Fine Tune Oracle Execution Plans for Performance Gains

Janis Griffin
Senior DBA
Who Am I?

- Senior DBA for Confio Software
  - JanisGriffin@confio.com
  - @DoBoutAnything

- Current - 24+ 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?
Agenda

- What are Execution Plans
- How to View Them
- Interpret Plan Details / Tips / Techniques
- Tune – Case Study
- Additional Tools
What are Execution Plans

Execution plans provide the **sequence of operations** performed in order to run SQL Statements. They show:

- Order of the tables referenced in the statement
- Access method for each table in the statement
  - INDEX
  - INLIST ITERATOR
  - TABLE ACCESS
  - VIEW
- Join method in statement accessing multiple tables
  - HASH JOIN
  - MERGE JOIN
  - NESTED LOOPS
- Data manipulations
  - CONCATENATION
  - COUNT
  - FILTER
  - SORT
Execution Plan Information

- **Operation** - Name of the internal action performed in each step.
  - First Row is always:
    - DELETE STATEMENT
    - INSERT STATEMENT
    - SELECT STATEMENT
    - UPDATE STATEMENT

- **Options**
  - Variation of Operation

- **Object_name**
  - Table or Index

- **Parent_id**
  - Related Operations

- **More Information**
  - Not listed here

---

**Operation List:**
- AND-EQUAL
- CONNECT BY
- CONCATENATION
- COUNT
- DOMAIN INDEX
- FILTER
- FIRST ROW
- FOR UPDATE
- HASH JOIN
- INDEX
- ITERATOR INTERSECTION
- MERGE JOIN
- MINUS
- NESTED LOOPS
- PARTITION
- REMOTE
- SEQUENCE
- SORT
- TABLE ACCESS
- UNION
- VIEW

**Option Examples:**
- AGGREGATE
- ALL
- BY INDEX ROWID
- BY LOCAL INDEX ROWID
- BY USER ROWID
- CARTESIAN
- FAST FULL SCAN
- FULL
- FULL OUTER
- FULL SCAN
- GROUP BY
- ITERATOR
- ORDER BY
- OUTER
- PARTITION
- RANGE SCAN
- SAMPLE FAST FULL SCAN
- SKIP SCAN
- UNIQUE
Optimizer’s detailed steps to execute a SQL Statement

```
SELECT e.empno EID, e.ename "Employee_name",
      d.dname "Department", e.hiredate "Hired_Date"
FROM  emp e,  dept d
WHERE  d.deptno = '40'
AND e.deptno = d.deptno;
```

---

<table>
<thead>
<tr>
<th>Id</th>
<th>Operation &amp; Option</th>
<th>Name</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>SELECT STATEMENT</td>
<td></td>
</tr>
<tr>
<td>1</td>
<td>NESTED LOOPS</td>
<td></td>
</tr>
<tr>
<td>2</td>
<td>TABLE ACCESS BY INDEX ROWID</td>
<td>DEPT</td>
</tr>
<tr>
<td>3</td>
<td>INDEX UNIQUE SCAN</td>
<td>PK_DEPT</td>
</tr>
<tr>
<td>4</td>
<td>TABLE ACCESS FULL</td>
<td>EMP</td>
</tr>
</tbody>
</table>

---
How To View Plans

- **EXPLAIN PLAN**
  - Estimated plan - can be wrong for many reasons
    - Explain Plan For ... *sql statement*
    - Set autotrace *(on | trace | exp | stat | off)*

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

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

- **Historical Plans – AWR, Confio Ignite**
  - Shows plan changes over time

### DBMS_XPLAN - Table Functions to Format & Display Plans
- `DISPLAY` - contents of a plan table.
- `DISPLAY_AWR` - the plan of a stored SQL statement in AWR.
- `DISPLAY_CURSOR` - the execution plan of any loaded cursor.
- `DISPLAY_SQL_PLAN_BASELINE` - 1+ plans for a SQL statement
- `DISPLAY_SQLSET` - multi-plans of statements in SQL tuning set
## New Functions 11g

<table>
<thead>
<tr>
<th>Function</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>BUILD_PLAN_XML</td>
<td>Return the last plan, or a named plan, explained as XML</td>
</tr>
<tr>
<td>DISPLAY</td>
<td>Shows the last plan explained – EXPLAIN PLAN ** Only FUNCTION in Oracle 9i</td>
</tr>
<tr>
<td>DISPLAY_AWR</td>
<td>Format &amp; display the plan of a stored SQL statement in AWR</td>
</tr>
<tr>
<td>DISPLAY_CURSOR</td>
<td>Format &amp; display the execution plan of any loaded cursor</td>
</tr>
<tr>
<td>DISPLAY_PLAN</td>
<td>Return the last plan, or a named plan, explained as a CLOB</td>
</tr>
<tr>
<td>DISPLAY_SQLSET</td>
<td>Format &amp; display the execution plan of statements stored in a SQL tuning set</td>
</tr>
<tr>
<td>DISPLAY_SQL_PLAN_BASELINE</td>
<td>Displays one or more plans for the specified SQL statement</td>
</tr>
<tr>
<td>FORMAT_NUMBER</td>
<td>Returns a number as a string</td>
</tr>
<tr>
<td>FORMAT_NUMBER2</td>
<td>Returns a number as a string formatted with a leading space (CHR(32))</td>
</tr>
<tr>
<td>FORMAT_SIZE</td>
<td>Undocumented</td>
</tr>
<tr>
<td>FORMAT_SIZE2</td>
<td>Undocumented</td>
</tr>
<tr>
<td>FORMAT_TIME_S</td>
<td>Undocumented</td>
</tr>
<tr>
<td>PREPARE_PLAN_XML_QUERY</td>
<td>- function to build the XML version of a select query that is run before the display function to retrieve and display the execution plan of a SQL</td>
</tr>
<tr>
<td>PREPARE_RECORDS</td>
<td>Used Internally</td>
</tr>
<tr>
<td>VALIDATE_FORMAT</td>
<td>Used Internally</td>
</tr>
</tbody>
</table>
SELECT e.empno EID, e.ename "Employee_name", d.dname "Department", e.hiredate "Date_Hired"
FROM emp e, dept d WHERE d.deptno = :P1 AND e.deptno = d.deptno;

SET AUTOTRACE TRACEONLY:

Execution Plan

Plan hash value: 568005898

<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>1503</td>
<td>54108</td>
<td>15</td>
<td>00:00:01</td>
</tr>
<tr>
<td>1</td>
<td>NESTED LOOPS</td>
<td></td>
<td>1503</td>
<td>54108</td>
<td>15</td>
<td>00:00:01</td>
</tr>
<tr>
<td>2</td>
<td>TABLE ACCESS BY INDEX ROWID</td>
<td>DEPT</td>
<td>1</td>
<td>11</td>
<td>2</td>
<td>00:00:01</td>
</tr>
<tr>
<td>*3</td>
<td>INDEX UNIQUE SCAN</td>
<td>PK_DEPT</td>
<td>1</td>
<td></td>
<td>1</td>
<td>00:00:01</td>
</tr>
<tr>
<td>*4</td>
<td>TABLE ACCESS FULL</td>
<td>EMP</td>
<td>1503</td>
<td>37575</td>
<td>13</td>
<td>00:00:01</td>
</tr>
</tbody>
</table>

