



Fine Tune Oracle Execution Plans for Performance Gains

Janis Griffin
Senior DBA, Confio Software

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?

- What are Execution Plans
- How to View Them
- Interpret Plan Details / Tips / Techniques
- Tune – Case Study
- Additional Tuning Aids – 10g / 11g

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:

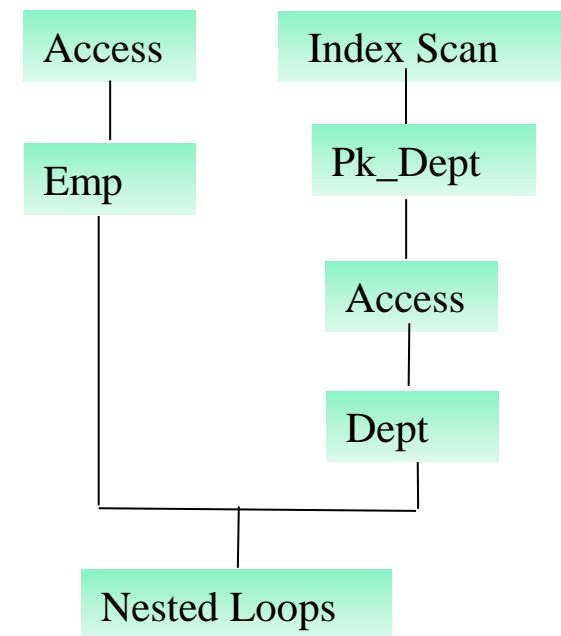
AGGREGATE
ALL
BY INDEX ROWID
BY LOCAL INDEX ROWID
BY USER ROWID
CARTESIAN
FAST FULL SCAN
FULL
FULL OUTER
FULL SCAN
GROUP BY
ITERATOR
ORDER BY
OUTER
PARTITION
RANGE SCAN
SAMPLE FAST FULL SCAN
SKIP SCAN
UNIQUE

What are Execution Plans

- 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;
```

Id	Operation & Option	Name
0	SELECT STATEMENT	
1	NESTED LOOPS	
2	TABLE ACCESS BY INDEX ROWID	DEPT
3	INDEX UNIQUE SCAN	PK_DEPT
4	TABLE ACCESS FULL	EMP



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

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

■ 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

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;
```

SET AUTOTRACE TRACEONLY:

```
Execution Plan  
-----  
Plan hash value: 568005898  
-----  
| Id | Operation | Name | Rows | Bytes | Cost (<%CPU>) | Time  
-----  
| 0 | SELECT STATEMENT | | 1503 | 72144 | 15 (<0>) | 00:00:01  
| 1 | NESTED LOOPS | | 1503 | 72144 | 15 (<0>) | 00:00:01  
| 2 | TABLE ACCESS BY INDEX ROWID | DEPT | 1 | 23 | 2 (<0>) | 00:00:01  
|* 3 | INDEX UNIQUE SCAN | PK_DEPT | 1 | | 1 (<0>) | 00:00:01  
|* 4 | TABLE ACCESS FULL | EMP | 1503 | 37575 | 13 (<0>) | 00:00:01  
-----  
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  
124620 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>
- Run it for expensive data access targets

Table Info & Column Statistics

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

COLUMN_NAME	NUM_DISTINCT	NUM_NULLS	NUM_BUCKETS	DENSITY	SAMPLE_SIZE
COMM	1534	4430	1	.00065189	1583
DEPTNO	4	0	4	.00083153	6013
EMPNO	6013	0	1	.000166306	6013
ENAME	6013	0	254	.000166306	6013
HIREDATE	88	0	1	.011363636	6013
JOB	22	0	22	.00083153	6013
MGR	6	6000	1	.166666667	13
SAL	6000	0	1	.000166667	6013

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

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

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

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

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

```
  DEPTNO  COUNT(*)
-----  -
    10      77
    20    1500
    30     478
    40    3958
```

Would an index on EMP.DEPTNO increase performance?

Histograms

