Adopting the Adaptive Query Optimizer in Oracle 12c

Janis Griffin
Senior DBA
Who Am I?

• Senior DBA for Confio Software (Now Solarwinds)
  – JanisGriffin@confio.com
  – @DoBoutAnything

• Current - 25+ Years in Oracle
• Former – Database Design & Implementation
• Specialize in performance tuning
• Review database performance for customers and prospects
• Common thread – How do I tune it?
12c New Optimizer Features (+ a little bit of history :-)

- Adaptive Plans
  - What are they
  - How to Display

- Adaptive Statistics
  - Dynamic Statistics & Reoptimization
  - Best Practice in Gathering
  - New Histograms

- SQL Plan Management
  - What’s new
  - How it coexists with adaptive plans
In the Beginning…

- Rule Based Optimizer (Version <= 6)
  - Rules based on 17 possible access paths
  - Only one Execution Plan chosen based on ranking of rules
  - Simple rewrites of ‘OR’ to ‘Union ALL’

- Cost Based Optimizer (Version >= 7.3)
  - Multiple plans generated with estimated cost of IO/CPU
    - Plan with lowest cost chosen
  - Allowed for Hash joins, Histograms, Partitioning & Parallel queries
  - More complex rewrites / transformations
  - Required statistics gathering / Plans Changed
    - 8.1.7, Stored Outlines to control plan changes
    - 9.2, Dynamic sampling of Statistics
    - 10g, SQL Profiles / Tuning Advisor
      - DBMS_SQLTUNE – Costs $$$
    - Oracle 11, Adaptive Cursor Sharing / SQL Plan Management
How the Optimizer Works

1. **Parsed Query (from Parser)**
2. **Query Transformer** – rewrites query to be more efficient
3. **Estimator** – looks at selectivity, cardinality & cost
4. **Plan Generator** – creates multiple plans using different access paths & join types. Plan with lowest cost is chosen
5. **Data Dictionary**
   - Schema Definition
   - Statistics

---

*Init.ora parameter to control behavior:* 
**OPTIMIZER_FEATURES_ENABLED**
Execution Plans

• Show the **sequence of operations** performed to run SQL statements
  
  – Order of the tables referenced in the statement
  
  – Access method for each table in the statement
    • INDEX
    • INLIST ITERATOR
    • TABLE ACCESS
  
  – Join method in statement accessing multiple tables
    • HASH JOIN
    • MERGE JOIN
    • NESTED LOOPS
  
  – Data manipulations
    • CONCATENATION
    • FILTER
    • SORT
  
  – Statistic Collectors
    • New in 12C
Adaptive Query Optimizer

- Allows for run-time adjustments to execution plans
- Can discover additional information
  - which can lead to better statistics & optimal plans
Adaptive Plans

• Help the optimizer make final plan choices at Execution time
  – Optimizer instruments the default plan with statistics collectors
  – At runtime, estimates are compared with actual rows buffered (per operation)
  – If different, the plan (or step) is adapted to avoid poor performance
  – Works only on Join Method & Parallel Distribution Method operations
    • Can switch between nested loops and hash joins
    • If initial join is a sort merge join, no adaptation will occur

• How to identify Adaptive Plans?
  – New column in V$SQL: IS_RESOLVED_ADAPTIVE_PLAN
    • If ‘Y’, the plan was adapted & is the final plan
    • If ‘N’, the plan is adaptive but the final plan has not be selected yet
    • If NULL, the plan is non-adaptive
Control & View Adaptations

- Init.ora parameters that control Adaptive Plans

<table>
<thead>
<tr>
<th>Name</th>
<th>Type</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>optimizer_adaptive_features</td>
<td>boolean</td>
<td>TRUE</td>
</tr>
<tr>
<td>optimizer_adaptive_reporting_only</td>
<td>boolean</td>
<td>FALSE</td>
</tr>
<tr>
<td>optimizer_features_enable</td>
<td>string</td>
<td>12.1.0.1</td>
</tr>
<tr>
<td>optimizer_dynamic_sampling</td>
<td>integer</td>
<td>2</td>
</tr>
</tbody>
</table>

- Use DBMS_XPLAN.DISPLAY_CURSOR to view Optimizer Adaptations
  - Explain Plan (dbms_xplan.display) may only show default or initial plan
    • Be Careful!
  - Use format parameter ‘+report’ for testing
    • Shows what the adaptive plan would be but doesn’t use it
    ```sql
    select * from table(dbms_xplan.display_cursor ('&sql_id',&child,format=>'+report'));
    ```
  - Use format parameter ‘+adaptive’ to see all steps (active / inactive)
    • including optimizer statistics collectors
    ```sql
    select * from table(dbms_xplan.display_cursor ('&sql_id',&child,format=>'+adaptive'));
    ```
```
SELECT sql_id, child_number, plan_hash_value,
    is_resolved_adaptive_plan, substr(sql_text, 1,30) sql_text
FROM v$sql
WHERE sql_id in (SELECT sql_id FROM v$session
    WHERE username = 'SOE')
ORDER BY sql_id,child_number,is_resolved_adaptive_plan;
```
SELECT sql_id, child_number, 
    SUBSTR(sql_text, 1,30) sql_text, 
    IS_RESOLVED_ADAPTIVE_PLAN, 
    IS_REOPTIMIZABLE 
FROM v$sql 
WHERE sql_text like 'select /* jg */%' 
ORDER BY sql_id,child_number

- IS_REOPTIMIZABLE is for next execution
  - Y - the next execution will trigger a reoptimization
  - R – has reoptimization info but won’t trigger due to reporting mode
  - N -the child cursor has no reoptimization info
alter session set optimizer_adaptive_reporting_only=TRUE;
select * from table(dbms_xplan.display_cursor('8qpakg674n4mz',0,format=>'+report'));
Adaptive plan:

This cursor has an adaptive plan, but adaptive plans are enabled for reporting mode only. The plan that would be executed if adaptive plans were enabled is displayed below.