Predicate Information (identified by operation id):

3 - access("D"."DEPTNO"=TO_NUMBER(:P1))
4 - filter("E"."DEPTNO"=TO_NUMBER(:P1))

Statistics

0 recursive calls
0 db block gets
312 consistent gets
0 physical reads
0 redo size
124547 bytes sent via SQL*Net to client
3413 bytes received via SQL*Net from client
265 SQL*Net roundtrips to/from client
0 sorts (memory)
0 sorts (disk)
3958 rows processed
- Understand objects in execution plans.
  - Table & Segment sizes
  - Number of Rows
  - Indexes & their column order
  - Column data types
  - Cardinality of columns / Data Skew
  - Statistic Gathering
  - Histograms?

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

- Run it for expensive data access targets
### Table & Column Statistics

```sql
SELECT count(*) FROM EMP;
```

```
COUNT(*)
---------
6013
``` 

```sql
SELECT 6013/4 dist FROM DUAL;
```

```
DIST
------
1503
11
``` 

```sql
SELECT column_name, num_distinct, num_nulls, num_buckets, density, sample_size
FROM user_tab_columns
WHERE table_name = 'EMP'
ORDER BY column_name;
```

<table>
<thead>
<tr>
<th>COLUMN_NAME</th>
<th>NUM_DISTINCT</th>
<th>NUM_NULLS</th>
<th>NUM_BUCKETS</th>
<th>DENSITY</th>
<th>SAMPLE_SIZE</th>
</tr>
</thead>
<tbody>
<tr>
<td>COMM</td>
<td>1534</td>
<td>4430</td>
<td>1</td>
<td>.00065189</td>
<td>1583</td>
</tr>
<tr>
<td>DEPTNO</td>
<td>4</td>
<td>0</td>
<td>4</td>
<td>.000003153</td>
<td>6013</td>
</tr>
<tr>
<td>EMPNO</td>
<td>6013</td>
<td>0</td>
<td>1</td>
<td>.000166306</td>
<td>6013</td>
</tr>
<tr>
<td>ENAME</td>
<td>6013</td>
<td>0</td>
<td>254</td>
<td>.000166306</td>
<td>6013</td>
</tr>
<tr>
<td>HIREDATE</td>
<td>88</td>
<td>0</td>
<td>1</td>
<td>.011363636</td>
<td>6013</td>
</tr>
<tr>
<td>JOB</td>
<td>22</td>
<td>0</td>
<td>22</td>
<td>.000003153</td>
<td>6013</td>
</tr>
<tr>
<td>MGR</td>
<td>6</td>
<td>6000</td>
<td>1</td>
<td>.166666667</td>
<td>13</td>
</tr>
<tr>
<td>SAL</td>
<td>6000</td>
<td>0</td>
<td>1</td>
<td>.000166667</td>
<td>6013</td>
</tr>
</tbody>
</table>

```sql
SELECT DEPTNO, count(*) FROM EMP
GROUP BY DEPTNO;
```

<table>
<thead>
<tr>
<th>DEPTNO</th>
<th>COUNT(*)</th>
</tr>
</thead>
<tbody>
<tr>
<td>10</td>
<td>77</td>
</tr>
<tr>
<td>20</td>
<td>1500</td>
</tr>
<tr>
<td>30</td>
<td>478</td>
</tr>
<tr>
<td>40</td>
<td>3958</td>
</tr>
</tbody>
</table>

Would an index on EMP.DEPTNO increase performance?
exec dbms_stats.gather_schema_stats( ownname => 'SCOTT', 
options => 'GATHER AUTO', estimate_percent => dbms_stats.auto_sample_size, 
method_opt => 'for all columns size auto' ... 

<table>
<thead>
<tr>
<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>COMM</td>
<td>1534</td>
<td>4430</td>
<td>1</td>
<td>0.00065189</td>
<td>1583</td>
<td>NONE</td>
</tr>
<tr>
<td>DEPTNO</td>
<td>4</td>
<td>0</td>
<td>4</td>
<td>0.00083153</td>
<td>6013</td>
<td>FREQUENCY</td>
</tr>
<tr>
<td>EMPNO</td>
<td>6013</td>
<td>0</td>
<td>1</td>
<td>0.00166306</td>
<td>6013</td>
<td>NONE</td>
</tr>
<tr>
<td>ENAME</td>
<td>6013</td>
<td>0</td>
<td>254</td>
<td>0.00016630</td>
<td>6013</td>
<td>HEIGHT BALANCED</td>
</tr>
<tr>
<td>HIREDATE</td>
<td>88</td>
<td>0</td>
<td>1</td>
<td>0.01136363</td>
<td>6013</td>
<td>NONE</td>
</tr>
<tr>
<td>JOB</td>
<td>22</td>
<td>0</td>
<td>22</td>
<td>0.00083153</td>
<td>6013</td>
<td>FREQUENCY</td>
</tr>
<tr>
<td>MGR</td>
<td>6</td>
<td>6000</td>
<td>1</td>
<td>1.16666667</td>
<td>6013</td>
<td>NONE</td>
</tr>
<tr>
<td>SAL</td>
<td>6000</td>
<td>0</td>
<td>1</td>
<td>0.0016667</td>
<td>6013</td>
<td>NONE</td>
</tr>
</tbody>
</table>

