SQL Plan Management: New Approach for Providing SQL Performance Stability in Oracle Database 11g

Maria Colgan & Mohamed Zait
The following is intended to outline our general product direction. It is intended for information purposes only, and may not be incorporated into any contract. It is not a commitment to deliver any material, code, or functionality, and should not be relied upon in making purchasing decisions. The development, release, and timing of any features or functionality described for Oracle’s products remains at the sole discretion of Oracle.
SQL Plan Management

Business Requirement
• Unpredictable changes in execution plans can happen
  • New Statistics
  • Changes in the Environment
  • Software upgrades
• Today you have to ‘freeze’ critical plans or statistics

Solution
• Optimizer automatically manages ‘execution plans’
  • Only known and verified plans are used
• Plan changes are automatically verified
  • Only comparable or better plans are used going forward

SQL Plan Management is controlled plan evolution
SQL Plan Management

Guaranteed plan stability and controlled plan evolution
Without SQL Plan Management

- SQL statement is parsed for the first time and a plan is generated
- Does plan gives good performance? Plan is “verified by execution”

![Diagram showing the verification process]

- Something changes in the environment
  - Statistics are re-gathered, DB upgrade or parameter change
  - Changes result in new plan – does new plan meet perf criteria?
  - New plan implemented regardless of resulting performance

![Diagram showing the implementation of a new plan]
With SQL Plan Management

- SQL statement is parsed for the first time and a plan is generated
- Check the log to see if this is a repeatable SQL statement
- Add SQL statement signature to the log and execute it
- Plan performance is still “verified by execution”
With SQL Plan Management

- SQL statement is parsed again and a plan is generated
- Check log to see if this is a repeatable SQL statement
- Create a Plan history and use current plan as SQL plan baseline
- Plan performance is “verified by execution”
Let's look at an example of SPM in action
With SQL Plan Management

- Something changes in the environment
- SQL statement is parsed again and a **new plan is generated**
- New plan is not the same as the baseline – **new plan is not executed** but marked for verification
With SQL Plan Management

- Something changes in the environment
- SQL statement is parsed again and a **new plan is generated**
- New plan is not the same as the baseline – **new plan is not executed** but marked for verification
- Execute known plan baseline - plan performance is “verify by history”
Verifying the new plan

- Non-baseline plans will not be used until verified
- DBA can verify plan at any time

Optimizer checks if new plan is as good as or better than old plan

Invoke or schedule verification

Plans which perform as good as or better than original plan are added to the plan baseline

Plans which don’t perform as good as the original plan stay in the plan history and are marked unaccepted
SQL Plan Management – the details

• Controlled by two init.ora parameter
  • **optimizer_capture_sql_plan_baselines**
    • Controls auto-capture of SQL plan baselines for repeatable stmts
    • Set to false by default in 11gR1
  • **optimizer_use_sql_plan_baselines**
    • Controls the use of existing SQL plan baselines by the optimizer
    • Set to true by default in 11gR1

• Monitoring SPM
  • Dictionary view DBA_SQL_PLAN_BASELINE
  • Via the SQL Plan Control in EM DBControl

• Managing SPM
  • PL/SQL package DBMS_SPM
  • Via the SQL Plan Control in EM DBControl
SPM Plan Capture – Bulk

- From SQL Tuning Set (STS)
  - Captures plan details for a (critical) set of SQL Statement
  - Load these plans into SPM as baseline plans
  - Next time statements are executed baseline plans will be used

- From Cursor Cache
  - Load plans from the cursor cache into SPM as baseline plans
    - Filters can be specified (SQL_ID, Module name, schema)
    - Next time statements are executed baseline plans will be used

- From staging table
  - SQL plan baselines can be captured on another system
  - Exported via a table (similar to statistics) and imported locally
  - Plan are “unpacked” from the table and loaded into SPM
SQL Plan Management - general upgrade strategy