Plan hash value: 158447987

<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></td>
<td></td>
</tr>
<tr>
<td>1</td>
<td>NESTED LOOPS</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>2</td>
<td>NESTED LOOPS</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>3</td>
<td>TABLE ACCESS FULL</td>
<td>ORDER_ITEMS</td>
<td>1895</td>
<td>73905</td>
<td>13184 (100)</td>
<td>00:00:01</td>
</tr>
<tr>
<td>4</td>
<td>INDEX RANGE SCAN</td>
<td>PRODUCT_IDX</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>5</td>
<td>TABLE ACCESS BY INDEX ROWID</td>
<td>PRODUCT</td>
<td>1</td>
<td>28</td>
<td>1314 (2)</td>
<td>00:00:01</td>
</tr>
</tbody>
</table>

Predicate Information (identified by operation id):

3 - filter(("O"."UNIT_PRICE"=:B1 AND "O"."QUANTITY">:B2))
4 - access("O"."PRODUCT_ID"=:P"PRODUCT_ID")

Note
---
- this is an adaptive plan

Reoptimized plan:

This cursor is marked for automatic reoptimization, but automatic reoptimization is enabled for reporting mode only. The plan that would be selected on the next execution if automatic reoptimization were enabled is displayed below.

Plan hash value: 3627148456

<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>323</td>
<td>12597</td>
<td>12468 (3)</td>
<td>00:00:01</td>
</tr>
<tr>
<td>1</td>
<td>NESTED LOOPS</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>2</td>
<td>NESTED LOOPS</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>3</td>
<td>TABLE ACCESS FULL</td>
<td>ORDER_ITEMS</td>
<td>323</td>
<td>12597</td>
<td>12468 (3)</td>
<td>00:00:01</td>
</tr>
<tr>
<td>4</td>
<td>INDEX RANGE SCAN</td>
<td>PRODUCT_IDX</td>
<td>174</td>
<td>1914</td>
<td>11946 (3)</td>
<td>00:00:01</td>
</tr>
<tr>
<td>5</td>
<td>TABLE ACCESS BY INDEX ROWID</td>
<td>PRODUCT</td>
<td>2</td>
<td>56</td>
<td>3 (0)</td>
<td>00:00:01</td>
</tr>
</tbody>
</table>
Shows wrong plan in Report Mode!

SQL> explain plan for 
2 select /* jg */ p.product_name 
3 from order_items o, product p 
4 where o.unit_price = :b1 
5 and o.quantity > :b2 
6 and o.product_id = p.product_id 
7 /

Explained.

SQL> select * from table (dbms_xplan.display());

PLAN_TABLE_OUTPUT

Plan hash value: 3627148456

<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>174</td>
<td>6786</td>
<td>12468</td>
<td>(3)</td>
</tr>
<tr>
<td>1</td>
<td>NESTED LOOPS</td>
<td></td>
<td>174</td>
<td>6786</td>
<td>12468</td>
<td>(3)</td>
</tr>
<tr>
<td>2</td>
<td>NESTED LOOPS</td>
<td></td>
<td>174</td>
<td>1914</td>
<td>11946</td>
<td>(3)</td>
</tr>
<tr>
<td>3</td>
<td>TABLE ACCESS FULL</td>
<td>ORDER_ITEMS</td>
<td>174</td>
<td>1914</td>
<td>11946</td>
<td>(3)</td>
</tr>
<tr>
<td>4</td>
<td>INDEX RANGE SCAN</td>
<td>PRODUCT_IDX</td>
<td>1</td>
<td>2</td>
<td>3</td>
<td>(0)</td>
</tr>
<tr>
<td>5</td>
<td>TABLE ACCESS BY INDEX ROWID</td>
<td>PRODUCT</td>
<td>1</td>
<td>28</td>
<td>3</td>
<td>(0)</td>
</tr>
</tbody>
</table>

Predicate Information (identified by operation id):

3 - filter("O"."UNIT_PRICE"=TO_NUMBER(:B1) AND "O"."QUANTITY">TO_NUMBER(:B2))
4 - access("O"."PRODUCT_ID"="P"."PRODUCT_ID")

Note
- this is an adaptive plan
Shows all steps in v$sql_plan & OEM without distinguishing what is real!

Plan Hash: 158447987

- SELECT STATEMENT Optimizer=ALL_ROWS (Cost=13160)
- HASH JOIN (Cost=13160 Cardinality=879 Bytes=33402)
- NESTED LOOPS
- NESTED LOOPS (Cost=13160 Cardinality=879 Bytes=33402)
- STATISTICS COLLECTOR
  - TABLE ACCESS (FULL) OF ORDER_ITEMS (TABLE) (Cost=11861 Cardinality=428 Bytes=4708)
  - INDEX (RANGE SCAN) OF PRODUCT_ID (INDEX)
- TABLE ACCESS (BY INDEX ROWID) OF PRODUCT (TABLE) (Cost=1300 Cardinality=2 Bytes=54)
- TABLE ACCESS (FULL) OF PRODUCT (TABLE) (Cost=1300 Cardinality=1000 Bytes=27000)

Arrows point to inactive steps

Unless you review ‘OTHER_XML’ column
View Actual Adaptive Plan

Adapted on first execution

```sql
alter session set optimizer_adaptive_reporting_only=FALSE;
```

```sql
SQL> select * from table(dbms_xplan.display_cursor('8qpakg674n4mz',1,format=>'+adaptive'));
```

```
SQL_ID 8qpakg674n4mz, child number 1

select /* jg */ p.product_name from order_items o, product p
where o.unit_price = :b1 and o.quantity > :b2 and o.product_id = p.product_id

Plan hash value: 3627148456

<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></td>
<td></td>
</tr>
<tr>
<td>1</td>
<td>HASH JOIN</td>
<td></td>
<td>1895</td>
<td>73905</td>
<td>13184 (100)</td>
<td>00:00:01</td>
</tr>
<tr>
<td>2</td>
<td>NESTED LOOPS</td>
<td></td>
<td>1895</td>
<td>73905</td>
<td>13184 (3)</td>
<td>00:00:01</td>
</tr>
<tr>
<td>3</td>
<td>NESTED LOOPS</td>
<td></td>
<td>1895</td>
<td>73905</td>
<td>13184 (3)</td>
<td>00:00:01</td>
</tr>
<tr>
<td>4</td>
<td>STATISTICS COLLECTOR</td>
<td></td>
<td>1895</td>
<td>73905</td>
<td>11862 (3)</td>
<td>00:00:01</td>
</tr>
<tr>
<td>5</td>
<td>TABLE ACCESS FULL</td>
<td>ORDER_ITEMS</td>
<td>1895</td>
<td>20845</td>
<td>1314 (2)</td>
<td>00:00:01</td>
</tr>
<tr>
<td>6</td>
<td>INDEX RANGE SCAN</td>
<td>PRODUCT_IDX</td>
<td>1</td>
<td>28</td>
<td>1314 (2)</td>
<td>00:00:01</td>
</tr>
<tr>
<td>7</td>
<td>TABLE ACCESS BY INDEX ROWID</td>
<td>PRODUCT</td>
<td>1022K</td>
<td>27M</td>
<td>1314 (2)</td>
<td>00:00:01</td>
</tr>
</tbody>
</table>

Predicate Information (identified by operation id):

- access("O"."PRODUCT_ID"="P"."PRODUCT_ID")
- filter("O"."UNIT_PRICE"=:B1 AND "O"."QUANTITY">=:B2)
- access("O"."PRODUCT_ID"="P"."PRODUCT_ID")

Note
---
- this is an adaptive plan (rows marked '-' are inactive)
```
What Changed?

