Inside the Oracle Database 11g Optimizer
Removing the mystery

Maria Colgan – Principal Product Manager
Mohamed Zalt - Optimizer Architect
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.
Inside the Oracle Database 11g Optimizer

Removing the Mystery

- Plans change unexpectedly especially during upgrades
- Cardinality estimate is wrong so plan goes wrong
- Gathering Optimizer Statistics takes too long
- Bind peeking doesn’t work when there is a data skew
Inside the Oracle Database 11g Optimizer
Removing the Mystery

- Plans change unexpectedly especially during upgrades
  - Guaranteed plan stability and controlled plan evolution
  - Controlled statistics publication
- Cardinality estimate is wrong so plan goes wrong
  - Collect appropriate statistics
  - Eliminate wrong cardinality estimates
- Gathering Optimizer Statistics takes too long
  - Faster statistics gathering
  - Improved statistics quality
- Bind peeking doesn’t work when there is a data skew
  - Enhanced plan sharing with binds
SQL Plan Management

Guaranteed plan stability and controlled plan evolution
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
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”

- 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
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”
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”

![Diagram of SQL Plan Management process]

**Plan Acceptable**
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

Plans which don’t perform as good as the original plan stay in the plan history and are marked unaccepted

Plans which perform as good as or better than original plan are added to the plan baseline
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
Possible SQL Plan Manageability Scenarios

Database Upgrade using SQL Tuning Sets

Oracle Database 11g

Plan History
Plan
Baseline

No plan regressions

Oracle Database 10g

Well tuned plan

New Application Deployment, no Tuning Pack required

Production Database

Plan History
Plan
Baseline

No plan regressions

Development Database 11g

Plan History
Plan
Baseline

Well tuned plan

Baseline plans staging table

DBA
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

```sql
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>SLIVER</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'

<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>

- One record selected.
  - No correlated columns
  - Additional predicate reduces result set
  - Single column statistics are sufficient
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
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>SLIVER</td>
</tr>
</tbody>
</table>

- Three records selected.
  - Multi-column statistics solve the problem

```sql
<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>1</td>
<td>TABLE ACCESS FULL</td>
<td>CARS</td>
<td>1</td>
<td>3</td>
</tr>
</tbody>
</table>
```

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
Incremental Global Statistics

1. Partition level stats are gathered & synopsis created

Sales Table
- May 18th 2008
- May 19th 2008
- May 20th 2008
- May 21st 2008
- May 22nd 2008
- May 23rd 2008

- S1
- S2
- S3
- S4
- S5
- S6

2. Global stats generated by aggregating partition synopsis

Global Statistic

Sysaux Tablespace
Incremental Global Statistics Cont’d

3. A new partition is added to the table & data is loaded

Sales Table

- May 18th 2008
- May 19th 2008
- May 20th 2008
- May 21st 2008
- May 22nd 2008
- May 23rd 2008
- May 24th 2008

S1
S2
S3
S4
S5
S6

4. Global stats generated by aggregating the original partition synopsis with the new one

5. Retrieve synopsis for each of the other partitions from Sysaux

6. Global stats generated by aggregating the original partition synopsis with the new one

Global Statistic

Sysaux Tablespace
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

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

- 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>*</td>
<td>1</td>
<td>TABLE ACCESS FULL</td>
<td>EMP</td>
<td>1</td>
<td>5</td>
</tr>
</tbody>
</table>
Example with 10g cont.

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

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

http://search.oracle.com

Oracle optimizer

OR

http://www.optimizermagic.blogspot.com/

Optimizer development team’s blog
If you have more questions later, feel free to ask.
## Campground Demos

<table>
<thead>
<tr>
<th>Demo</th>
<th>Location</th>
</tr>
</thead>
<tbody>
<tr>
<td>Oracle Database 11g Optimizer - booth O27</td>
<td>Moscone West Exhibit Hall</td>
</tr>
</tbody>
</table>