Tuna Helper
Proven Process for SQL Tuning

Dean Richards
Senior DBA, Confio Software
Give a man a fish and you feed him for a day.
Teach a man to fish and you feed him for a lifetime.

Chinese Proverb
Who Am I?

- Senior DBA for Confio Software
  - DeanRichards@confio.com
- Current – 20+ Years in Oracle, SQL Server
- Former – 15+ Years in Oracle Consulting
- Specialize in Performance Tuning
- Review Performance of 100’s of Databases for Customers and Prospects
- Common Thread – Paralyzed by Tuning
Agenda

- Introduction
- Challenges
- Identify - Which SQL and Why
- Gather – Details about SQL
- Tune – Case Study
- Monitor – Make sure it stays tuned
- SQL Tuning Myths
SQL Tuning is Hard

This Presentation is an Introduction
- 3-5 day detailed classes are typical

Providing a Framework
- Helps develop your own processes
- There is no magic tool
- Tools cannot reliably tune SQL statements
- Tuning requires the involvement of you and other technical and functional members of team
Challenges

- SQL Tuning is Hard – did I mention that?
- 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
Identify – Which SQL

- Highest Wait Times (Ignite, AWR, etc)
- Tracing a Session / Process
- User / Batch Job Complaints
- Highest I/O (LIO, PIO)
- SQL Performing Full Table Scans
- Known Poorly Performing SQL
Who registered yesterday for SQL Tuning

SELECT s.fname, s.lname, r.signup_date
FROM student s, activeRegistrations r, class c
WHERE s.student_id = r.student_id
AND r.class_id = c.class_id
AND UPPER(c.name) = 'SQL TUNING'
AND c.class_level = 101
AND r.signup_date BETWEEN
    TRUNC(SYSDATE) AND TRUNC(SYSDATE-1)
Know the business reason for statement
  • Who registered yesterday for SQL Tuning
  • Why does the business need to know this
  • How often is the information needed
  • Who uses this information

Understand the technical aspects
  • Review ERD
  • Understand application architecture
  • Understand tables and the data (at a high level)

Understand the entire process
  • What portion of the total time is database
  • Where is it called from in the application
Identify – End-to-End Time

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
Wait Event Information

V$SESSION
SID
USERNAME
SQL_ID
PROGRAM
MODULE
ACTION
PLAN_HASH_VALUE

V$SESSION_WAIT
SID
EVENT
P1, P1RAW, P2, P2RAW, P3, P3RAW
STATE (WAITING, WAITED...)

• Oracle 10g added this info to V$SESSION

V$SQL
SQL_ID
SQL_FULLTEXT

V$SQLAREA
SQL_ID
EXECUTIONS
PARSE_CALLS
BUFFER_GETS
DISK_READS

V$SQL_PLAN
SQL_ID
PLAN_HASH_VALUE

DBA_OBJECTS
OBJECT_ID
OBJECT_NAME
OBJECT_TYPE
Which scenario is worse?

SQL Statement 1
- Executed 1000 times
- Caused 10 minutes of wait time for end user
- Waited 99% of time on “db file sequential read”

SQL Statement 2
- Executed 1 time
- Caused 10 minutes of wait time for end user
- Waited 99% on “enq: TX – row lock contention”
Break Down SQL Into Simplest Forms

- Complex SQL becomes multiple SQL
- Sub-Queries Should be Tuned Separately
- UNION’ed SQL Tuned Separately
- Get the definition of views
- Are synonyms being used
Look for obvious limiting factors
- column = :1 - (>, <, BETWEEN)
- column IN (‘A’, ‘B’) - EXISTS
- column LIKE ‘ABCD%’

Match up with existing indexes

Avoid obvious non-limiting factors
- <>, NOT LIKE, LIKE ‘%ABCD%’
Identify – Summary

- Determine the SQL
- Understand End-to-End
- Understand Database Wait Time
- Simplify Statement
- High-Level Analysis
Gather - Metrics

- Get baseline metrics
  - How long does it take now
  - What is acceptable (10 sec, 2 min, 1 hour)