exec dbms_stats.gather_table_stats( ownname => 'SCOTT', 
tabname => 'EMP', method_opt=>'FOR COLUMNS deptno SIZE 2');
Extra Info - 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 = '0zz5h1003f2dw';
```

NAME      POSITION  DATATYPE_STRING       VALUE_STRING
---------- ----------- -------------------------- ------------
:P1        1           VARCHAR2(128)      40

**EMP Columns:**

<table>
<thead>
<tr>
<th>Name</th>
<th>Type</th>
</tr>
</thead>
<tbody>
<tr>
<td>EMPNO</td>
<td>NUMBER(4)</td>
</tr>
<tr>
<td>ENAME</td>
<td>VARCHAR2(10)</td>
</tr>
<tr>
<td>JOB</td>
<td>VARCHAR2(9)</td>
</tr>
<tr>
<td>MGR</td>
<td>NUMBER(4)</td>
</tr>
<tr>
<td>HIREDATE</td>
<td>DATE</td>
</tr>
<tr>
<td>SAL</td>
<td>NUMBER(7,2)</td>
</tr>
<tr>
<td>COMM</td>
<td>NUMBER(7,2)</td>
</tr>
<tr>
<td>DEPTNO</td>
<td>NUMBER(2)</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
Execution Plan Details

SELECT e.empno EID, e.ename "Employee_name", d.dname "Department", e.hiredate "Date_Hired"
FROM emp e, dept d
WHERE d.deptno = :P1 AND e.deptno = d.deptno;

Actual Plan: V$SQL_PLAN using dbms_xplan.display_cursor

---

<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>15 (100)</td>
<td></td>
</tr>
<tr>
<td>1</td>
<td>NESTED LOOPS</td>
<td></td>
<td></td>
<td></td>
<td>15 (0)</td>
<td>00:00:01</td>
</tr>
<tr>
<td>2</td>
<td>TABLE ACCESS BY INDEX ROWID</td>
<td>DEPT</td>
<td>1</td>
<td>11</td>
<td>2 (0)</td>
<td>00:00:01</td>
</tr>
<tr>
<td>* 3</td>
<td>INDEX UNIQUE SCAN</td>
<td>PK_DEPT</td>
<td>1</td>
<td></td>
<td>1 (0)</td>
<td>00:00:01</td>
</tr>
<tr>
<td>* 4</td>
<td>TABLE ACCESS FULL</td>
<td>EMP</td>
<td>3958</td>
<td>98950</td>
<td>13 (0)</td>
<td>00:00:01</td>
</tr>
</tbody>
</table>

Predicate Information (identified by operation id):

<table>
<thead>
<tr>
<th>Id</th>
<th>Expression</th>
</tr>
</thead>
<tbody>
<tr>
<td>3</td>
<td>access(&quot;D&quot;.&quot;DEPTNO&quot;=TO_NUMBER(:P1))</td>
</tr>
<tr>
<td>4</td>
<td>filter(&quot;E&quot;.&quot;DEPTNO&quot;=TO_NUMBER(:P1))</td>
</tr>
</tbody>
</table>
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

```sql
SELECT * from table(dbms_xplan.display_cursor('az7r9s3wpqq7n',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>*</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):

```
l - filter(TO_BINARY_DOUBLE("SEGMENT")=:B1)
```
With Oracle optimizer, execution plans can change as the underlying inputs to the optimizer change.

- **Same Sql – Different Schemas**
  - Same schema changes i.e. adding / dropping indexes

- **Same Sql – Different Costs**
  - Data volume & Statistic Changes over time
  - Bind variable types and values
  - Initialization parameters (set globally or session level)

- **V$SQL_SHARED_CURSOR**
  - Can give clues to why plan changed
  - Approximately 60 columns showing mismatches /differences
Tuning List – How to

- Find the Expensive Operators
  - Examine Costs / Row Count / Time
  - Table Access Full?

- Review Filter and Access Predicates
  - Shows how query is interpreted, e.g. bind variables
  - Can help Diagnose Data Type Issues

- Evaluate Object Stats
  - Table Definitions
  - Sizes and Row Counts

- Determine Existing Indexes
  - Index Definitions
  - Index Selectivity

- Evaluate Column Stats
  - Limiting Factors from WHERE Clause

- Review Join Columns – Are they indexed / data skew?
Example SQL Statement

- Find inventory of products in a specific category at a particular location?

```
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 INVENTORIES.PRODUCT_ID = PRODUCTS.PRODUCT_ID
    AND PRODUCTS.CATEGORY_ID = :B3
    AND INVENTORIES.WAREHOUSE_ID = :B2
    AND ROWNUM < :B1 ;
```

- Average Execution Time – 10.81 seconds
- Wait Event – Waits 90% on direct path read
Why this SQL Statement?

**Top SQL Statements | CECE_JGRIMFIN-2 | March 12, 2011 - 4:00PM to 5:00PM**

- **Product_Inventories**
  - 182461657
  - 3224184847
  - 3813131513
  - 1349476360
  - 1701964146
  - 832045657

**March 12 4:00PM-5:00PM**

<table>
<thead>
<tr>
<th>SQL Name</th>
<th>Wait Time</th>
<th>Total Wait Time for Time Period</th>
<th>% of Total Wait Time</th>
<th>Average (seconds)</th>
<th>Executions</th>
</tr>
</thead>
<tbody>
<tr>
<td>Product_Inventories</td>
<td>3:58:08</td>
<td>8:49:12 (hh:mm:ss)</td>
<td>45%</td>
<td>10.81604845</td>
<td>1,321</td>
</tr>
</tbody>
</table>
SELECT .... columns
FROM PRODUCTS,
    INVENTORIES
WHERE INVENTORIES.PRODUCT_ID = PRODUCTS.PRODUCT_ID
AND PRODUCTS.CATEGORY_ID = :B3
AND INVENTORIES.WAREHOUSE_ID = :B2
AND ROWNUM < :B1
Actual Plan – No Statistics

SELECT .... columns FROM PRODUCTS, INVENTORIES WHERE INVENTORIES.PRODUCT_ID = PRODUCTS.PRODUCT_ID AND PRODUCTS.CATEGORY_ID = :B3 AND INVENTORIES.WAREHOUSE_ID = :B2 AND ROWNUM < :B1
Understanding the Objects

PRODUCTS View

```
SELECT i.product_id, d.language_id, CASE WHEN d.language_id IS NOT NULL THEN d.translated_name ELSE TRANSLATE(i.product_name USING NCHAR_CS) END AS product_name, i.category_id, CASE WHEN d.language_id IS NOT NULL THEN d.translated_description ELSE TRANSLATE(i.product_description USING NCHAR_CS) END AS product_description, i.weight_class, i.warranty_period, i.supplier_id, i.product_status, i.list_price, i.min_price FROM product_information i, product_descriptions d WHERE d.product_id (++) = i.product_id AND d.language_id (++) = sys_context('USERENV', 'LANG')
```

No Statistics

<table>
<thead>
<tr>
<th>COLUMN_NAME</th>
<th>NUM_DISTINCT</th>
<th>NUM_NULLS</th>
<th>NUM_BUCKETS</th>
<th>DENSITY</th>
<th>SAMPLE_SIZE</th>
<th>HISTORAM</th>
</tr>
</thead>
<tbody>
<tr>
<td>PRODUCT_ID</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>NONE</td>
</tr>
<tr>
<td>QUANTITY_ON_HAND</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>NONE</td>
</tr>
<tr>
<td>WAREHOUSE_ID</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>NONE</td>
</tr>
</tbody>
</table>

DBMS_STATS

```
exec dbms_stats.gather_schema_stats( - 
    ownername => 'SOE', - 
    options => 'GATHER AUTO', - 
    estimate_percent => dbms_stats.auto_sample_size, - 
    method_opt => 'for all columns size auto', - 
    cascade => true, - 
    degree => 15 - 
)
```