After Reoptimization has occurred

```sql
SQL> select * from table(dbms_xplan.display_cursor('8qpakg674n4mz',2,format=>'+adaptive'));
```

<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></td>
<td></td>
</tr>
<tr>
<td>1</td>
<td>HASH JOIN</td>
<td></td>
<td>1</td>
<td>39</td>
<td>13184 (100)</td>
<td>00:00:01</td>
</tr>
<tr>
<td>2</td>
<td>NESTED LOOPS</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>3</td>
<td>NESTED LOOPS</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>4</td>
<td>STATISTICS COLLECTOR</td>
<td></td>
<td>1</td>
<td>39</td>
<td>3184 (3)</td>
<td>00:00:01</td>
</tr>
<tr>
<td>5</td>
<td>TABLE ACCESS FULL</td>
<td>ORDER_ITEMS</td>
<td>1895</td>
<td>20845</td>
<td>11862 (3)</td>
<td>00:00:01</td>
</tr>
<tr>
<td>6</td>
<td>INDEX RANGE SCAN</td>
<td>PRODUCT_IDX</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>7</td>
<td>TABLE ACCESS BY INDEX ROWID</td>
<td>PRODUCT</td>
<td>1</td>
<td>28</td>
<td>1314 (2)</td>
<td>00:00:01</td>
</tr>
<tr>
<td>8</td>
<td>TABLE ACCESS FULL</td>
<td>PRODUCT</td>
<td>1022K</td>
<td>27M</td>
<td>1314 (2)</td>
<td>00:00:01</td>
</tr>
</tbody>
</table>

Predicate Information (identified by operation id):

1 - access("O","PRODUCT_ID"="P","PRODUCT_ID")
5 - filter("O"."UNIT_PRICE"=:b1 AND "O"."QUANTITY">:b2)
6 - access("O"."PRODUCT_ID"="P","PRODUCT_ID")

Note:
- statistics feedback used for this statement
- this is an adaptive plan (rows marked '-' are inactive)
Why Plans Changes

Execution plans can change as underlying inputs to the optimizer change.

– Same Sql – Different Schemas
  • Different table sizes / statistics / indexes

– Same Sql – Different Costs
  • Data volume & Statistic Changes over time
  • Bind variable types and values
  • Initialization parameters (set globally or session level)
  • Adaptive Cursor Sharing – 11G
    – V$SQL - IS_BIND_SENSITIVE: optimizer peeked – plan may change
    – V$SQL - IS_BIND_AWARE: ‘Y’ after query has been marked bind sensitive
  • Adaptive Plans / Statistics – 12C

– V$SQL_SHARED_CURSOR
  • Can give clues to why plan changed
  • 70 columns showing mismatches / differences
  • Hard to view
V$SQL_SHARED_CURSOR via ‘shared_proc.sql’ (see appendix)
• Parallel execution needs to distribute data across all parallel processes
  • For sorts, aggregation & join operations
  • Chosen method depends on number of rows & Degree of Parallelism (DOP)

• Potential performance problem if few parallel processes distribute many rows
  • Data skew could cause unequal distribution of rows

• New Hybrid Hash distribution technique
  • Optimizer decides final data distribution method during execution time
  • Statistic collectors are inserted in front of the parallel server processes
    • On producer side of the operation.
  • Chooses:
    • Hash, if rows > than threshold
    • Broadcast, if rows < than threshold
    • Threshold defined as 2 X DOP
Parallel Distribution

Uses Hybrid Hash - 75,329 rows greater than threshold of 40 (2 x 20 DOP = 40)
Why Adaptive Statistics

- Complex queries require more info than base table statistics
- Optimizer augments with these adaptive statistics techniques

Dynamic Statistics

• Augment missing or base table statistics
  – Table & index block counts
  – Table & join cardinalities (estimated number of rows)
  – Join column statistics
  – GROUP BY statistics

• Are gathered during the parse stage
  – Uses recursive SQL to scan a random sample of table blocks

• Controlled by dynamic sampling init.ora parameter
  – OPTIMIZER_DYNAMIC_SAMPLING

• New in 12c - level 11
  – controls the creation of dynamic statistics

alter session set OPTIMIZER_DYNAMIC_SAMPLING = 11;
Dynamic Statistics Example

SQL> EXPLAIN PLAN FOR
2 SELECT product_id, product_name, product_description
3 FROM product_information
4 WHERE category_id IN (12,126)
5 AND product_name LIKE '%j%';

Explain.

SQL> select * from table (dbms_xplan.display());

Plan hash value: 2715330242

<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>PRODUCT_INFORMATION</td>
<td>65537</td>
<td>8064K</td>
<td>9169 (1)</td>
<td>00:00:01</td>
</tr>
<tr>
<td>1</td>
<td>TABLE ACCESS FULL</td>
<td>PRODUCT_INFORMATION</td>
<td>65537</td>
<td>8064K</td>
<td>9169 (1)</td>
<td>00:00:01</td>
</tr>
</tbody>
</table>

Predicate Information (identified by operation id):

1 - filter("PRODUCT_NAME" LIKE '%j%'

SQL> alter session set optimizer_dynamic_sampling = 11;

Session altered.

SQL> EXPLAIN PLAN FOR
2 SELECT product_id, product_name, product_description
3 FROM product_information
4 WHERE category_id IN (12,126)
5 AND product_name LIKE '%j%';

Explain.