- Collect Wait Time information
  - Locking / Blocking
  - I/O problem
  - Latch contention
  - Network slowdown
  - May be multiple issues
  - All have different resolutions

- Document everything in simple language
Gather - Execution Plan

- **EXPLAIN PLAN**
  - Estimated execution plan - can be wrong for many reasons

- **V$SQL_PLAN** (Oracle 9i+)
  - Real execution plan
  - Use DBMS_XPLAN for display

- **Tracing** (all versions)
  - Get all sorts of good information
  - Works when you know a problem will occur

- **Historical** - AWR, Confio Ignite
SELECT company, attribute FROM data_out WHERE segment = :B1

- Wait Time - 100% on “db file scattered read”

- Plan from EXPLAIN PLAN

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

- Plan from V$SQL_PLAN using DBMS_XPLAN

```
select * from table(dbms_xplan.display_cursor('az7r9s3wpqy7n',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>* 1</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)
```
V$SQL_BIND_CAPTURE
- \texttt{STATISTICS\_LEVEL = TYPICAL} or ALL
- Collected at 15 minute intervals

\begin{verbatim}
SELECT name, position, datatype_string, value_string
FROM v$sql_bind_capture
WHERE sql_id = '15uughacxfh13';
\end{verbatim}

<table>
<thead>
<tr>
<th>NAME</th>
<th>POSITION</th>
<th>DATATYPE_STRING</th>
<th>VALUE_STRING</th>
</tr>
</thead>
<tbody>
<tr>
<td>:B1</td>
<td>1</td>
<td>BINARY_DOUBLE</td>
<td></td>
</tr>
</tbody>
</table>

Bind Values also provided by tracing
- Level 4 - bind values
- Level 8 - wait information
- Level 12 - bind values and wait information
Provides data on objects in execution plans.
- Table sizes
- Existing indexes
- Cardinality of columns
- Segment sizes
- Histograms and Data Skew
- Many things the CBO uses

Use TuningStats.sql
- [http://support.confio.com/kb/1534](http://support.confio.com/kb/1534)

Run it for expensive data access targets
Who registered yesterday for SQL Tuning

SELECT s.fname, s.lname, r.signup_date
FROM   student s, active_registrations r, class c
WHERE  s.student_id = r.student_id
AND    r.class_id = c.class_id
AND    UPPER(c.name)  = 'SQL TUNING'
AND    c.class_level = 101
AND    r.signup_date BETWEEN
        TRUNC(SYSDATE) AND TRUNC(SYSDATE-1)
CLASS
  class_id
  name
  class_level

REGISTRATION
  class_id
  student_id
  signup_date
  cancelled

STUDENT
  student_id
  fname
  lname
Gather - Summary

- Execution Plan
  - V$SQL_PLAN
  - Do not use EXPLAIN PLAN
  - DBMS_XPLAN
- Bind Values
  - V$SQL_BIND_CAPTURE
  - Tracing
- Table and Index Statistics
- ERD
Find the Expensive Steps
Review Predicates for these Steps
Evaluate Object Stats
  • Table Definitions
  • Sizes and Row Counts
Determine Existing Indexes
  • Index Definitions
  • Index Selectivity
Evaluate Column Stats
  • Limiting Factors from WHERE Clause
Ensure Join Columns are Indexed
## Execution Plan