- Seeding the SQL Plan Baselines with 10g plans No plan change on upgrade
- After all SQL Plan Baselines are populated switch Optimizer_Features_Enable to 11g
  - new 11g plans will only be used after they have been verified
Lets try this upgrade strategy

Go to Upgrade demo
Possible SQL Plan Manageability Scenarios

Database Upgrade using SQL Tuning Sets

Oracle Database 11g

Plan History

Plan
Well tuned plan

Baseline

Oracle Database 10g

No plan regressions

New Application Deployment, no Tuning Pack required

Production Database

Plan History

Plan
Well tuned plan

Baseline

Development Database 11g

Plan History

Plan
Well tuned plan

Baseline

Baseline plans staging table
For More Information

http://search.oracle.com

SQL Plan Management

OR

Accompanying white paper for this session
QUESTIONS & ANSWERS
# Recommended Campground Demos

<table>
<thead>
<tr>
<th>Demo</th>
<th>Location</th>
</tr>
</thead>
<tbody>
<tr>
<td>Oracle Database 11g SQL Plan Management - booth O27</td>
<td>Moscone West Exhibit Hall</td>
</tr>
<tr>
<td>Oracle Real Application Testing: Database Replay</td>
<td>Moscone West Exhibit Hall</td>
</tr>
<tr>
<td>Oracle Real Application Testing: SQL Performance Analyzer</td>
<td>Moscone West Exhibit Hall</td>
</tr>
<tr>
<td>Self-Managing Database: Automatic Performance Diagnostics</td>
<td>Moscone West Exhibit Hall</td>
</tr>
<tr>
<td>Self-Managing Database: Automatic Application &amp; SQL Tuning</td>
<td>Moscone West Exhibit Hall</td>
</tr>
<tr>
<td>Self-Managing Database: Automatic Fault Diagnostics</td>
<td>Moscone West Exhibit Hall</td>
</tr>
<tr>
<td>Change Management &amp; Data Masking for DBAs</td>
<td>Moscone West Exhibit Hall</td>
</tr>
</tbody>
</table>
## Recommended Sessions (Thursday sessions)

<table>
<thead>
<tr>
<th>Session Title</th>
<th>Date</th>
<th>Time</th>
<th>Location</th>
</tr>
</thead>
<tbody>
<tr>
<td>DBA’s New Best Friend: Advanced SQL Tuning Features of Oracle Database 11g</td>
<td>Thursday, Nov. 15</td>
<td>10:00 a.m.</td>
<td>Moscone West: 2001 - L2</td>
</tr>
<tr>
<td>SQL Plan Management: New Approach for Providing SQL Performance Stability in Oracle Database 11g</td>
<td>Thursday, Nov. 15</td>
<td>10:00 a.m.</td>
<td>Moscone South: 309</td>
</tr>
<tr>
<td>Fault-Diagnostic Best Practices: What Every DBA Must Know About Oracle Database 11g</td>
<td>Thursday, Nov. 15</td>
<td>11:30 a.m.</td>
<td>Moscone South: 309</td>
</tr>
</tbody>
</table>
Using SQL Plan Management in EM
SQL Plan Management in EM

From the Database home page click on server
SQL Plan Management in EM

From the server page click on SQL Plan Control.
SQL Plan Management in EM

Select the SQL Plan Baseline tab
SQL Plan Management in EM

A SQL Plan Baseline is an execution plan deemed to have acceptable performance for a given SQL statement.