SQL> select * from table (dbms_xplan.display());

Plan hash value: 2715330242

<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>PRODUCT_INFORMATION</td>
<td>349K</td>
<td>42M</td>
<td>9169 (1)</td>
<td>00:00:01</td>
</tr>
<tr>
<td>1</td>
<td>TABLE ACCESS FULL</td>
<td>PRODUCT_INFORMATION</td>
<td>349K</td>
<td>42M</td>
<td>9169 (1)</td>
<td>00:00:01</td>
</tr>
</tbody>
</table>

Predicate Information (identified by operation id):

1 - filter("PRODUCT_NAME" LIKE '%j%'

Note
dynamic statistics used: dynamic sampling (level=AUTO)

Notes:
Parse time takes longer.
Results are persisted & used elsewhere

Estimates over 6X off!
Automatic Reoptimization

- Different from Adaptive Plans
- Optimizer can reoptimize a query several times
  - Learning more info & further improving the plan
- Uses Statistics Feedback
- Or Performance Feedback to change the plan
Statistics Feedback

• Estimated cardinalities can be incorrect for many reasons
  – missing statistics, inaccurate statistics, or complex predicates

• Optimizer enables monitoring for statistics feedback when
  – Tables have no statistics
  – Multiple conjunctive or disjunctive filter predicates are on a table
  – Predicates contain complex operators so optimizer can’t compute estimates

• After 1st execution, estimates are compared with actual rows
  – If they differ significantly, optimizer stores correct estimates for future use

• Can create a SQL PLAN DIRECTIVE for other SQL statements
  – They benefit from information obtained during initial execution

• After 1st execution, optimizer disables statistics feedback
Statistics Feedback

```sql
SELECT * from table(DBMS_XPLAN.DISPLAY_CURSOR(FORMAT=>'ALLSTATS LAST'));
```

### Statistics Feedback

**Plan hash value:** 218532850

<table>
<thead>
<tr>
<th>Id</th>
<th>Operation</th>
<th>Name</th>
<th>Start</th>
<th>E-Rows</th>
<th>A-Rows</th>
<th>A-Time</th>
<th>Buffers</th>
<th>Reads</th>
<th>OMem</th>
<th>IMem</th>
<th>Used-Mem</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>SELECT STATEMENT</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>1</td>
<td>HASH JOIN</td>
<td></td>
<td>1</td>
<td>5</td>
<td>5</td>
<td>00:00:10.11</td>
<td>190K</td>
<td>145K</td>
<td>3483K</td>
<td>1239K</td>
<td>4462K (0)</td>
</tr>
<tr>
<td>2</td>
<td>HASH JOIN</td>
<td></td>
<td>1</td>
<td>3</td>
<td>26077</td>
<td>00:00:09.00</td>
<td>180K</td>
<td>140K</td>
<td>2324K</td>
<td>1378K</td>
<td>2781K (0)</td>
</tr>
<tr>
<td>3</td>
<td>MERGE JOIN CARTESIAN</td>
<td></td>
<td>1</td>
<td>1</td>
<td>18821</td>
<td>00:00:28.86</td>
<td>121K</td>
<td>81694</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>4</td>
<td>NESTED LOOPS</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>5</td>
<td>TABLE ACCESS FULL</td>
<td>CUSTOMERS</td>
<td>1</td>
<td>1</td>
<td>18821</td>
<td>00:00:27.27</td>
<td>79141</td>
<td>67659</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>6</td>
<td>INDEX RANGE SCAN</td>
<td>ORD_CUSTOMER_IX</td>
<td>9440</td>
<td>2</td>
<td>18821</td>
<td>00:00:27.52</td>
<td>80395</td>
<td>63700</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>7</td>
<td>TABLE ACCESS BY INDEX ROWID</td>
<td>ORDERS</td>
<td>18821</td>
<td>1</td>
<td>18821</td>
<td>00:00:27.80</td>
<td>1824</td>
<td>9051</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>8</td>
<td>BUFFER SORT</td>
<td>Vw_NSO_1</td>
<td>18821</td>
<td>1</td>
<td>18821</td>
<td>00:00:31.04</td>
<td>23225</td>
<td>4984</td>
<td>2048</td>
<td>2048</td>
<td>2048 (0)</td>
</tr>
<tr>
<td>9</td>
<td>VIEW</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>10</td>
<td>HASH UNIQUE</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>11</td>
<td>HASH JOIN SEMI</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>12</td>
<td>TABLE ACCESS FULL</td>
<td>PRODUCT</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>00:00:00.96</td>
<td>23225</td>
<td>4984</td>
<td>2294K</td>
<td>2294K</td>
<td>435K (0)</td>
</tr>
<tr>
<td>13</td>
<td>TABLE ACCESS FULL</td>
<td>INVENTORIES</td>
<td>1</td>
<td>125K</td>
<td>1083K</td>
<td>00:00:00.50</td>
<td>13376</td>
<td>0</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>14</td>
<td>TABLE ACCESS FULL</td>
<td>ORDER_ITEMS</td>
<td>1</td>
<td>3</td>
<td>12M</td>
<td>00:00:05.41</td>
<td>59010</td>
<td>58760</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>15</td>
<td>TABLE ACCESS FULL</td>
<td>PRODUCT</td>
<td>1</td>
<td>117K</td>
<td>111K</td>
<td>00:00:05.41</td>
<td>59010</td>
<td>58760</td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

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

1. access("I"."PRODUCT_ID"="P"."PRODUCT_ID" AND "P"."PRODUCT_ID"="PRODUCT_ID")
2. access("O"."ORDER_ID"="I"."ORDER_ID")
6. filter("C"."CUST_FIRST_NAME" LIKE '%gig%' OR "C"."CUST_LAST_NAME" LIKE '%gri%')
7. access("C"."CUSTOMER_ID"="O"."CUSTOMER_ID")
12. access("I"."PRODUCT_ID"="P"."PRODUCT_ID")
13. filter("PRODUCT_NAME" IS NOT NULL AND "PRODUCT_NAME" LIKE '%jhtqiwxr5E%')
14. filter("I"."QUANTITY_ON_HAND">10)

**Note:**
- dynamic statistics feedback used: dynamic sampling (level-2)
- statistics feedback used for this statement
- this is an adaptive plan
• Automatically improves the degree of parallelism
  • Init.ora parameter, PARALLEL_DEGREE_POLICY = ’ADAPTIVE’

• On 1\textsuperscript{st} execution, the optimizer decides
  • Whether to execute the statement in parallel
  • The degree of parallelism based on estimates

• After 1\textsuperscript{st} execution, optimizer compares
  • Estimates with actual performance statistics
    • e.g. CPU Time
    • i.e. PARALLEL_MIN_TIME_THRESHOLD
  • If significantly different, the statement
    • is marked for reparsing
    • new execution statistics are stored as feedback