<table>
<thead>
<tr>
<th>Id</th>
<th>Operation</th>
<th>Name</th>
<th>Rows</th>
<th>Bytes</th>
<th>Cost</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>SELECT STATEMENT</td>
<td></td>
<td></td>
<td></td>
<td>79</td>
</tr>
<tr>
<td>1</td>
<td>NESTED LOOPS</td>
<td></td>
<td>1</td>
<td>167</td>
<td>79</td>
</tr>
<tr>
<td>2</td>
<td>NESTED LOOPS</td>
<td></td>
<td>1</td>
<td>81</td>
<td>78</td>
</tr>
<tr>
<td>3</td>
<td>NESTED LOOPS</td>
<td></td>
<td>1</td>
<td>51</td>
<td>77</td>
</tr>
<tr>
<td>4</td>
<td>VIEW</td>
<td>VW_SQ_1</td>
<td>1</td>
<td>35</td>
<td>77</td>
</tr>
<tr>
<td>5</td>
<td>FILTER</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>6</td>
<td>HASH GROUP BY</td>
<td></td>
<td>1</td>
<td>17</td>
<td>77</td>
</tr>
<tr>
<td>7</td>
<td>FILTER</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>8</td>
<td>TABLE ACCESS FULL</td>
<td>REGISTRATION</td>
<td>1</td>
<td>17</td>
<td>76</td>
</tr>
<tr>
<td>9</td>
<td>INDEX UNIQUE SCAN</td>
<td>SYS_C0020876</td>
<td>1</td>
<td>16</td>
<td>0</td>
</tr>
<tr>
<td>10</td>
<td>TABLE ACCESS BY INDEX ROWID</td>
<td>STUDENT</td>
<td>1</td>
<td>30</td>
<td>1</td>
</tr>
<tr>
<td>11</td>
<td>INDEX UNIQUE SCAN</td>
<td>SYS_C0020874</td>
<td>1</td>
<td></td>
<td>0</td>
</tr>
<tr>
<td>12</td>
<td>TABLE ACCESS BY INDEX ROWID</td>
<td>CLASS</td>
<td>1</td>
<td>86</td>
<td>1</td>
</tr>
<tr>
<td>13</td>
<td>INDEX UNIQUE SCAN</td>
<td>SYS_C0020875</td>
<td>1</td>
<td></td>
<td>0</td>
</tr>
</tbody>
</table>

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

5 - filter((MAX("SIGNUP_DATE")>=SYSDATE@! AND MAX("SIGNUP_DATE")<=TRUNC(SYSDATE@!-1)))
7 - filter(SYSDATE@!<=TRUNC(SYSDATE@!-1))
8 - filter("CANCELLED"='N')
9 - access("R1"."STUDENT_ID"="STUDENT_ID" AND "R1"."CLASS_ID"="CLASS_ID" AND "SIGNUP_DATE"="VW_COL_1")
   filter(('"SIGNUP_DATE"'>=SYSDATE@! AND "SIGNUP_DATE"<=TRUNC(SYSDATE@!-1)))
11 - access("S"."STUDENT_ID"="STUDENT_ID")
12 - filter("C"."CLASS_LEVEL"=101 AND UPPER("C"."NAME")='SQL TUNING')
13 - access("CLASS_ID"="C"."CLASS_ID")
## Expensive Steps

<table>
<thead>
<tr>
<th>Id</th>
<th>Operation</th>
<th>Name</th>
<th>Rows</th>
<th>Bytes</th>
<th>Cost</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>SELECT STATEMENT</td>
<td></td>
<td></td>
<td></td>
<td>79</td>
</tr>
<tr>
<td>1</td>
<td>NESTED LOOPS</td>
<td></td>
<td>1</td>
<td>167</td>
<td>79</td>
</tr>
<tr>
<td>2</td>
<td>NESTED LOOPS</td>
<td></td>
<td>1</td>
<td>81</td>
<td>78</td>
</tr>
<tr>
<td>3</td>
<td>NESTED LOOPS</td>
<td></td>
<td>1</td>
<td>51</td>
<td>77</td>
</tr>
<tr>
<td>4</td>
<td>VIEW</td>
<td>VW_SQ_1</td>
<td>1</td>
<td>35</td>
<td>77</td>
</tr>
<tr>
<td>* 5</td>
<td>FILTER</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>* 6</td>
<td>HASH GROUP BY</td>
<td></td>
<td>1</td>
<td>17</td>
<td>77</td>
</tr>
<tr>
<td>* 7</td>
<td>FILTER</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>* 8</td>
<td>TABLE ACCESS FULL</td>
<td>REGISTRATION</td>
<td>1</td>
<td>17</td>
<td>76</td>
</tr>
<tr>
<td>* 9</td>
<td>INDEX UNIQUE SCAN</td>
<td>SYS_C0020876</td>
<td>1</td>
<td>16</td>
<td>0</td>
</tr>
<tr>
<td>10</td>
<td>TABLE ACCESS BY INDEX ROWID</td>
<td>STUDENT</td>
<td>1</td>
<td>30</td>
<td>1</td>
</tr>
<tr>
<td>* 11</td>
<td>INDEX UNIQUE SCAN</td>
<td>SYS_C0020874</td>
<td>1</td>
<td></td>
<td>0</td>
</tr>
<tr>
<td>* 12</td>
<td>TABLE ACCESS BY INDEX ROWID</td>
<td>CLASS</td>
<td>1</td>
<td>86</td>
<td>1</td>
</tr>
<tr>
<td>* 13</td>
<td>INDEX UNIQUE SCAN</td>
<td>SYS_C0020875</td>
<td>1</td>
<td></td>
<td>0</td>
</tr>
</tbody>
</table>