Improved in 11g
Plan hash value: 3363672010

<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>COUNT STOPKEY</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>* 2</td>
<td>HASH JOIN OUTER</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>NESTED LOOPS</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>* 5</td>
<td>TABLE ACCESS FULL</td>
<td>INVENTORIES</td>
<td>896</td>
<td>11648</td>
<td>2988</td>
<td></td>
</tr>
<tr>
<td>* 6</td>
<td>INDEX RANGE SCAN</td>
<td>PROD_CATEGORY_IDX</td>
<td>1</td>
<td></td>
<td>0</td>
<td></td>
</tr>
<tr>
<td>* 7</td>
<td>TABLE ACCESS BY INDEX ROWNUM</td>
<td>PRODUCT_INFORMATION</td>
<td>1</td>
<td>1153</td>
<td>0</td>
<td></td>
</tr>
<tr>
<td>* 8</td>
<td>TABLE ACCESS FULL</td>
<td>PRODUCT_DESCRIPTIONS</td>
<td>10</td>
<td>170</td>
<td>18</td>
<td>00:00:01</td>
</tr>
</tbody>
</table>

Predicate Information (identified by operation id):

1 - filter<ROWNUM>:B1
2 - access"D"."PRODUCT_ID"="I"."PRODUCT_ID"
5 - filter<"INVENTORIES"."WAREHOUSE_ID"=:B2
6 - access"I"."CATEGORY_ID"=:B3
7 - filter<"INVENTORIES"."PRODUCT_ID"="I"."PRODUCT_ID"
8 - filter"D"."LANGUAGE_ID"=SYS_CONTEXT('USERENV','LANG')>

Statistics

- 15 recursive calls
- 0 db block gets
- 11010 consistent gets
- 0 physical reads
- 0 redo size
- 2935 bytes sent via SQL*Net to client
- 519 bytes received via SQL*Net from client
- 2 SQL*Net roundtrips to/from client
- 0 sorts (memory)
- 0 sorts (disk)
- 6 rows processed
SELECT .... columns
FROM PRODUCTS,
    INVENTORIES
WHERE INVENTORIES.PRODUCT_ID = PRODUCTS.PRODUCT_ID
AND PRODUCTS.CATEGORY_ID = :B3
AND INVENTORIES.WAREHOUSE_ID = :B2
AND ROWNUM < :B1;

V$SQL_BIND_CAPTURE

<table>
<thead>
<tr>
<th>NAME</th>
<th>POSITION</th>
<th>DATATYPE_STRING</th>
<th>VALUE_STRING</th>
</tr>
</thead>
<tbody>
<tr>
<td>:B3</td>
<td>1</td>
<td>NUMBER</td>
<td>136 - category_id</td>
</tr>
<tr>
<td>:B2</td>
<td>2</td>
<td>NUMBER</td>
<td>454 - warehouse_id</td>
</tr>
<tr>
<td>:B1</td>
<td>3</td>
<td>NUMBER</td>
<td>15 - rownum</td>
</tr>
</tbody>
</table>

INVENTORIES Table Filter (warehouse_id=454) -1000 records

<table>
<thead>
<tr>
<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>PRODUCT_ID</td>
<td>1000</td>
<td>0</td>
<td>1</td>
<td>.001</td>
<td>894861</td>
<td>NONE</td>
</tr>
<tr>
<td>QUANTITY_ON_HAND</td>
<td>894861</td>
<td>0</td>
<td>1</td>
<td>1.117E-06</td>
<td>894861</td>
<td>NONE</td>
</tr>
<tr>
<td>WAREHOUSE_ID</td>
<td>999</td>
<td>0</td>
<td>1</td>
<td>.001001001</td>
<td>894861</td>
<td>NONE</td>
</tr>
</tbody>
</table>

select round((1000 / 894861 * 100), 2) pct_of_inventory from dual;

PCT_OF_INVENTORY

----------------------
.11
How to tune?

```sql
SELECT .... columns
FROM PRODUCTS,
    INVENTORIES
WHERE INVENTORIES.PRODUCT_ID = PRODUCTS.PRODUCT_ID
AND PRODUCTS.CATEGORY_ID = :B3
AND INVENTORIES.WAREHOUSE_ID = :B2
AND ROWNUM < :B1;
```

**V$SQL_BIND_CAPTURE**

<table>
<thead>
<tr>
<th>NAME</th>
<th>POSITION</th>
<th>DATATYPE_STRING</th>
<th>VALUE_STRING</th>
</tr>
</thead>
<tbody>
<tr>
<td>:B3</td>
<td>1</td>
<td>NUMBER</td>
<td>136</td>
</tr>
</tbody>
</table>

**PRODUCT_INFORMATION Table**

<table>
<thead>
<tr>
<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>CATALOG_URL</td>
<td>1000</td>
<td>0</td>
<td>1</td>
<td>.001</td>
<td>1000</td>
<td>NONE</td>
</tr>
<tr>
<td>CATEGORY_ID</td>
<td>196</td>
<td>0</td>
<td>196</td>
<td>.00005</td>
<td>1000</td>
<td>FREQUENCY</td>
</tr>
<tr>
<td>LIST_PRICE</td>
<td>910</td>
<td>0</td>
<td>1</td>
<td>.001098901</td>
<td>1000</td>
<td>NONE</td>
</tr>
<tr>
<td>MIN_PRICE</td>
<td>916</td>
<td>0</td>
<td>1</td>
<td>.001091703</td>
<td>1000</td>
<td>NONE</td>
</tr>
<tr>
<td>PRODUCT_DESCRIPTION</td>
<td>1000</td>
<td>0</td>
<td>1</td>
<td>.001</td>
<td>1000</td>
<td>NONE</td>
</tr>
<tr>
<td>PRODUCT_ID</td>
<td>1000</td>
<td>0</td>
<td>1</td>
<td>.001</td>
<td>1000</td>
<td>NONE</td>
</tr>
</tbody>
</table>

**USER_TAB_HISTOGRAMS**

```sql
SELECT endpoint_value, endpoint_number, endpoint_number - nvl(prev_number,0) from
    (SELECT endpoint_value, endpoint_number, lag(endpoint_number,1) over
        (order by endpoint_number) prev_number
    )
FROM user_tab_histograms
WHERE table_name = 'PRODUCT_INFORMATION'
    and column_name = 'CATEGORY_ID'
ORDER BY endpoint_value
```