---

<table>
<thead>
<tr>
<th>Select Name</th>
<th>SQL Text</th>
<th>Enabled</th>
<th>Accepted</th>
<th>Fixed</th>
<th>Auto Purge</th>
<th>Created</th>
<th>Last Modified</th>
</tr>
</thead>
<tbody>
<tr>
<td>SYS_SQL_PLAN.353ae9acef51249</td>
<td>select subbidcmp(sqla,df,df,df)</td>
<td>YES</td>
<td>YES</td>
<td>NO</td>
<td>YES</td>
<td>Apr 27, 2007 9:53:01 AM</td>
<td>Apr 27, 2007 9:53:01 AM</td>
</tr>
<tr>
<td>SYS_SQL_PLAN.5a5d7b53d52aa65</td>
<td>SELECT HOUR, INTRADAY, EXTRADAY FROM BILL_TIME</td>
<td>YES</td>
<td>YES</td>
<td>NO</td>
<td>YES</td>
<td>Apr 27, 2007 9:53:01 AM</td>
<td>Apr 27, 2007 9:53:01 AM</td>
</tr>
<tr>
<td>SYS_SQL_PLAN.143f81e29a6c3ba</td>
<td>select subbidcmp(sqla,df,df,df)</td>
<td>YES</td>
<td>YES</td>
<td>NO</td>
<td>YES</td>
<td>Apr 27, 2007 9:53:01 AM</td>
<td>Apr 27, 2007 9:53:01 AM</td>
</tr>
<tr>
<td>SYS_SQL_PLAN.5e892d3879061b</td>
<td>select inst.10498f89e7a943c98</td>
<td>YES</td>
<td>YES</td>
<td>NO</td>
<td>YES</td>
<td>Apr 26, 2007 5:17:28 PM</td>
<td>Apr 26, 2007 5:17:28 PM</td>
</tr>
</tbody>
</table>
SQL Plan Management in EM

Settings quickly shows you if SPM is on or not.
SQL Plan Management in EM

The job status area lists all SPM jobs. Clicking on a job_id will take you to the job page.
SQL Plan Management in EM

The list shows all Plan Baselines and their current status.
SQL Plan Management in EM

Clicking on the name of a plan to display the plan in detail.
Display execution plan for SQL plan baseline

SQL Plan Baseline Details

SQL handle: SYS_SQL_6fe28d439dfe352f
SQL text: select /*LOAD_AUTO*/ * from sh.sales where quantity_sold > 40 order by prod_id

Plan name: SYS_SQL_PLAN_8dfe352f554bc8843
Enabled: YES  Fixed: NO  Accepted: YES  Origin: AUTO-CAPTURE

Plan hash value: 3803407550

<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>Petstart</th>
<th>Petstop</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>SELECT STATEMENT</td>
<td></td>
<td>1</td>
<td>29</td>
<td>514 (7)</td>
<td>00:00:07</td>
<td></td>
<td></td>
</tr>
<tr>
<td>1</td>
<td>SORT ORDER BY</td>
<td></td>
<td>1</td>
<td>29</td>
<td>514 (7)</td>
<td>00:00:07</td>
<td></td>
<td></td>
</tr>
<tr>
<td>2</td>
<td>PARTITION RANGE ALL</td>
<td></td>
<td>1</td>
<td>29</td>
<td>513 (7)</td>
<td>00:00:07</td>
<td>1</td>
<td>28</td>
</tr>
<tr>
<td>* 3</td>
<td>TABLE ACCESS FULL</td>
<td>SALES</td>
<td>1</td>
<td>29</td>
<td>513 (7)</td>
<td>00:00:07</td>
<td>1</td>
<td>28</td>
</tr>
</tbody>
</table>
SQL Plan Management in EM

Clicking on the SQL text to display all execute plans for that stmt in detail
SQL Plan Management in EM

The status of any or all Plan Baselines can be changed by using one of these buttons.
SQL Plan Management in EM

New Plan Baselines can be bulk loaded via a STS or from the Cursor Cache.
Loading plans from cursor cache
SQL Plan Management in EM

New Plan Baselines can be imported from a table
Pending Statistics

Controlled statistics publication
Pending Statistics

Business Requirement
• Statistics are published as soon as we complete gathering
  => Possibly unpredictable changes of execution plans
• Today you have ‘freeze’ critical plans or statistics

Solution
• Gather statistics and save as pending
• Verify the new statistics don’t change plans adversely
  • Either on the same or a different system
• Publish verified statistics