Predicate Information (identified by operation id):

5 - filter((MAX("SIGNUP_DATE")>=SYSDATE@! AND MAX("SIGNUP_DATE")<=TRUNC(SYSDATE@!-1)))
7 - filter(SYSDATE@!<=TRUNC(SYSDATE@!-1))
8 - filter("CANCELED"='N')
9 - access("R1"."STUDENT_ID"="STUDENT_ID" AND "R1"."CLASS_ID"="CLASS_ID" AND "SIGNUP_DATE"="VW_COL_1")
   filter(("SIGNUP_DATE">=SYSDATE@! AND "SIGNUP_DATE"<=TRUNC(SYSDATE@!-1)))
11 - access("S"."STUDENT_ID"="STUDENT_ID"
12 - filter("C"."CLASS_LEVEL"=101 AND UPPER("C"."NAME")='SQL TUNING')
13 - access("CLASS_ID"="C"."CLASS_ID")
Who registered for SQL Tuning within last day

SELECT s.fname, s.lname, r.signup_date
FROM student s, active_registrations r, class c
WHERE s.student_id = r.student_id
AND r.class_id = c.class_id
AND UPPER(c.name) = 'SQL TUNING'
AND c.class_level = 101
AND r.signup_date BETWEEN
    TRUNC(SYSDATE) AND TRUNC(SYSDATE-1)
set long 8000
select text from user_views where
view_name='ACTIVE_REGISTRATIONS';

TEXT
----------------------------------------
SELECT student_id, class_id, signup_date
FROM registration r1
WHERE signup_date = (  
    SELECT MAX(signup_date)
    FROM registration r2
    WHERE r1.class_id = r2.class_id
    AND r1.student_id = r2.student_id
    AND r2.cancelled = 'N'
)
# REGISTRATION Table Data

<table>
<thead>
<tr>
<th>Name</th>
<th>Null?</th>
<th>Type</th>
</tr>
</thead>
<tbody>
<tr>
<td>STUDENT_ID</td>
<td>NOT NULL</td>
<td>NUMBER</td>
</tr>
<tr>
<td>CLASS_ID</td>
<td>NOT NULL</td>
<td>NUMBER</td>
</tr>
<tr>
<td>SIGNUP_DATE</td>
<td>NOT NULL</td>
<td>DATE</td>
</tr>
<tr>
<td>CANCELLED</td>
<td></td>
<td>CHAR(1)</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>INDEX_NAME</th>
<th>UNIQUENESS</th>
<th>COLUMN_NAME</th>
<th>COLUMN_POSITION</th>
</tr>
</thead>
<tbody>
<tr>
<td>SYS_C0020876</td>
<td>UNIQUE</td>
<td>STUDENT_ID</td>
<td>1</td>
</tr>
<tr>
<td>SYS_C0020876</td>
<td>UNIQUE</td>
<td>CLASS_ID</td>
<td>2</td>
</tr>
<tr>
<td>SYS_C0020876</td>
<td>UNIQUE</td>
<td>SIGNUP_DATE</td>
<td>3</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>COLUMN_NAME</th>
<th>NUM_DISTINCT</th>
<th>NUM_NULLS</th>
<th>DENSITY</th>
<th>SAMPLE_SIZE</th>
</tr>
</thead>
<tbody>
<tr>
<td>CANCELLED</td>
<td>2</td>
<td>0</td>
<td>.5</td>
<td>5443</td>
</tr>
<tr>
<td>CLASS_ID</td>
<td>998</td>
<td>0</td>
<td>.001002004</td>
<td>5443</td>
</tr>
<tr>
<td>SIGNUP_DATE</td>
<td>32817</td>
<td>0</td>
<td>.000030472</td>
<td>79983</td>
</tr>
<tr>
<td>STUDENT_ID</td>
<td>9999</td>
<td>0</td>
<td>.00010001</td>
<td>79983</td>
</tr>
</tbody>
</table>
select cancelled, count(1)
from registration group by cancelled;