<table>
<thead>
<tr>
<th>ENDPOINT_VALUE</th>
<th>ENDPOINT_NUMBER</th>
<th>FREQUENCY</th>
</tr>
</thead>
<tbody>
<tr>
<td>134</td>
<td>676</td>
<td>4</td>
</tr>
<tr>
<td>135</td>
<td>683</td>
<td>7</td>
</tr>
<tr>
<td>136</td>
<td>689</td>
<td>6</td>
</tr>
<tr>
<td>137</td>
<td>693</td>
<td>4</td>
</tr>
<tr>
<td>138</td>
<td>697</td>
<td>4</td>
</tr>
<tr>
<td>139</td>
<td>699</td>
<td>2</td>
</tr>
<tr>
<td>140</td>
<td>706</td>
<td>7</td>
</tr>
</tbody>
</table>
CREATE INDEX inventories_ix1 ON inventories(warehouse_id);

Plan hash value: 750880835

<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>COUNT STOPKEY</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>2</td>
<td>HASH JOIN</td>
<td></td>
<td>11</td>
<td>2266</td>
<td>927</td>
<td>&lt;100</td>
</tr>
<tr>
<td>3</td>
<td>HASH JOIN OUTER</td>
<td></td>
<td>7</td>
<td>1351</td>
<td>27</td>
<td>&lt;4</td>
</tr>
<tr>
<td>4</td>
<td>TABLE ACCESS BY INDEX ROWID</td>
<td>PRODUCT_INFORMATION</td>
<td>7</td>
<td>1232</td>
<td>8</td>
<td>&lt;0</td>
</tr>
<tr>
<td>5</td>
<td>INDEX RANGE SCAN</td>
<td>PROD_CATEGORY_IDX</td>
<td>7</td>
<td></td>
<td>1</td>
<td>&lt;0</td>
</tr>
<tr>
<td>6</td>
<td>TABLE ACCESS FULL</td>
<td>PRODUCT_DESCRIPTIONS</td>
<td>10</td>
<td>170</td>
<td>18</td>
<td>&lt;0</td>
</tr>
<tr>
<td>7</td>
<td>TABLE ACCESS BY INDEX ROWID</td>
<td>INVENTORIES</td>
<td>896</td>
<td>11648</td>
<td>900</td>
<td>&lt;0</td>
</tr>
<tr>
<td>8</td>
<td>INDEX RANGE SCAN</td>
<td>INVENTORIES_ix1</td>
<td>896</td>
<td></td>
<td>4</td>
<td>&lt;0</td>
</tr>
</tbody>
</table>

Predicate Information (identified by operation id):

1 - filter<ROWNUM:<B1>
2 - access<"INVENTORIES"."PRODUCT_ID"="I"."PRODUCT_ID">
3 - access<"D"."PRODUCT_ID"="I"."PRODUCT_ID">
4 - access<"I"."CATEGORY_ID"=:B3>
5 - filter<"D"."LANGUAGE_ID"=SYS_CONTEXT('USERENV','LANG')>
8 - access<"INVENTORIES"."WAREHOUSE_ID"=:B2>

March 13 4:00AM-5:00AM

<table>
<thead>
<tr>
<th>SQL Name</th>
<th>Product_Inventories</th>
</tr>
</thead>
<tbody>
<tr>
<td>Wait Time</td>
<td>03:12 (mm:ss)</td>
</tr>
<tr>
<td>Total Wait Time</td>
<td>27:59 (mm:ss)</td>
</tr>
<tr>
<td>% of Total Wait Time</td>
<td>11%</td>
</tr>
<tr>
<td>Average (seconds)</td>
<td>0.01872988</td>
</tr>
</tbody>
</table>
| Executions        | 10,251              | 100% on CPU
Did performance improve?

CREATE INDEX inventories_ix1 ON inventories(warehouse_id);
Which is the better Plan?
EXPLAIN PLAN
SET STATEMENT_ID = 'inventory' FOR
    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 INVENTORIES.PRODUCT_ID = PRODUCTS.PRODUCT_ID
AND PRODUCTS.CATEGORY_ID = :B3
AND INVENTORIES.WAREHOUSE_ID = :B2
AND ROWNUM < :B1;

set pages 0 head off
set linesize 132
set long 1000000
col xplan format a100

spool inventory.html

SELECT dbms_xplan.display_plan(statement_id => 'inventory',type=>'HTML') AS XPLAN
FROM dual;

spool off;
Plan Hash Value: 1842583762

<table>
<thead>
<tr>
<th>Id</th>
<th>Operation</th>
<th>Name</th>
<th>Rows</th>
<th>Bytes</th>
<th>Cost</th>
<th>Time</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>SELECT STATEMENT</td>
<td></td>
<td>6001</td>
<td>1260210</td>
<td>1090</td>
<td>00:00:14</td>
</tr>
<tr>
<td>1</td>
<td>COUNT STOPKEY</td>
<td></td>
<td>6001</td>
<td>1260210</td>
<td>1090</td>
<td>00:00:14</td>
</tr>
<tr>
<td>2</td>
<td>HASH JOIN RIGHT OUTER</td>
<td></td>
<td>6001</td>
<td>1260210</td>
<td>1090</td>
<td>00:00:14</td>
</tr>
<tr>
<td>3</td>
<td>TABLE ACCESS FULL</td>
<td>PRODUCT_DESCRIPTIONS</td>
<td>10</td>
<td>180</td>
<td>18</td>
<td>00:00:01</td>
</tr>
<tr>
<td>4</td>
<td>HASH JOIN</td>
<td></td>
<td>6001</td>
<td>1152192</td>
<td>1071</td>
<td>00:00:13</td>
</tr>
<tr>
<td>5</td>
<td>TABLE ACCESS BY INDEX ROWID</td>
<td>INVENTORIES</td>
<td>896</td>
<td>12544</td>
<td>876</td>
<td>00:00:11</td>
</tr>
<tr>
<td>6</td>
<td>INDEX RANGE SCAN</td>
<td>INVENTORIES_IX1</td>
<td>896</td>
<td></td>
<td>4</td>
<td>00:00:01</td>
</tr>
<tr>
<td>7</td>
<td>TABLE ACCESS BY INDEX ROWID</td>
<td>PRODUCT_INFORMATION</td>
<td>6692</td>
<td>1191176</td>
<td>194</td>
<td>00:00:03</td>
</tr>
<tr>
<td>8</td>
<td>INDEX RANGE SCAN</td>
<td>PROD_CATEGORY_IX</td>
<td>6692</td>
<td></td>
<td>17</td>
<td>00:00:01</td>
</tr>
</tbody>
</table>

Predicate Information (identified by operation id):

- 1 - filter(ROWNUM<TO_NUMBER(:B1))
- 2 - access("D"."PRODUCT_ID"(+)="I"."PRODUCT_ID")
- 3 - filter("D"."LANGUAGE_ID"(+)=SYS_CONTEXT('USERENV','LANG'))
- 4 - access("INVENTORIES"."PRODUCT_ID"="I"."PRODUCT_ID")
- 6 - access("INVENTORIES"."WAREHOUSE_ID"=TO_NUMBER(:B2))
- 8 - access("I"."CATEGORY_ID"=TO_NUMBER(:B3))