Controlled and DBA-verified statistics management
Pending Statistics – in detail

- Controlled by init.ora parameter
  - `optimizer_use_pending_statistics`
    - Determines if the optimizer will use pending statistics
    - Set to false by default in 11gR1

- Use `dbms_stats` package
  - `set_table_prefs`
    - All tables preferences have “publish” set to true by default
  - `publish_private_stats`
    - Once stats have been tested publish them for general use

- Monitor
  - Look at dictionary table `user_*_pending_stats` (* = tab, col, ind)
Extended Optimizer Statistics

Eliminate wrong cardinality estimates
Extended Optimizer Statistics

Business problem - Correlated Columns

- Real data often shows correlations between various attributes
  - e.g. job title influences salary, car model influences make, seasons affect the amount of sold goods (e.g. snow shoes in winter)
- Optimizer has to estimate the correct cardinality
  - “Does an additional filter reduce the result set or not?”

Solution

- Extended Optimizer Statistics provides a mechanism to collect statistics on a group of columns
- Full integration into existing statistics framework
  - Automatically maintained with column statistics
  - Instantaneous and transparent benefit for any migrated application

Improved Cardinality leads to Improved Plans
Extended Statistic Example

single column

SELECT ......FROM..
WHERE model = '530xi'

<table>
<thead>
<tr>
<th>Make</th>
<th>Model</th>
<th>Color</th>
</tr>
</thead>
<tbody>
<tr>
<td>BMW</td>
<td>530xi</td>
<td>RED</td>
</tr>
<tr>
<td>BMW</td>
<td>530xi</td>
<td>BLACK</td>
</tr>
<tr>
<td>BMW</td>
<td>530xi</td>
<td>SILVER</td>
</tr>
<tr>
<td>PORSCHE</td>
<td>911</td>
<td>RED</td>
</tr>
<tr>
<td>MERC</td>
<td>SLK</td>
<td>BLACK</td>
</tr>
<tr>
<td>MERC</td>
<td>C320</td>
<td>SILVER</td>
</tr>
</tbody>
</table>

- Three records selected.
- Single column statistics are accurate

<table>
<thead>
<tr>
<th>Id</th>
<th>Operation</th>
<th>Name</th>
<th>Starts</th>
<th>E-Rows</th>
<th>A-Rows</th>
</tr>
</thead>
<tbody>
<tr>
<td>* 1</td>
<td>TABLE ACCESS FULL</td>
<td>CARS</td>
<td>1</td>
<td>3</td>
<td>3</td>
</tr>
</tbody>
</table>
Example
non-correlated columns

```
SELECT ......FROM..
WHERE model = '530xi'
AND color = 'RED'
```

| BMW | 530xi | RED |

- One record selected.
  - No correlated columns
  - Additional predicate reduces result set
  - Single column statistics are sufficient

<table>
<thead>
<tr>
<th>Id</th>
<th>Operation</th>
<th>Name</th>
<th>Starts</th>
<th>E-Rows</th>
<th>A-Rows</th>
</tr>
</thead>
<tbody>
<tr>
<td>*  1</td>
<td>INDEX RANGE SCAN</td>
<td>C_MC</td>
<td>1</td>
<td>1</td>
<td>1</td>
</tr>
</tbody>
</table>

```
<table>
<thead>
<tr>
<th>Make</th>
<th>Model</th>
<th>Color</th>
</tr>
</thead>
<tbody>
<tr>
<td>BMW</td>
<td>530xi</td>
<td>RED</td>
</tr>
<tr>
<td>BMW</td>
<td>530xi</td>
<td>BLACK</td>
</tr>
<tr>
<td>BMW</td>
<td>530xi</td>
<td>SILVER</td>
</tr>
<tr>
<td>PORSCHE</td>
<td>911</td>
<td>RED</td>
</tr>
<tr>
<td>MERC</td>
<td>SLK</td>
<td>BLACK</td>
</tr>
<tr>
<td>MERC</td>
<td>C320</td>
<td>SLIVER</td>
</tr>
</tbody>
</table>
```
Example

correlated columns, no extended statistics

SELECT ......FROM.. WHERE model = '530xi'
AND make = 'BMW';