• Following executions use the performance feedback to determine DOP

• If PARALLEL_DEGREE_POLICY not set, statistics feedback may change DOP
Performance Feedback

Alter session set PARALLEL_DEGREE_POLICY = 'ADAPTIVE';

```
SQL> select * from table(CBMS_XPLAN.DISPLAY_CURSOR('13Sprg8vt6d9','2',FORMAT=>'ALLSTATS LAST'));
```

<table>
<thead>
<tr>
<th>Id</th>
<th>Operation</th>
<th>Name</th>
<th>Start</th>
<th>E-Rows</th>
<th>A-Rows</th>
<th>A-Time</th>
<th>Buffers</th>
<th>Reads</th>
<th>OMem</th>
<th>1Mem</th>
<th>Used-Mem</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>SELECT STATEMENT</td>
<td></td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>1</td>
<td>PX COORDINATOR</td>
<td>:TQ10001</td>
<td>0</td>
<td>4966K</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>2</td>
<td>PX SEND QC (RANDOM)</td>
<td></td>
<td>0</td>
<td>4966K</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>3</td>
<td>HASH JOIN</td>
<td></td>
<td>0</td>
<td>4966K</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>4</td>
<td>PX RECEIVE</td>
<td>:TQ10000</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>5</td>
<td>PX SEND BROADCAST</td>
<td>:TQ10000</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>6</td>
<td>NESTED LOOPS</td>
<td></td>
<td>1</td>
<td>3786</td>
<td>525</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>7</td>
<td>NESTED LOOPS</td>
<td></td>
<td>1</td>
<td>2743</td>
<td>385</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>8</td>
<td>NESTED LOOPS</td>
<td></td>
<td>1</td>
<td>2743</td>
<td>385</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>9</td>
<td>PX BLOCK ITERATOR</td>
<td>CUSTOMERS</td>
<td>1</td>
<td>1152</td>
<td>201</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>10</td>
<td>TABLE ACCESS FULL</td>
<td>ORDER_ITEMS</td>
<td>23</td>
<td>1</td>
<td>525</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>11</td>
<td>TABLE ACCESS BY INDEX ROWID</td>
<td>ORDER_ITEMS</td>
<td>23</td>
<td>1</td>
<td>525</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>12</td>
<td>INDEX RANGE SCAN</td>
<td>ORDER_ITEMS</td>
<td>23</td>
<td>1</td>
<td>525</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>13</td>
<td>INDEX RANGE SCAN</td>
<td>ORDER_ITEMS</td>
<td>23</td>
<td>1</td>
<td>525</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>14</td>
<td>TABLE ACCESS BY INDEX ROWID</td>
<td>ORDER_ITEMS</td>
<td>23</td>
<td>1</td>
<td>525</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>15</td>
<td>PX BLOCK ITERATOR</td>
<td>PRODUCT</td>
<td>0</td>
<td>1311K</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>16</td>
<td>TABLE ACCESS FULL</td>
<td></td>
<td>0</td>
<td>1311K</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
</tbody>
</table>

Predicate Information (identified by operation id):

3  - access('I','.PRODUCT_ID='P','.PRODUCT_ID')
10 - access(':Z=-2 AND :Z=x=-2')
12 - access('C','.CUSTOMER_ID='O','.CUSTOMER_ID')
13 - access('O','.ORDER_ID='I','.ORDER_ID')
16 - access(':Z=-2 AND :Z=x=-2')

Note
- dynamic statistics used; dynamic sampling (level=AUTO)
- automatic DOP: Computed Degree of Parallelism is 8 because of degree limit
- parallel scans affinitized
SQL Plan Directives

• Are additional Instructions for missing column group statistics or histograms
  – Dynamic sampling performed on directive
    • Until statistics are gathered for the column group (e.g. City / State / Country)
• Not tied to a specific sql statement – defined on a query expression
  – Can be used by similar queries
• Are created in shared_pool & periodically written to the SYSAUX tablespace
  – DBA_SQL_PLAN_DIRECTIVES
  – DBA_SQL_PLAN_DIR_OBJECTS

SELECT TO_CHAR(d.directive_id) dir_id,
        o.owner, o.object_name, o.subobject_name col_name,
        o.object_type, d.type, d.state, d.reason
FROM dba_sql_plan_directives d, dba_sql_plan_dir_objects o
WHERE d.directive_id = o.directive_id
AND o.owner IN ('SOE') ORDER BY 1,2,3,4,5;
SQL Plan Directives

- Use DBMS_SPD package to manage
  - Can’t manually create directives

<table>
<thead>
<tr>
<th>Name</th>
<th>P / F</th>
<th>Task</th>
</tr>
</thead>
<tbody>
<tr>
<td>ALTER_SQL_PLAN_DIRECTIVE</td>
<td>P</td>
<td>Changes either STATE or AUTO_DROP</td>
</tr>
<tr>
<td>DROP_SQL_PLAN_DIRECTIVE</td>
<td>P</td>
<td>Drops SQL Plan Directive (SPD)</td>
</tr>
<tr>
<td>FLUSH_SQL_PLAN_DIRECTIVE</td>
<td>P</td>
<td>Flushes SQL Plan Directives out of SGA</td>
</tr>
<tr>
<td>CREATE_STGTAB_DIRECTIVE</td>
<td>P</td>
<td>Staging table for exporting SPDs</td>
</tr>
<tr>
<td>PACK_STGTAB_DIRECTIVE</td>
<td>F</td>
<td>Exports SPDs in to Staging table</td>
</tr>
<tr>
<td>UNPACK_STGTAB_DIRECTIVE</td>
<td>F</td>
<td>Imports SPDs from Staging table</td>
</tr>
<tr>
<td>GET_PREFS</td>
<td>F</td>
<td>Get setting for SPD_RETENTION_WEEKS</td>
</tr>
<tr>
<td>SET_PREFS</td>
<td>P</td>
<td>Sets SPD_RETENTION_WEEKS (default set to 53 weeks)</td>
</tr>
<tr>
<td>TRANSFER_SPD_FOR_DP</td>
<td>P</td>
<td>Undocumented</td>
</tr>
</tbody>
</table>
Example of dropping all directives for ‘SOE’ user

BEGIN
FOR get_rec in (SELECT distinct TO_CHAR(d.directive_id) dir_id
    FROM dba_sql_plan_directives d, dba_sql_plan_dir_objects o
    WHERE d.directive_id = o.directive_id
    AND o.owner in ('SOE')) LOOP
    DBMS_SPD.DROP_SQL_PLAN_DIRECTIVE(get_rec.dir_id);
end loop;
end;
/
commit;
Gathering Statistics