View using Display Plan
‘Free’ Graphical Plans

http://www.epviewer.bplaced.net/downloads
March 13 4:00AM-5:00AM

SQL Name: Product_ Inventories
Wait Time: 03:12 (mm:ss)
Total Wait Time for Time Period: 27:59 (mm:ss)
% of Total Wait Time: 11%
Average (seconds): 0.01872988
Executions: 10,251

SELECT STATEMENT
Est. Costs = 924

COUNT STOPKEY
filter: ROWNUM < TO_NUMBER(:B1)

HASH JOIN
access: 'INVENTORIES'. 'PRODUCT_ID' = 'I'. 'PRODUCT_ID'
Est. Elapsed Time = 12s Est. Costs = 924
Est. Rows = 10 Est. Bytes = 2,080

HASH JOIN OUTER
access: 'D'. 'PRODUCT_ID' = 'I'. 'PRODUCT_ID'
Est. Elapsed Time = 1s Est. Costs = 23
Est. Rows = 7 Est. Bytes = 1,365

AND

PRODUCT_INFORMATION
TABLE ACCESS BY INDEX ROWID
Est. Elapsed Time = 1s Est. Costs = 4
Est. Rows = 7 Est. Bytes = 1,246

PRODUCT_DESCRIPTIONS
TABLE ACCESS FULL
filter: 'D'. 'LANGUAGE_ID' = 'SYS_CONTEXT('USERENV', 'LANG')
Est. Elapsed Time = 1s Est. Costs = 18
Est. Rows = 10 Est. Bytes = 170

INVENTORIES
TABLE ACCESS BY INDEX ROWID
Est. Elapsed Time = 11s Est. Costs = 900
Est. Rows = 896 Est. Bytes = 11,648

INDEX RANGE SCAN
access: 'INVENTORIES'. 'WAREHOUSE_ID' = TO_NUMBER(:B2)
Est. Elapsed Time = 1s Est. Costs = 4
Est. Rows = 896

INDEX RANGE SCAN
access: 'L'. 'CATEGORY_ID' = TO_NUMBER(:B3)
Est. Elapsed Time = 1s Est. Costs = 3
Est. Rows = 7
Execution Plans show the internal steps Oracle takes to run SQL statements
  • Reports how the data is accessed, manipulated and joined
  • Shows Expensive steps with Cost and Time spent.
  • Gives information on Bind Variables (predicate information)

Viewing actual plans are better then using EXPLAIN PLAN
  • V$SQL_PLAN using DBMS_XPLAN.display_cursor(&sql_id, 0)
  • Tracing / TKPROF

Gather additional data when tuning an execution plan
  • Table sizes, Index selectivity and column details
  • How Statistic Gathering is performed
  • Bind value from V$SQL_BIND_CAPTURE

Tune only the execution plans that make a difference
  • Monitor response time – Using Ignite
Tuning Aids

- DBMS_XPLAN
  - Table Functions
  - New Additions – 11g

- SQL Plan management (DBMS_SPM)
  - Free For Enterprise users

- Oracle - Note: requires Tuning/Diagnostic Packs
  - SQL Tuning Advisor
  - ADDM
### DBMS_XPLAN – Functions

#### New Functions 11g

<table>
<thead>
<tr>
<th>Function</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>BUILD_PLAN_XML</td>
<td>Return the last plan, or a named plan, explained as XML</td>
</tr>
<tr>
<td>DISPLAY</td>
<td>Shows the last plan explained – EXPLAIN PLAN ** Only FUNCTION in Oracle 9i**</td>
</tr>
<tr>
<td>DISPLAY_AWR</td>
<td>Format &amp; display the plan of a stored SQL statement in AWR</td>
</tr>
<tr>
<td>DISPLAY_CURSOR</td>
<td>Format &amp; display the execution plan of any loaded cursor</td>
</tr>
<tr>
<td>DISPLAY_PLAN</td>
<td>Return the last plan, or a named plan, explained as a CLOB</td>
</tr>
<tr>
<td>DISPLAY_SQLSET</td>
<td>Format &amp; display the execution plan of statements stored in a SQL tuning set</td>
</tr>
<tr>
<td>DISPLAY_SQL_PLAN_BASELINE</td>
<td>Displays one or more plans for the specified SQL statement</td>
</tr>
<tr>
<td>FORMAT_NUMBER</td>
<td>Returns a number as a string</td>
</tr>
<tr>
<td>FORMAT_NUMBER2</td>
<td>Returns a number as a string formatted with a leading space (CHR(32))</td>
</tr>
<tr>
<td>FORMAT_SIZE</td>
<td>Undocumented</td>
</tr>
<tr>
<td>FORMAT_SIZE2</td>
<td>Undocumented</td>
</tr>
<tr>
<td>FORMAT_TIME_S</td>
<td>Undocumented</td>
</tr>
<tr>
<td>PREPARE_PLAN_XML_QUERY</td>
<td>- function to build the XML version of a select query that is run before the display function to retrieve and display the execution plan of a SQL</td>
</tr>
<tr>
<td>PREPARE_RECORDS</td>
<td>Used Internally</td>
</tr>
<tr>
<td>VALIDATE_FORMAT</td>
<td>Used Internally</td>
</tr>
</tbody>
</table>
DBMS_XPLAN – Example

- EXPLAIN PLAN set statement_id = 'prI' for select ...
- SELECT dbms_xplan.build_plan_xml(statement_id => 'prI') AS XPLAN FROM dual;

In browser: file:///c:/users/jgriffin/explan/xml.xml

```xml
<plan>
  + <operation name="SELECT STATEMENT" id="0" depth="0" pos="925"></operation>
  + <operation name="COUNT" options="STOPKEY" id="1" depth="1" pos="1"></operation>
  + <operation name="HASH JOIN" id="2" depth="2" pos="1"></operation>
  + <operation name="HASH JOIN" options="OUTER" id="3" depth="3" pos="1"></operation>
  + <operation name="TABLE ACCESS" options="BY INDEX ROWID" id="4" depth="4" pos="1"></operation>
  + <operation name="INDEX" options="RANGE SCAN" id="5" depth="5" pos="1"></operation>
  + <operation name="TABLE ACCESS" options="FULL" id="6" depth="4" pos="2"></operation>
  + <operation name="TABLE ACCESS" options="BY INDEX ROWID" id="7" depth="3" pos="2"></operation>
  - <operation name="INDEX" options="RANGE SCAN" id="8" depth="4" pos="1">
      <object>INVENTORIES IX1</object>
      <card>896</card>
      <cost>4</cost>
      <io_cost>4</io_cost>
      <cpu_cost>207686</cpu_cost>
      <time>0:00:01</time>
      <project>"INVENTORIES" ROWID[ROWID,10]"</project>
      <predicates type="access">"INVENTORIES"."WAREHOUSE_ID"=TO_NUMBER(B2)"</predicates>
      <qblock>SEL$F5BB74E1</qblock>
  </operation>
</plan>
```
### SQL Plan Management – 11g