<table>
<thead>
<tr>
<th></th>
<th>COUNT(1)</th>
</tr>
</thead>
<tbody>
<tr>
<td>Y</td>
<td>638</td>
</tr>
<tr>
<td>N</td>
<td>79345</td>
</tr>
</tbody>
</table>

select trunc(signup_date), count(1)
from registration group by trunc(signup_date)

<table>
<thead>
<tr>
<th>TRUNC(SIGNUP_DATE)</th>
<th>COUNT(1)</th>
</tr>
</thead>
<tbody>
<tr>
<td>01/01/09 00:00</td>
<td>100</td>
</tr>
<tr>
<td>01/02/09 00:00</td>
<td>290</td>
</tr>
<tr>
<td>01/03/09 00:00</td>
<td>107</td>
</tr>
<tr>
<td>01/04/09 00:00</td>
<td>845</td>
</tr>
<tr>
<td>01/05/09 00:00</td>
<td>3190</td>
</tr>
<tr>
<td>01/06/09 00:00</td>
<td>2727</td>
</tr>
<tr>
<td>...</td>
<td></td>
</tr>
<tr>
<td>01/29/09 00:00</td>
<td>2693</td>
</tr>
</tbody>
</table>

45 Buckets with fairly even distribution.
```
create index reg_sudt on registration(signup_date)

<table>
<thead>
<tr>
<th>Id</th>
<th>Operation</th>
<th>Name</th>
<th>Rows</th>
<th>Bytes</th>
<th>Cost</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>SELECT STATEMENT</td>
<td></td>
<td></td>
<td></td>
<td>10</td>
</tr>
<tr>
<td>* 1</td>
<td>FILTER</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>2</td>
<td>HASH GROUP BY</td>
<td></td>
<td>1</td>
<td>174</td>
<td>10</td>
</tr>
<tr>
<td>* 3</td>
<td>FILTER</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>* 4</td>
<td>TABLE ACCESS BY INDEX ROWID</td>
<td>REGISTRATION</td>
<td>1</td>
<td>17</td>
<td>3</td>
</tr>
<tr>
<td>5</td>
<td>NESTED LOOPS</td>
<td></td>
<td>1</td>
<td>174</td>
<td>9</td>
</tr>
<tr>
<td>6</td>
<td>NESTED LOOPS</td>
<td></td>
<td>1</td>
<td>157</td>
<td>6</td>
</tr>
<tr>
<td>7</td>
<td>NESTED LOOPS</td>
<td></td>
<td>1</td>
<td>59</td>
<td>5</td>
</tr>
<tr>
<td>8</td>
<td>TABLE ACCESS BY INDEX ROWID</td>
<td>REGISTRATION</td>
<td>1</td>
<td>17</td>
<td>4</td>
</tr>
<tr>
<td>* 9</td>
<td>INDEX RANGE SCAN</td>
<td>REG_SUDT</td>
<td>2</td>
<td></td>
<td>2</td>
</tr>
<tr>
<td>10</td>
<td>TABLE ACCESS BY INDEX ROWID</td>
<td>STUDENT</td>
<td>1</td>
<td>42</td>
<td>1</td>
</tr>
<tr>
<td>* 11</td>
<td>INDEX UNIQUE SCAN</td>
<td>SYS_C0020874</td>
<td>1</td>
<td></td>
<td>0</td>
</tr>
<tr>
<td>* 12</td>
<td>TABLE ACCESS BY INDEX ROWID</td>
<td>CLASS</td>
<td>1</td>
<td>98</td>
<td>1</td>
</tr>
<tr>
<td>* 13</td>
<td>INDEX UNIQUE SCAN</td>
<td>SYS_C0020875</td>
<td>1</td>
<td></td>
<td>0</td>
</tr>
<tr>
<td>* 14</td>
<td>INDEX RANGE SCAN</td>
<td>SYS_C0020876</td>
<td>1</td>
<td></td>
<td>2</td>
</tr>
</tbody>
</table>
```
**AND UPPER(c.name) = ‘SQL TUNING’**

