Tuna Helper
Proven Process for SQL Tuning

Dean Richards
Senior DBA, Confio Software
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
Introduction

- 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

- 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
Tracing a Session / Process
User / Batch Job Complaints
Highest I/O (LIO, PIO)
SQL Performing Full Table Scans
Known Poorly Performing SQL
Highest Wait Times (Ignite, AWR, etc)
Identify – End-to-End

- Business Aspects
  - Who registered yesterday for SQL Tuning
  - Who uses this information?
  - Why does the business need to know this?
  - How often is the information needed?

- Technical Information
  - Review Tables, Indexes, Triggers, Views, etc
  - Understand Relationships
  - Know the Data (High Level)

- End-to-End Process
  - Understand Application Architecture
  - What Portion of the Total Time is Database
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
- ROW_WAIT_OBJ#

**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
Wait Event Information

**V$SESSION**
- SID
- USERNAME
- SQL_ID
- PROGRAM
- MODULE
- ACTION
- PLAN_HASH_VALUE
- ROW_WAIT_OBJ#

**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
SELECT s.sql_id, sql.sql_text, sql.plan_hash_value,
    DECODE(s.state, 'WAITING', s.event, 'CPU') waitevent,
    s.p1, s.p2, s.p3
FROM v$session s
JOIN v$sql sql ON (   
    s.sql_id = sql.sql_id AND s.sql_address = sql.address
)
AND sql.sql_text LIKE 'SELECT%' -- substitute your own
AND s.sid = 20        -- if you know it
AND <whatever else you know>
Which scenario is worse?

SQL Statement 1
- Executed 100 times
- Caused 100 minutes of wait time for end user
- Waited 99% of time on “db file sequential read”

SQL Statement 2
- Executed 1 time
- Caused 100 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

Use Execution Plan (later)

- Helps isolate the portion of the query that is performing poorly
Identify – Summary

- Determine the SQL
- Understand End-to-End
- Measure Wait Time
- Simplify Statement
  - Based on Execution Plan
Gather - Metrics

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

- Collect Wait Time Metrics – How Long
  - Locking / Blocking
  - I/O problem, Latch contention
  - May be multiple issues
  - All have different resolutions

- Document everything in simple language
- 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)
  - Works when you know a problem will occur
    ALTER SESSION SET tracefile_identifier = dean;
    ALTER SESSION SET sql_trace = true;
- 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('az7r9s3wpgq7n',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</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)
```
Gather – Bind Values

- **V$SQL_BIND_CAPTURE**
  - STATISTICS_LEVEL = TYPICAL or ALL
  - Collected at 15 minute intervals

```sql
SELECT name, position, datatype_string, value_string
FROM v$sql_bind_capture
WHERE sql_id = '15uughacxfh13';
```

<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
Use TuningStats.sql
  - http://support.confio.com/kb/1534

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

Run it for any table involved in query
Who registered yesterday for SQL Tuning