<table>
<thead>
<tr>
<th>Make</th>
<th>Model</th>
<th>Color</th>
</tr>
</thead>
<tbody>
<tr>
<td>BMW</td>
<td>530xi</td>
<td>RED</td>
</tr>
<tr>
<td>BMW</td>
<td>530xi</td>
<td>BLACK</td>
</tr>
<tr>
<td>BMW</td>
<td>530xi</td>
<td>SILVER</td>
</tr>
<tr>
<td>PORSCHE</td>
<td>911</td>
<td>RED</td>
</tr>
<tr>
<td>MERC</td>
<td>SLK</td>
<td>BLACK</td>
</tr>
<tr>
<td>MERC</td>
<td>C320</td>
<td>SILVER</td>
</tr>
</tbody>
</table>

• Three records selected.
  • Correlated columns
  • Additional predicate has no effect
  • Single column statistics are NOT sufficient

<table>
<thead>
<tr>
<th>Id</th>
<th>Operation</th>
<th>Name</th>
<th>Starts</th>
<th>E-Rows</th>
<th>A-Rows</th>
</tr>
</thead>
<tbody>
<tr>
<td>* 1</td>
<td>INDEX RANGE SCAN</td>
<td>C_MC</td>
<td>1</td>
<td>1</td>
<td>3</td>
</tr>
</tbody>
</table>
Example
correlated columns, extended statistics

SELECT ......FROM..
WHERE model = '530xi'
AND make = 'BMW';

<table>
<thead>
<tr>
<th>Make</th>
<th>Model</th>
<th>Color</th>
</tr>
</thead>
<tbody>
<tr>
<td>BMW</td>
<td>530xi</td>
<td>RED</td>
</tr>
<tr>
<td>BMW</td>
<td>530xi</td>
<td>BLACK</td>
</tr>
<tr>
<td>BMW</td>
<td>530xi</td>
<td>SILVER</td>
</tr>
<tr>
<td>PORSCHE</td>
<td>911</td>
<td>RED</td>
</tr>
<tr>
<td>MERC</td>
<td>SLK</td>
<td>BLACK</td>
</tr>
<tr>
<td>MERC</td>
<td>C320</td>
<td>SILVER</td>
</tr>
</tbody>
</table>

• Three records selected.
  • Multi-column statistics solve the problem
Extended Statistics – in detail

• Use dbms_stats package
  • Create_extended_stats
    • Manually specify the group of columns
  • Show_extended_stats_name
    • Displays the system generated name for the column group
  • Drop_extended_stats
    • Drop a column group and all the statistics associated with it

• Monitor
  • New dictionary table user_stat_extensions
    • Shows sys generated name & actual column group desc
  • Look at dictionary table user_tab_col_statistics
    • New row with sys generated name will be add for each column group
Optimizer Statistics

Improved Efficiency and Quality
Improved Efficiency and Quality
New statistics gathering algorithm

Business problem

- “.. Compute statistics gives accurate results but takes too long ..”
- “.. Sampling is fast but not always accurate ..”
- “.. AUTO SAMPLE SIZE does not always work with data skew ..”