- Should only return one row (or just a few) from CLASS and join to REGISTRATION table
- Created a function-based index on UPPER(name)
- Added another index on registration.class_id

<table>
<thead>
<tr>
<th>Id</th>
<th>Operation</th>
<th>Name</th>
<th>Rows</th>
<th>Bytes</th>
<th>Cost</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>SELECT STATEMENT</td>
<td></td>
<td></td>
<td></td>
<td>7</td>
</tr>
<tr>
<td>1</td>
<td>FILTER</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>2</td>
<td>NESTED LOOPS</td>
<td></td>
<td>1</td>
<td>132</td>
<td>4</td>
</tr>
<tr>
<td>3</td>
<td>NESTED LOOPS</td>
<td></td>
<td>1</td>
<td>102</td>
<td>3</td>
</tr>
<tr>
<td>4</td>
<td>TABLE ACCESS BY INDEX ROWID</td>
<td>CLASS</td>
<td>1</td>
<td>86</td>
<td>2</td>
</tr>
<tr>
<td>5</td>
<td>INDEX RANGE SCAN</td>
<td>CL_FUNC</td>
<td>1</td>
<td></td>
<td>1</td>
</tr>
<tr>
<td>6</td>
<td>INDEX RANGE SCAN</td>
<td>REG_ALT</td>
<td>1</td>
<td>16</td>
<td>1</td>
</tr>
<tr>
<td>7</td>
<td>SORT AGGREGATE</td>
<td></td>
<td>1</td>
<td>18</td>
<td></td>
</tr>
<tr>
<td>8</td>
<td>TABLE ACCESS BY INDEX ROWID</td>
<td>REGISTRATION</td>
<td>1</td>
<td>18</td>
<td>3</td>
</tr>
<tr>
<td>9</td>
<td>INDEX RANGE SCAN</td>
<td>REG_ALT</td>
<td>1</td>
<td></td>
<td>2</td>
</tr>
<tr>
<td>10</td>
<td>TABLE ACCESS BY INDEX ROWID</td>
<td>STUDENT</td>
<td>1</td>
<td>30</td>
<td>1</td>
</tr>
<tr>
<td>11</td>
<td>INDEX UNIQUE SCAN</td>
<td>SYS_C0020874</td>
<td>1</td>
<td></td>
<td>0</td>
</tr>
</tbody>
</table>
Who cancelled classes within the week

```sql
SELECT s.lname, c.name, r.signup_date cancel_date
FROM registration r, student s, class c
WHERE r.signup_date between sysdate and sysdate-7
AND r.cancelled = 'Y'
AND r.student_id = s.student_id
AND r.class_id = c.class_id
```

- 30% of rows are dated within last week
- No index on CANCELLED column = FTS
- Will an index on CANCELLED column help?
  - Why or why not?
select cancelled, count(1)
from registration group by cancelled;

<table>
<thead>
<tr>
<th>C</th>
<th>COUNT(1)</th>
</tr>
</thead>
<tbody>
<tr>
<td>Y</td>
<td>638</td>
</tr>
<tr>
<td>N</td>
<td>79345</td>
</tr>
</tbody>
</table>

- **Oracle will not use an index on this column**
  - Unless it has more information
  - CBO assumes an even data distribution