```sql
SELECT s.fname, s.lname, r.signup_date
FROM student s, registration r, class c
WHERE s.student_id = r.student_id
AND r.class_id = c.class_id
AND UPPER(c.name) = 'SQL TUNING'
AND r.signup_date BETWEEN TRUNC(SYSDATE - 1) AND TRUNC(SYSDATE)
AND r.cancelled = 'N'
```

Execution Time – 12:38
Wait Time – 95% on “db file scattered read”
CLASS
  class_id
  name
  class_level

REGISTRATION
  class_id
  student_id
  signup_date
  cancelled

STUDENT
  student_id
  fname
  lname
## 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>95</td>
</tr>
<tr>
<td>1</td>
<td>NESTED LOOPS</td>
<td></td>
<td>1</td>
<td>167</td>
<td>95</td>
</tr>
<tr>
<td>2</td>
<td>NESTED LOOPS</td>
<td></td>
<td>1</td>
<td>138</td>
<td>94</td>
</tr>
<tr>
<td>3</td>
<td>NESTED LOOPS</td>
<td></td>
<td>7</td>
<td>357</td>
<td>87</td>
</tr>
<tr>
<td>4</td>
<td>VIEW</td>
<td>VW_SQ_1</td>
<td>201</td>
<td>7035</td>
<td>87</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>201</td>
<td>3417</td>
<td>87</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>80000</td>
<td>1328K</td>
<td>76</td>
</tr>
<tr>
<td>* 9</td>
<td>INDEX UNIQUE SCAN</td>
<td>SYS_C0036920</td>
<td>1</td>
<td>16</td>
<td>0</td>
</tr>
<tr>
<td>* 10</td>
<td>TABLE ACCESS BY INDEX ROWID</td>
<td>CLASS</td>
<td>1</td>
<td>87</td>
<td>1</td>
</tr>
<tr>
<td>* 11</td>
<td>INDEX UNIQUE SCAN</td>
<td>SYS_C0036919</td>
<td>1</td>
<td></td>
<td>0</td>
</tr>
<tr>
<td>12</td>
<td>TABLE ACCESS BY INDEX ROWID</td>
<td>STUDENT</td>
<td>1</td>
<td>29</td>
<td>1</td>
</tr>
<tr>
<td>* 13</td>
<td>INDEX UNIQUE SCAN</td>
<td>SYS_C0036918</td>
<td>1</td>
<td></td>
<td>0</td>
</tr>
</tbody>
</table>

Predicate Information (identified by operation id):

5 - filter((MAX("SIGNUP_DATE")>=TRUNC(SYSDATE@!-1) AND MAX("SIGNUP_DATE")<=TRUNC(SYSDATE@!)))
7 - filter(TRUNC(SYSDATE@!-1)<=TRUNC(SYSDATE@!))
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">=TRUNC(SYSDATE@!-1) AND "SIGNUP_DATE"<=TRUNC(SYSDATE@!)))
10 - filter(UPPER("C"."NAME")='SQL TUNING')
11 - access("CLASS_ID"="C"."CLASS_ID")
13 - access("S"."STUDENT_ID"="STUDENT_ID")
Execution Plan
- V$SQL_PLAN
- Do not use EXPLAIN PLAN
- DBMS_XPLAN

Bind Values
- V$SQL_BIND_CAPTURE
- Tracing

Table and Index Statistics

ERD
**SQL Tuning – Dan Tow**
- Great book that teaches SQL Diagramming
- [http://www.singingsql.com](http://www.singingsql.com)

```
select count(1) from registration where cancelled = 'N'
and signup_date between trunc(sysdate-1) and trunc(sysdate)
```

\[
\frac{3562}{80000} = 0.0445
\]

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

\[
\frac{2}{1000} = 0.002
\]
create index cl_uname on class (upper(name));

- Index on registration was (student_id, class_id)

create index reg_alt 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>10</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>7</td>
</tr>
<tr>
<td>3</td>
<td>NESTED LOOPS</td>
<td></td>
<td>1</td>
<td>103</td>
<td>6</td>
</tr>
<tr>
<td>4</td>
<td>TABLE ACCESS BY INDEX ROWID</td>
<td>CLASS</td>
<td>1</td>
<td>87</td>
<td>5</td>
</tr>
<tr>
<td>5</td>
<td>INDEX RANGE SCAN</td>
<td>CL_UNAME</td>
<td>4</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>17</td>
<td></td>
</tr>
<tr>
<td>8</td>
<td>TABLE ACCESS BY INDEX ROWID</td>
<td>REGISTRATION</td>
<td>1</td>
<td>17</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>29</td>
<td>1</td>
</tr>
<tr>
<td>11</td>
<td>INDEX UNIQUE SCAN</td>
<td>SYS_C0036918</td>
<td>1</td>
<td></td>
<td>0</td>
</tr>
</tbody>
</table>
Who cancelled classes within the week

```
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 count(1) from registration where cancelled = 'Y'
and signup_date between trunc(sysdate-1) and trunc(sysdate)

622 / 80000 = .0077

select count(1) from registration where cancelled = 'Y'

638 / 80000 = .0079

select count(1) from registration
where signup_date between trunc(sysdate-1) and trunc(sysdate)

11598 / 80000 = .1449
create index reg_can on registration(cancelled);

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
```sql
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
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](http://www.confio.com)