DBMS_SPM manages execution plans & ensures only known or verified plans are used

<table>
<thead>
<tr>
<th>Procedure</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>ALTER_SQL_PLAN_BASELINE</strong></td>
<td>Changes an attribute of a single plan or all plans associated with a SQL statement using the attribute name/value format</td>
</tr>
<tr>
<td><strong>CONFIGURE</strong></td>
<td>Set configuration options for the SQL Management Base (SMB) as well as the maintenance of SQL plan baselines</td>
</tr>
<tr>
<td><strong>CREATE_STGTAB_BASELINE</strong></td>
<td>Creates a staging table that will be used for the purpose of transporting SQL plan baselines from one system to another</td>
</tr>
<tr>
<td><strong>DROP_SQL_PLAN_BASELINE</strong></td>
<td>Drops a single plan, or all plans associated with a SQL statement</td>
</tr>
<tr>
<td><strong>EVOLVE_SQL_PLAN_BASELINE</strong></td>
<td>Evolves SQL plan baselines associated with one or more SQL statements</td>
</tr>
<tr>
<td><strong>LOAD_PLANS_FROM_CURSOR_CACHE</strong></td>
<td>Loads one or more plans present in the cursor cache for a SQL statement</td>
</tr>
<tr>
<td><strong>LOAD_PLANS_FROM_SQLSET</strong></td>
<td>Loads plans stored in a SQL tuning set (STS) into SQL plan baselines</td>
</tr>
<tr>
<td><strong>PACK_STGTAB_BASELINE</strong></td>
<td>Packs (exports) SQL plan baselines from SQL management base into a staging table</td>
</tr>
<tr>
<td><strong>UNPACK_STGTAB_BASELINE</strong></td>
<td>Unpacks (imports) SQL plan baselines from a staging table into SQL management base</td>
</tr>
</tbody>
</table>
### Without Statistics

<table>
<thead>
<tr>
<th>Select Type</th>
<th>Findings</th>
<th>Recommendations</th>
<th>Rationale</th>
</tr>
</thead>
<tbody>
<tr>
<td>Statistics</td>
<td>Table &quot;CSJ&quot;, &quot;CLASS&quot; was not analyzed.</td>
<td>Consider collecting optimizer statistics for this table.</td>
<td>The optimizer requires up-to-date statistics for the table in order to select a good execution plan.</td>
</tr>
<tr>
<td>Statistics</td>
<td>Table &quot;CSJ&quot;, &quot;REGISTRATION&quot; was not analyzed.</td>
<td>Consider collecting optimizer statistics for this table.</td>
<td>The optimizer requires up-to-date statistics for the table in order to select a good execution plan.</td>
</tr>
<tr>
<td>Index</td>
<td>The execution plan of this statement can be improved by creating one or more indices.</td>
<td>Consider running the Access Advisor to improve the physical schema design or creating the recommended index.</td>
<td>Creating the recommended indices significantly improves the execution plan of this statement. However, it might be preferable to run &quot;Access Advisor&quot; using a representative SQL workload as opposed to a single statement. This will allow to get comprehensive index recommendations which takes into account index maintenance overhead and additional space consumption.</td>
</tr>
</tbody>
</table>

### With Statistics

<table>
<thead>
<tr>
<th>Select Type</th>
<th>Findings</th>
<th>Recommendations</th>
<th>Rationale</th>
</tr>
</thead>
<tbody>
<tr>
<td>Index</td>
<td>The execution plan of this statement can be improved by creating one or more indices.</td>
<td>Consider running the Access Advisor to improve the physical schema design or creating the recommended index.</td>
<td>Creating the recommended indices significantly improves the execution plan of this statement. However, it might be preferable to run &quot;Access Advisor&quot; using a representative SQL workload as opposed to a single statement. This will allow to get comprehensive index recommendations which takes into account index maintenance overhead and additional space consumption.</td>
</tr>
</tbody>
</table>
column sql_id new_value sql_id
select sql_id from v$sql where hash_value = &hash_value;

DECLARE
  l_sql_tune_task_id VARCHAR2(100);
BEGIN
  l_sql_tune_task_id := DBMS_SQLTUNE.create_tuning_task (sql_id => '&sql_id',
                  scope => DBMS_SQLTUNE.scope_comprehensive,
                  time_limit => 60,
                  task_name => '&sql_id',
                  description => 'Tuning task for statement 19v5guvsgcd1v.');</n
  DBMS_OUTPUT.put_line('l_sql_tune_task_id: ' || l_sql_tune_task_id);
END;
/

EXEC DBMS_SQLTUNE.execute_tuning_task(task_name => '&sql_id');

SET LONG 10000;
SET PAGESIZE 1000
SET LINESIZE 200
SELECT DBMS_SQLTUNE.report_tuning_task('&sql_id') AS recommendations FROM dual;
SET PAGESIZE 24

exec DBMS_SQLTUNE.drop_tuning_task (task_name => '&sql_id');
## RECOMMENDATIONS

### GENERAL INFORMATION SECTION

<table>
<thead>
<tr>
<th>Tuning Task Name</th>
<th>cpn6dwxd743s1</th>
</tr>
</thead>
<tbody>
<tr>
<td>Tuning Task Owner</td>
<td>SYS</td>
</tr>
<tr>
<td>Workload Type</td>
<td>Single SQL Statement</td>
</tr>
<tr>
<td>Scope</td>
<td>COMPREHENSIVE</td>
</tr>
<tr>
<td>Time Limit (seconds)</td>
<td>60</td>
</tr>
<tr>
<td>Completion Status</td>
<td>COMPLETED</td>
</tr>
<tr>
<td>Started at</td>
<td>11/10/2010 16:29:44</td>
</tr>
<tr>
<td>Completed at</td>
<td>11/10/2010 16:29:44</td>
</tr>
</tbody>
</table>

### Schema Name: DEMO8

### SQL ID: cpn6dwxd743s1

### SQL Text:

```
select NAME, SINCENORMAL from CON_ALERT_DB_STATUS_HISTORY where ALERTID=:1 and DBID=:2
```

### FINDINGS SECTION (1 finding)

1. **Index Finding (see explain plans section below)**

   The execution plan of this statement can be improved by creating one or more indices.

   **Recommendation** *(estimated benefit: 56.4%)*

   - Consider running the Access Advisor to improve the physical schema design or creating the recommended index.

     ```sql
     create index DEMO8.IDX$$_11570001 on DEMO8.CON_ALERT_DB_STATUS_HISTORY("DBID","ALERTID");
     ```

   **Rationale**

   Creating the recommended indices significantly improves the execution plan of this statement. However, it might be preferable to run "Access Advisor" using a representative SQL workload as opposed to a single statement. This will allow to get comprehensive index recommendations which takes into account index maintenance overhead and additional space consumption.
Sql Tuning Advice - New index contains same columns but reversed.