- **Histograms give more information to Oracle**
  - Based on skewed data, CBO realizes an index would be beneficial
  - Works best with literal values
  - Bind Variables – Oracle peeks first time only
dbms_stats.gather_table_stats(
    ownname => 'STDMGMT',
    tabname => 'REGISTRATION',
    method_opt=>'FOR COLUMNS cancelled SIZE AUTO')

<table>
<thead>
<tr>
<th>Id</th>
<th>Operation</th>
<th>Name</th>
<th>Rows</th>
<th>Bytes</th>
<th>Cost</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>SELECT STATEMENT</td>
<td></td>
<td></td>
<td></td>
<td>7</td>
</tr>
<tr>
<td>*  1</td>
<td>FILTER</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>*  2</td>
<td>TABLE ACCESS BY INDEX ROWID</td>
<td>REGISTRATION</td>
<td>1</td>
<td>17</td>
<td>7</td>
</tr>
<tr>
<td>*  3</td>
<td>INDEX RANGE SCAN</td>
<td>REG_CAN</td>
<td>754</td>
<td></td>
<td>2</td>
</tr>
</tbody>
</table>
Monitor

- Monitor the improvement
  - Be able to prove that tuning made a difference
  - Take new metrics measurements
  - Compare them to initial readings
  - Brag about the improvements – no one else will

- Monitor for next tuning opportunity
  - Tuning is iterative
  - There are always room for improvements
  - Make sure you tune things that make a difference

- Shameless Product Pitch - Ignite
Summary

- **Identify**
  - What is the Bottleneck
  - End-to-End view of performance
  - Simplify

- **Gather**
  - Metrics – Current Performance
  - Wait Time
  - Execution Plan
  - Object Definitions and Statistics

- **Tune**

- **Monitor**
  - New Metrics, Wait Time Profile, Execution Plan
- Use outer joins vs. NOT IN / NOT EXISTS
- Which class is currently empty?

```sql
select class_id, name
from class
where class_id not in (
    select class_id from registration)
```

<table>
<thead>
<tr>
<th>Id</th>
<th>Operation</th>
<th>Name</th>
<th>Rows</th>
<th>Bytes</th>
<th>Cost</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>SELECT STATEMENT</td>
<td></td>
<td></td>
<td></td>
<td>88</td>
</tr>
<tr>
<td>* 1</td>
<td>HASH JOIN ANTI</td>
<td></td>
<td>1</td>
<td>68</td>
<td>88</td>
</tr>
<tr>
<td>2</td>
<td>TABLE ACCESS FULL</td>
<td>CLASS</td>
<td>1000</td>
<td>64000</td>
<td>14</td>
</tr>
<tr>
<td>3</td>
<td>TABLE ACCESS FULL</td>
<td>REGISTRATION</td>
<td>80056</td>
<td>312K</td>
<td>72</td>
</tr>
</tbody>
</table>
Try NOT EXISTS vs. NOT IN

```sql
select class_id, name
from class c
where not exists ( select 1 from registration r
                      where c.class_id = r.class_id)
```

<table>
<thead>
<tr>
<th>Id</th>
<th>Operation</th>
<th>Name</th>
<th>Rows</th>
<th>Bytes</th>
<th>Cost</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>SELECT STATEMENT</td>
<td></td>
<td></td>
<td></td>
<td>88</td>
</tr>
<tr>
<td>* 1</td>
<td>HASH JOIN ANTI</td>
<td></td>
<td>1</td>
<td>68</td>
<td>88</td>
</tr>
<tr>
<td>2</td>
<td>TABLE ACCESS FULL</td>
<td>CLASS</td>
<td>1000</td>
<td>64000</td>
<td>14</td>
</tr>
<tr>
<td>3</td>
<td>TABLE ACCESS FULL</td>
<td>REGISTRATION</td>
<td>80056</td>
<td>312K</td>
<td>72</td>
</tr>
</tbody>
</table>
Try OUTER JOIN

No Differences with 3 options

```sql
select c.class_id, c.name
from class c, registration r
where c.class_id = r.class_id (+)
and r.class_id is null
```