- DBMS_STATS package
  - Old ‘Analyze’ command deprecated (Still can use for row chaining)
  - Has many new procedures & functions (e.g. Session level stats for GTT)
  - Rewritten in 11g – faster & better AUTO_SAMPLE_SIZE
    - 100% in less time & more accurate than 10% estimate
    - Don’t use ESTIMATE_PERCENT

- GATHER_*_STATS procedures can take up to 13 parameters
  - Should only set 2-4 parameters (per Tom Kyte)
    - SCHEMA NAME
    - TABLE NAME
    - PARTITION NAME
    - DOP

- Histograms – tells optimizer of data skew
  - Default: any column in WHERE / GROUP BY clause which has a data skew
  - Controlled by METHOD_OPT parameter
  - Uses column info captured at compilation time in SYS.COL_USAGE$
12c Statistics Changes

• Basic statistics automatically gathered during
  • Index, CTAS & IAS Creation
• Histograms
  • 11g & before - 2 types of histograms
    • Frequency - Distinct values (DV) < 254 buckets
    • Height-balanced - Any DV that didn’t fit in < 254 buckets
  • 12c – 2 additional
    • Top Frequency
      • Problem: if specific values occupy most the rows (>99% rows)
      • But DV > 254 buckets
      • Ignores unpopular values so histogram works only on popular
    • Hybrid
      • Similar to height balanced as created if DV >254
      • Stores actual frequencies of bucket endpoints in histograms
        • More endpoints are stored in histogram
        • Achieves the same effect as increasing the # of buckets
    • Both only work when created with AUTO_SAMPLE_SIZE
### Top Frequency Histograms

#### Popular Values

```sql
SQL> select table_name, column_name, num_distinct, num_nulls, num_buckets, sample_size, histogram from user_tab_columns
where histogram != 'NONE'
order by table_name;
```

<table>
<thead>
<tr>
<th>TABLE_NAME</th>
<th>COLUMN_NAME</th>
<th>NUM_DISTINCT</th>
<th>NUM_NULLS</th>
<th>NUM_BUCKETS</th>
<th>SAMPLE_SIZE</th>
<th>HISTOGRAM</th>
</tr>
</thead>
<tbody>
<tr>
<td>CUSTOMERS</td>
<td>CUST_FIRST_NAME</td>
<td>5338</td>
<td>0</td>
<td>254</td>
<td>5435 HYBRID</td>
<td></td>
</tr>
<tr>
<td>CUSTOMERS</td>
<td>CUST_LAST_NAME</td>
<td>89248</td>
<td>0</td>
<td>254</td>
<td>5435 HYBRID</td>
<td></td>
</tr>
<tr>
<td>INVENTORIES</td>
<td>WAREHOUSE_ID</td>
<td>999</td>
<td>0</td>
<td>254</td>
<td>5463 HYBRID</td>
<td></td>
</tr>
<tr>
<td>ORDERS</td>
<td>ORDER_STATUS</td>
<td>10</td>
<td>0</td>
<td>10</td>
<td>7740027 FREQUENCY</td>
<td></td>
</tr>
<tr>
<td>ORDER_ITEMS</td>
<td>QUANTITY</td>
<td>18</td>
<td>66070</td>
<td>18</td>
<td>11260202 FREQUENCY</td>
<td></td>
</tr>
<tr>
<td>ORDER_ITEMS</td>
<td>LINE_ITEM_ID</td>
<td></td>
<td>0</td>
<td></td>
<td></td>
<td>11326272 FREQUENCY</td>
</tr>
<tr>
<td>PRODUCT</td>
<td>PRODUCT_NAME</td>
<td>1000</td>
<td>0</td>
<td>254</td>
<td>1311710 TOP-FREQUENCY</td>
<td></td>
</tr>
<tr>
<td>PRODUCT_INFO</td>
<td>CATEGORY_ID</td>
<td>196</td>
<td>0</td>
<td>196</td>
<td>1311710 TOP-FREQUENCY</td>
<td></td>
</tr>
<tr>
<td>PRODUCT_INFO</td>
<td>PRODUCT_STATUS</td>
<td>4</td>
<td>0</td>
<td>4</td>
<td>1311710 TOP-FREQUENCY</td>
<td></td>
</tr>
<tr>
<td>PRODUCT_INFO</td>
<td>PRODUCT_ID</td>
<td>1000</td>
<td>0</td>
<td>254</td>
<td>1311710 TOP-FREQUENCY</td>
<td></td>
</tr>
</tbody>
</table>

```sql
SQL> select product_name, count(*) from product
group by product_name
order by 2;
```

<table>
<thead>
<tr>
<th>PRODUCT_NAME</th>
<th>COUNT(*)</th>
</tr>
</thead>
<tbody>
<tr>
<td>mBC WF08Cy0e MRb46sDx</td>
<td>1</td>
</tr>
<tr>
<td>hps49faqjIUuMnalBjMj3jBCZ</td>
<td>1</td>
</tr>
<tr>
<td>RwL1B2mdUS_Ly AL4wkl 1iw</td>
<td>1</td>
</tr>
<tr>
<td>nCACupN3m4fCj o7ZiQdgupw23</td>
<td>1</td>
</tr>
<tr>
<td>iBLCu AjL</td>
<td>1</td>
</tr>
<tr>
<td>HQkUa7Gg 3vY5YV4ABBIFuD</td>
<td>1</td>
</tr>
<tr>
<td>Ruzt3Uw is2o UAdCxa</td>
<td>1</td>
</tr>
<tr>
<td>ZTS 21FBPY</td>
<td>1</td>
</tr>
<tr>
<td>Bv8LNiju69EVwn IvWmIMXi8kA</td>
<td>1</td>
</tr>
<tr>
<td>IxWGTyLRRnxw0eoa</td>
<td>1</td>
</tr>
<tr>
<td>HqFUt7jVofXPeXRecZm6SEBC5I</td>
<td>1</td>
</tr>
<tr>
<td>KaMBiTFvP3H 1ceEL siO</td>
<td>1</td>
</tr>
<tr>
<td>aAhX5Tv4dFdnqWe</td>
<td>1</td>
</tr>
<tr>
<td>g iQt1dOs9oms</td>
<td>1</td>
</tr>
<tr>
<td>kANpWUmA oBwxDkgxK6Z</td>
<td>1</td>
</tr>
<tr>
<td>NSLy1Qphj24</td>
<td>1</td>
</tr>
<tr>
<td>YhBv3Ff4v</td>
<td>1</td>
</tr>
<tr>
<td>1000 rows selected.</td>
<td></td>
</tr>
</tbody>
</table>

```sql
SQL> select endpoint_number, endpoint_actual_value from user_histograms
where table_name = 'PRODUCT'
and column_name = 'PRODUCT_NAME'
order by 1;
```