### EXPLAIN PLANS SECTION

**1- Original**

Plan hash value: 642510383

<table>
<thead>
<tr>
<th>Id</th>
<th>Operation</th>
<th>Name</th>
<th>Rows</th>
<th>Byte</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>Cost</td>
<td>CPU</td>
<td>Time</td>
<td></td>
</tr>
<tr>
<td>0</td>
<td>39</td>
<td>0</td>
<td>00:00:01</td>
<td></td>
</tr>
<tr>
<td>1</td>
<td>TABLE ACCESS BY INDEX ROWID</td>
<td>CON_ALERT_DB_STATUS_HISTORY</td>
<td>36</td>
<td>1004</td>
</tr>
<tr>
<td>4</td>
<td>17</td>
<td>0</td>
<td>00:00:01</td>
<td></td>
</tr>
<tr>
<td>4</td>
<td>SELECT STATEMENT</td>
<td></td>
<td>36</td>
<td>1004</td>
</tr>
<tr>
<td>4</td>
<td>INDEX RANGE SCAN</td>
<td>SYS_C0013034</td>
<td>36</td>
<td></td>
</tr>
</tbody>
</table>

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

2 - access("ALERTID"=:1 AND "DBID"=:2)

**2- Using New Indices**

Plan hash value: 2465160697

<table>
<thead>
<tr>
<th>Id</th>
<th>Operation</th>
<th>Name</th>
<th>Rows</th>
<th>Byte</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>Cost</td>
<td>CPU</td>
<td>Time</td>
<td></td>
</tr>
<tr>
<td>0</td>
<td>17</td>
<td>0</td>
<td>00:00:01</td>
<td></td>
</tr>
<tr>
<td>4</td>
<td>17</td>
<td>0</td>
<td>00:00:01</td>
<td></td>
</tr>
<tr>
<td>0</td>
<td>SELECT STATEMENT</td>
<td></td>
<td>36</td>
<td>1004</td>
</tr>
<tr>
<td>4</td>
<td>TABLE ACCESS BY INDEX ROWID</td>
<td>CON_ALERT_DB_STATUS_HISTORY</td>
<td>36</td>
<td>1004</td>
</tr>
<tr>
<td>4</td>
<td>INDEX RANGE SCAN</td>
<td>IDX$$_11570001</td>
<td>36</td>
<td></td>
</tr>
</tbody>
</table>

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

2 - access("DBID"=:2 AND "ALERTID"=:1)
### With Statistics

<table>
<thead>
<tr>
<th>Select Type</th>
<th>Findings</th>
<th>Recommendations</th>
<th>Rationale</th>
</tr>
</thead>
<tbody>
<tr>
<td>Index</td>
<td>The execution plan of this statement can be improved by creating one or more indices.</td>
<td>Consider running the Access Advisor to improve the physical schema design or creating the recommended index. <code>&lt;INDEX_NAME&gt;</code></td>
<td>Creating the recommended indices significantly improves the execution plan of this statement. However, it might be preferable to run the Access Advisor using a representative SQL workload as opposed to a single statement. This will allow to get comprehensive index recommendations which takes into account index maintenance overhead and additional space consumption.</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Benefit (%)</th>
<th>New Explain Plan</th>
<th>Compare Explain Plans</th>
</tr>
</thead>
<tbody>
<tr>
<td>91.46</td>
<td>☺️</td>
<td>☺️</td>
</tr>
</tbody>
</table>
var tname VARCHAR2(60);

prompt Enter Start Date (mm/dd/yy hh24)
accept beg_time
prompt Enter End Date (mm/dd/yy hh24)
accept end_time

DECLARE
bsnap NUMBER;
esnap NUMBER;
BEGIN
select distinct BEGIN_SNAP_ID,END_SNAP_ID
into bsnaps,esnap
from WRI$_ADV_ADDM_TASKS
where BEGIN_TIME between to_date('&beg_time', 'mm/dd/yy hh24') and to_date('&end_time', 'mm/dd/yy hh24');

tname := 'ADDM: '||bsnap||'-'||esnap;
DBMS_ADDM.ANALYZE_DB(:tname, bsnaps, esnap);
END;
/

SET LONG 100000
SET PAGESIZE 50000
SELECT DBMS_ADDM.GET_REPORT(:tname) FROM DUAL;
Example of ADDM Output Analysis for current hour.

Specific findings and suggestions

---

**ADDM Report for Task 'ADDM: 2-3'**

**Analysis Period**

AWR snapshot range from 2 to 3.
Time period starts at 15-NOV-10 04.00.07 PM
Time period ends at 15-NOV-10 05.00.21 PM

**Analysis Target**

Database 'CECE' with DB ID 1100810298.
Database version 11.1.0.7.0.
Analysis was requested for all instances, but ADDM analyzed instance cece,
numbered 1 and hosted at JGRiffin-2.
See the "Additional Information" section for more information on the requested
instances.

**Activity During the Analysis Period**

Total database time was 1558 seconds.
The average number of active sessions was .43.
ADDM analyzed 1 of the requested 1 instances.

**Summary of Findings**

<table>
<thead>
<tr>
<th>Description</th>
<th>Active Sessions</th>
<th>Percent of Activity</th>
<th>Recommendations</th>
</tr>
</thead>
<tbody>
<tr>
<td>1 Top SQL by DB Time</td>
<td>.22</td>
<td>51</td>
<td>5</td>
</tr>
<tr>
<td>2 I/O Throughput</td>
<td>.17</td>
<td>39.91</td>
<td>7</td>
</tr>
<tr>
<td>3 Checkpoints Due to Log File Size</td>
<td>.09</td>
<td>21.24</td>
<td>1</td>
</tr>
<tr>
<td>4 Undo I/O</td>
<td>.09</td>
<td>20.99</td>
<td>8</td>
</tr>
<tr>
<td>5 Top SQL By I/O</td>
<td>.09</td>
<td>20.22</td>
<td>4</td>
</tr>
<tr>
<td>6 Row Lock Waits</td>
<td>.06</td>
<td>12.8</td>
<td>1</td>
</tr>
<tr>
<td>7 Log File Switches</td>
<td>.05</td>
<td>11.29</td>
<td>2</td>
</tr>
<tr>
<td>8 Top Segments by I/O</td>
<td>.04</td>
<td>9.3</td>
<td>2</td>
</tr>
<tr>
<td>9 Hard Parse</td>
<td>.02</td>
<td>5.33</td>
<td>0</td>
</tr>
<tr>
<td>10 Connites and Rollbacks</td>
<td>.02</td>
<td>4.84</td>
<td>1</td>
</tr>
</tbody>
</table>