---

<table>
<thead>
<tr>
<th>Id</th>
<th>Operation</th>
<th>Name</th>
<th>Rows</th>
<th>Bytes</th>
<th>Cost</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>SELECT STATEMENT</td>
<td></td>
<td></td>
<td></td>
<td>88</td>
</tr>
<tr>
<td>* 1</td>
<td>HASH JOIN ANTI</td>
<td></td>
<td>1</td>
<td>68</td>
<td>88</td>
</tr>
<tr>
<td>2</td>
<td>TABLE ACCESS FULL</td>
<td>CLASS</td>
<td>1000</td>
<td>64000</td>
<td>14</td>
</tr>
<tr>
<td>3</td>
<td>TABLE ACCESS FULL</td>
<td>REGISTRATION</td>
<td>80056</td>
<td>312K</td>
<td>72</td>
</tr>
</tbody>
</table>
**Myth 2 – Option 1**

- **Use MINUS vs. NOT IN**
- **Which students live in DC area but not in 20002 or 20003 zip**
- **Cost = 15, LIO = 20**

```sql
select student_id from student
where state in ('VA', 'DC', 'MD')
and zip not in (20002, 20003)
```

<table>
<thead>
<tr>
<th>Id</th>
<th>Operation</th>
<th>Name</th>
<th>Rows</th>
<th>Bytes</th>
<th>Cost</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>SELECT STATEMENT</td>
<td></td>
<td></td>
<td></td>
<td>15</td>
</tr>
<tr>
<td>1</td>
<td>INLIST ITERATOR</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>*  2</td>
<td>TABLE ACCESS BY INDEX ROWID</td>
<td>STUDENT</td>
<td>11</td>
<td>110</td>
<td>15</td>
</tr>
<tr>
<td>*  3</td>
<td>INDEX RANGE SCAN</td>
<td>ST_ST</td>
<td>11</td>
<td></td>
<td>3</td>
</tr>
</tbody>
</table>
Try MINUS vs. NOT IN
Cost = 20, LIO = 23 – Worse Performance

select student_id from student
where state in ('VA', 'DC', 'MD')
minus
select student_id from student
where zip in (20002, 20003)

<table>
<thead>
<tr>
<th>Id</th>
<th>Operation</th>
<th>Name</th>
<th>Rows</th>
<th>Bytes</th>
<th>Cost</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>SELECT STATEMENT</td>
<td></td>
<td></td>
<td></td>
<td>20</td>
</tr>
<tr>
<td>1</td>
<td>MINUS</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>2</td>
<td>SORT UNIQUE</td>
<td></td>
<td>11</td>
<td>77</td>
<td>16</td>
</tr>
<tr>
<td>3</td>
<td>INLIST ITERATOR</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>4</td>
<td>TABLE ACCESS BY INDEX ROWID</td>
<td>STUDENT</td>
<td>11</td>
<td>77</td>
<td>15</td>
</tr>
<tr>
<td>* 5</td>
<td>INDEX RANGE SCAN</td>
<td>ST_ST</td>
<td>11</td>
<td></td>
<td>3</td>
</tr>
<tr>
<td>6</td>
<td>SORT UNIQUE</td>
<td></td>
<td>2</td>
<td>14</td>
<td>4</td>
</tr>
<tr>
<td>7</td>
<td>INLIST ITERATOR</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>8</td>
<td>TABLE ACCESS BY INDEX ROWID</td>
<td>STUDENT</td>
<td>2</td>
<td>14</td>
<td>3</td>
</tr>
<tr>
<td>* 9</td>
<td>INDEX RANGE SCAN</td>
<td>ST_ZIP</td>
<td>2</td>
<td></td>
<td>2</td>
</tr>
</tbody>
</table>
Confio Software - Monitor

- Developer of Wait-Based Performance Tools
- Igniter Suite
  - Ignite for SQL Server, Oracle, DB2, Sybase
- Provides Help With
  - Identify
  - Gather
  - Monitor
- Based in Colorado, worldwide customers
- Free trial at www.confio.com