gorman, www.sagelogix.com

Top Offensive SQL Statements, Oracle Professional June 2000; A Pinnacle Publication

Oracle8i from Scratch by Dan Hotka; A Que Publication

Oracle8i SQL Tuning Guide (Oracle Doc Set)