<table>
<thead>
<tr>
<th>ENDPOINT_NUMBER</th>
<th>ENDPOINT_ACTUAL_VALUE</th>
</tr>
</thead>
<tbody>
<tr>
<td>53</td>
<td>AT4rC1TYNy2W</td>
</tr>
<tr>
<td>54</td>
<td>BCMiI0s_B=66cm8n</td>
</tr>
<tr>
<td>55</td>
<td>B0 tYRzi XGpzb8tqsXkSgO_b</td>
</tr>
<tr>
<td>56</td>
<td>BSY86KtAE3l23FUsHg3c</td>
</tr>
<tr>
<td>131128</td>
<td>Bv8LNiju69EVwn IvWmIMXi8kA</td>
</tr>
<tr>
<td>131129</td>
<td>BvEznEeAHs_voIji</td>
</tr>
<tr>
<td>131130</td>
<td>BwMT14KSS1COlTL1Aw92o5 dfPhU9z</td>
</tr>
<tr>
<td>131131</td>
<td>CeT v8Wy JA3EywufTwVbId8P Ta1RhEw</td>
</tr>
</tbody>
</table>

1000 rows selected.
Hybrid Histograms

Mix between Frequency & Height Balanced

```
SQL> select table_name, column_name, num_distinct, num_nulls, num_buckets, density, sample_size, histogram
2  from user_tab_columns
3  where histogram != 'NONE'
4  order by table_name;
```

<table>
<thead>
<tr>
<th>TABLE_NAME</th>
<th>COLUMN_NAME</th>
<th>NUM_DISTINCT</th>
<th>NUM_NULLS</th>
<th>NUM_BUCKETS</th>
<th>DENSITY</th>
<th>SAMPLE_SIZE</th>
<th>HISTOGRAM</th>
</tr>
</thead>
<tbody>
<tr>
<td>CUSTOMERS</td>
<td>CUST_FIRST_NAME</td>
<td>5338</td>
<td>0</td>
<td>254</td>
<td>.000004</td>
<td>5435</td>
<td>HYBRID</td>
</tr>
<tr>
<td>CUSTOMERS</td>
<td>CUST_LAST_NAME</td>
<td>89248</td>
<td>0</td>
<td>254</td>
<td>0</td>
<td>5435</td>
<td>HYBRID</td>
</tr>
<tr>
<td>INVENTORIES</td>
<td>WAREHOUSE_ID</td>
<td>999</td>
<td>0</td>
<td>254</td>
<td>.000614</td>
<td>5463</td>
<td>HYBRID</td>
</tr>
</tbody>
</table>

```
SQL> SELECT ENDPOINT_NUMBER, ENDPOINT_ACTUAL_VALUE, ENDPOINT_REPEAT_COUNT
2  FROM USER_HISTOGRAMS
3  WHERE TABLE_NAME='CUSTOMERS'
4  AND COLUMN_NAME='CUST_LAST_NAME'
5  ORDER BY 1;
```

<table>
<thead>
<tr>
<th>ENDPOINT_NUMBER</th>
<th>ENDPOINT_ACTUAL_VALUE</th>
<th>ENDPOINT_REPEAT_COUNT</th>
</tr>
</thead>
<tbody>
<tr>
<td>14</td>
<td>Adjani</td>
<td>14</td>
</tr>
<tr>
<td>42</td>
<td>Andrews</td>
<td>9</td>
</tr>
<tr>
<td>70</td>
<td>Baldwin</td>
<td>11</td>
</tr>
<tr>
<td>95</td>
<td>Bates</td>
<td>17</td>
</tr>
<tr>
<td>118</td>
<td>Belushi</td>
<td>14</td>
</tr>
<tr>
<td>147</td>
<td>Boyer</td>
<td>13</td>
</tr>
<tr>
<td>177</td>
<td>Brando</td>
<td>14</td>
</tr>
<tr>
<td>202</td>
<td>Brown</td>
<td>11</td>
</tr>
<tr>
<td>3748</td>
<td>magno</td>
<td>1</td>
</tr>
<tr>
<td>3769</td>
<td>mandala</td>
<td>1</td>
</tr>
<tr>
<td>3790</td>
<td>marius</td>
<td>1</td>
</tr>
<tr>
<td>5433</td>
<td>zweier</td>
<td>1</td>
</tr>
<tr>
<td>5435</td>
<td>zywiec</td>
<td>1</td>
</tr>
</tbody>
</table>

254 rows selected.

Tries to evenly distribute buckets, some values can cross buckets. HYBRID uses ENDPOINT_REPEAT_COUNT to show Data skew & get more specific stats.
SQL Plan Management

• 11g feature helps stop performance regression via plan changes
  • Uses baselines to guarantee only better plans are used

• 12c SPM/Baseline Changes
  • Plan evolution now allows for automatic plan evolution
  • SPM evolve advisor is an Auto Task (SYS_AUTO_SPM_EVOLVE_TASK)
    • Runs nightly in maintenance window
    • Automatically runs the evolve process for non-accepted plans in SPM
    • DBA views results of nightly task using DBMS_SPM.REPORT_AUTO_EVOLVE_TASK
    • Can Manage via OEM or DBMS_AUTO_TASK_ADMIN

• Still can manually evolve an unaccepted plan using OEM or DBMS_SPM
  • DBMS_SPM.EVOLVE_SQL_PLAN_BASELINE has been deprecated

```sql
var task varchar2(1000);
var evolve varchar2(100);
var imple varchar2(1000);
var rpt clob;
exec :task := DBMS_SPM.CREATE_EVOLVE_TASK(sql_handle=>'&sql_handle');
exec :evolve := DBMS_SPM.EXECUTE_EVOLVE_TASK(task_name=>:task);
exec :imple := DBMS_SPM.IMPLEMENT_EVOLVE_TASK(task_name=>:task, FORCE=>true);
exec :rpt := DBMS_SPM.REPORT_EVOLVE_TASK(task_name=>:task, type=>'TEXT', execution_name=>:evolve);
print
```