Solution

- New groundbreaking implementation
  - Faster than sampling
  - Accuracy comparable to compute statistics
- Used by default with AUTO_SAMPLE_SIZE value
- No need to use manual sampling anymore

FASTER AND BETTER
Speed of sampling with the accuracy of compute
Improved Efficiency and Quality
Incremental Statistics Maintenance

Business Requirement

• Gathering statistics on one partition (e.g. after a bulk load) causes a full scan of all partitions to gather global table statistics
• Extremely time consuming

Solution

• Gather statistics for touched partition(s) ONLY
• Table (global) statistics are built from partition statistics

Refreshed WITHOUT scanning the NON touched partitions
Adaptive Cursor Sharing

Enhanced Bind Peeking
Adaptive Cursor Sharing

Business Requirement

- The optimizer peeks bind values during plan selection
- Initial value of the binds determines the plan
- Same execution plan shared regardless of future bind values

One plan not always appropriate for all bind values
Example with 10g

```
SELECT ......FROM..
WHERE Job = :B1
Value of B1 = CLERK
```

![Table with data]

- If clerk is the bind value at hard parse five out six records will be selected.

<table>
<thead>
<tr>
<th>Id</th>
<th>Operation</th>
<th>Name</th>
<th>Starts</th>
<th>E-Rows</th>
<th>A-Rows</th>
</tr>
</thead>
<tbody>
<tr>
<td>* 1</td>
<td>TABLE ACCESS FULL</td>
<td>EMP</td>
<td>1</td>
<td>5</td>
<td>5</td>
</tr>
</tbody>
</table>
Example with 10g cont.

```sql
SELECT ....FROM..
WHERE Job = :B1
Value of B1 = VP
```

- If VP is the bind value at hard parse one out six records will be selected.

<table>
<thead>
<tr>
<th>Enname</th>
<th>Eno</th>
<th>Job</th>
</tr>
</thead>
<tbody>
<tr>
<td>KING</td>
<td>8739</td>
<td>VP</td>
</tr>
</tbody>
</table>

---

<table>
<thead>
<tr>
<th>Id</th>
<th>Operation</th>
<th>Name</th>
<th>Starts</th>
<th>E-Rows</th>
<th>A-Rows</th>
</tr>
</thead>
<tbody>
<tr>
<td>*</td>
<td>INDEX RANGE SCAN</td>
<td>IND_EMP_JOB</td>
<td>1</td>
<td>1</td>
<td>1</td>
</tr>
</tbody>
</table>

---
With 11g

SELECT …..FROM ..
WHERE Job = :B1

BOTH

**B1 = CLERK**

<table>
<thead>
<tr>
<th>Ename</th>
<th>Eno</th>
<th>Job</th>
</tr>
</thead>
<tbody>
<tr>
<td>SMITH</td>
<td>6973</td>
<td>CLERK</td>
</tr>
<tr>
<td>ALLEN</td>
<td>7499</td>
<td>CLERK</td>
</tr>
<tr>
<td>WARD</td>
<td>7521</td>
<td>CLERK</td>
</tr>
<tr>
<td>SCOTT</td>
<td>7788</td>
<td>CLERK</td>
</tr>
<tr>
<td>CLARK</td>
<td>7782</td>
<td>CLERK</td>
</tr>
</tbody>
</table>

**B1 = VP**

<table>
<thead>
<tr>
<th>Ename</th>
<th>Eno</th>
<th>Job</th>
</tr>
</thead>
<tbody>
<tr>
<td>KING</td>
<td>8739</td>
<td>VP</td>
</tr>
</tbody>
</table>

Index Access is optimal

Full Table Scan is optimal

Peek all binds & take the plan that is optimal for each bind set
Adaptive Cursor Sharing

Solution

- Share the plan when binds values are “equivalent”
  - Plans are marked with selectivity range
  - If current bind values fall within range they use the same plan
- Create a new plan if binds are not equivalent
  - Generating a new plan with a different selectivity range
Adaptive Cursor Sharing – in detail

• Controlled by init.ora parameter
  • `_optim.peek_user.binds`
    • Determines if the optimizer will peek at bind values
    • Set to TRUE by default in 11gR1

• Monitor
  • Two new view V$SQL_CS_HISTOGRAM & V$SQL_CS_SELECTIVITY
  • V$SQL has 2 new columns
    • IS_BIND_SENSITIVE
    • IS_BIND_AWARE