## SQL Plan Management

### General Information Section

**Task Information:**
- **Task Name:** TASK_231
- **Task Owner:** SOE
- **Execution Name:** EXEC_241
- **Execution Type:** SPM EVOLVE
- **Scope:** COMPREHENSIVE
- **Status:** COMPLETED
- **Started:** 01/16/2014 15:36:22
- **Finished:** 01/16/2014 15:36:22
- **Last Updated:** 01/16/2014 15:36:22
- **Global Time Limit:** 2147483646
- **Per-Plan Time Limit:** UNUSED
- **Number of Errors:** 0

### Execution Statistics:

<table>
<thead>
<tr>
<th></th>
<th>Base Plan</th>
<th>Test Plan</th>
</tr>
</thead>
<tbody>
<tr>
<td>Elapsed Time (s)</td>
<td>0.001386</td>
<td>0.001356</td>
</tr>
<tr>
<td>CPU Time (s)</td>
<td>0.001387</td>
<td>0.001387</td>
</tr>
<tr>
<td>Buffer Gets</td>
<td>286</td>
<td>286</td>
</tr>
<tr>
<td>Optimizer Cost</td>
<td>5306</td>
<td>5306</td>
</tr>
<tr>
<td>Disk Reads</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>Direct Writes</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>Rows Processed</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>Executions</td>
<td>10</td>
<td>10</td>
</tr>
</tbody>
</table>

### Findings Section

**Findings (1):**
1. The plan was verified in 0.29600 seconds. It failed the benefit criterion because its verified performance was 1.00000 times worse than that of the baseline plan.

### Details Section

**Object ID:** 2
**Test Plan Name:** SQL_PLAN_9f0vf3un13nc19ac244c
**Base Plan Name:** SQL_PLAN_9f0vf3un13nc618516c2
**SQL Handle:** SQL_97036e1e811e28c
**Parsing Schema:** SOE
**Test Plan Creator:** SOE
**SQL Text:**
```
SELECT PRODUCTS.PRODUCT_ID, PRODUCT_NAME,
PRODUCT_DESCRIPTION, CATEGORY_ID, WEIGHT_CLASS,
WARRANTY_PERIOD, SUPPLIER_ID, PRODUCT_STATUS, LIST_PRICE,
MIN_PRICE, CATALOG_URL, QUANTITY_ON_HAND FROM PRODUCTS,
INVENTORIES WHERE PRODUCTS.PRODUCT_ID = :B2 AND
INVENTORIES.PRODUCT_ID = PRODUCTS.PRODUCT_ID AND ROWNUM < :B1
```

**Bind Variables:**
- 1 - (NUMBER): 266
- 2 - (NUMBER): 15
Does Adaptation Work?

**Average Wait Time per Execution for SQL Statement Product_Inventories | SPEED12_OWNER-PC**

January 2, 2014

- **CPU**
- **resmgr: cpu quantum**
- **db file sequential read**
- **read by other session**
- **direct path read temp**
- **simulator: hash latch**
- **buffer busy waits**
- **simulator: lru latch**
- **resmgr: internal state change**
- **sort extent pool**
- **local write wait**
- **Disk file operations I/O**
- **latch: cache buffers chains**
- **cursor: pin S**
- **latch: shared pool**
- **log buffer space**
- **latch: enqueue hash chains**
- **row cache lock**
- **cursor: pin S wait on X**
- **latch: row cache objects**
- **library cache load lock**
- **latch: in memory undo latch**
- **dbms ipc reply**

**Top Plans for SQL Statement Product_Inventories | SPEED12_OWNER-PC**

January 2, 2014

(only active time for cached plans)

- 1259487565
- 3789654942
- 1138932093
- 2334414977
- 1437210771
- 2768157693

Minutes

- 140
- 120
- 100
- 80
- 60
- 40
- 20
- 0

12AM 2AM 4AM 6AM 8AM 10AM 12PM 2PM 4PM 6PM 8PM 10PM
Adaptive Query Optimizer can improve plans over time

• **Adaptative plans**
  - Adapt on first execution
  - Need to view via DBMS_XPLAN to see what’s changed
  - If in ‘report_only’ mode, don’t trust explain plan

• **Adaptive statistics augments base statistics after initial execution**
  - Dynamic Statistics (level=11)
  - Automatic Reoptimization
    - SQL_PLAN_DIRECTIVES stored so other sqls can use

• **New histograms help ‘almost popular’ data skews**
  - Top Frequency & Hybrid
  - DBMS_STATS, use the defaults (AUTO_SAMPLE_SIZE)

• **SPM - Baselines now automatically evolve**
  - Nightly Process
  - EVOLVE_SQL_PLAN_BASELINE has been deprecated but still there
About Confio

• Wait-Based Performance Tools
• DPA (formally Ignite)  
  — for Oracle, SQL Server, Sybase, DB2
• Helps show which SQL to tune
• Based in Colorado, worldwide customers
• Free trial at www.confio.com
• http://www.ignitefree.com – Free Current View
APPENDIX
declare c number;
col_cnt number;
col_rec dbms_sql.desc_tab;
col_value varchar2(4000);
ret_val number;

Begin

c := dbms_sql.open_cursor;
dbms_sql.parse(c,'select q.sql_text, s.* from v$sql_shared_cursor s, v$sql q where s.sql_id = q.sql_id and s.child_number = q.child_number and q.sql_id = "&1"', dbms_sql.native);
dbms_sql.describe_columns(c, col_cnt, col_rec);
for idx in 1 .. Col_cnt loop
    dbms_sql.define_column(c, idx, col_value, 4000);
end loop;
ret_val := dbms_sql.execute(c);
while(dbms_sql.fetch_rows(c) > 0) loop
    for idx in 1 .. Col_cnt loop
        dbms_sql.column_value(c, idx, col_value);
        if col_rec(idx).col_name in ('SQL_ID', 'ADDRESS', 'CHILD_ADDRESS', 'CHILD_NUMBER', 'SQL_TEXT', 'REASON') then
            dbms_output.put_line(rpad(col_rec(idx).col_name, 30) || '=' || col_value);
        elsif col_value = 'Y' then
            dbms_output.put_line(rpad(col_rec(idx).col_name, 30) || '=' || col_value);
        end if;
    end loop;
    dbms_output.put_line('--------------------------------------------------');
end loop;
dbms_sql.close_cursor(c);
End;
/