```
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' ...
```

COLUMN_NAME	NUM_DISTINCT	NUM_NULLS	NUM_BUCKETS	DENSITY	SAMPLE_SIZE	HISTOGRAM
COMM	1534	4430	1	.00065189	1583	NONE
DEPTNO	4	0	4	.000083153	6013	FREQUENCY
EMPNO	6013	0	1	.000166306	6013	NONE
ENAME	6013	0	254	.000166306	6013	HEIGHT BALANCED
HIREDATE	88	0	1	.011363636	6013	NONE
JOB	22	0	22	.000083153	6013	FREQUENCY
MGR	6	6000	1	.166666667	13	NONE
SAL	6000	0	1	.000166667	6013	NONE

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

COLUMN_NAME	NUM_DISTINCT	NUM_NULLS	NUM_BUCKETS	DENSITY	SAMPLE_SIZE	HISTOGRAM
COMM	1534	4430	1	.00065189	1583	NONE
DEPTNO	4	0	2	.201057724	6013	HEIGHT BALANCED
EMPNO	6013	0	1	.000166306	6013	NONE
ENAME	6013	0	254	.000166306	6013	HEIGHT BALANCED
HIREDATE	88	0	1	.011363636	6013	NONE
JOB	22	0	22	.000083153	6013	FREQUENCY
MGR	6	6000	1	.166666667	13	NONE
SAL	6000	0	1	.000166667	6013	NONE

Additional Info - Bind Values

■ V\$SQL_BIND_CAPTURE

- STATISTICS_LEVEL = TYPICAL or ALL
- Collected at 15 minute intervals

```
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:

Name	Type
EMPNO	NUMBER(4)
ENAME	VARCHAR2(10)
JOB	VARCHAR2(9)
MGR	NUMBER(4)
HIREDATE	DATE
SAL	NUMBER(7,2)
COMM	NUMBER(7,2)
DEPTNO	NUMBER(2)

■ 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

```
SQL> select * from table(dbms_xplan.display_cursor('0zz5h1003f2dw',0));  
SQL_ID 0zz5h1003f2dw, child number 0  
-----  
SELECT empno EID, ename "Employee_name", dname "Department", hiredate  
"Date_Hired", loc "Location" FROM emp e, dept d WHERE d.deptno =:p1  
AND e.deptno = d.deptno  
Plan hash value: 568005898  
-----  
| Id | Operation | Name | Rows | Bytes | Cost (%CPU)| Time  
-----  
| 0 | SELECT STATEMENT | | | | 15 (100)|  
| 1 | NESTED LOOPS | | 3958 | 185K | 15 (0)| 00:00:01  
| 2 | TABLE ACCESS BY INDEX ROWID | DEPT | 1 | 23 | 2 (0)| 00:00:01  
|* 3 | INDEX UNIQUE SCAN | PK_DEPT | 1 | | 1 (0)| 00:00:01  
|* 4 | TABLE ACCESS FULL | EMP | 3958 | 98950 | 13 (0)| 00:00:01  
-----  
Predicate Information (identified by operation id):  
-----  
3 - access("D"."DEPTNO"=TO_NUMBER(:P1))  
4 - filter("E"."DEPTNO"=TO_NUMBER(:P1))
```

Plans Not Created Equal

```
SELECT company, attribute FROM data_out WHERE segment = :B1
```

- Wait Time – 100% on “db file scattered read”
- Plan from EXPLAIN PLAN

```
SELECT STATEMENT Optimizer=ALL_ROWS (Cost=1 Card=1 Bytes=117)
  TABLE ACCESS (BY INDEX ROWID) OF 'DATA_OUT' (TABLE) (Cost=1 Card=1 Bytes=117)
    INDEX (UNIQUE SCAN) OF 'IX1_DATA_OUT' (INDEX (UNIQUE)) (Cost=1 Card=1)
```

- Plan from V\$SQL_PLAN using DBMS_XPLAN

```
select * from table(dbms_xplan.display_cursor('az7r9s3wpgg7n',0));
```

```
-----
```

Id	Operation	Name	Rows	Bytes	Cost (%CPU)	Time
0	SELECT STATEMENT				370 (100)	
* 1	TABLE ACCESS FULL	DATA_OUT	1	117	370 (4)	00:00:05

```
-----
```

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

```
-----
```

```
1 - 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

Tune the Execution Plan

- 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 scew?

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?

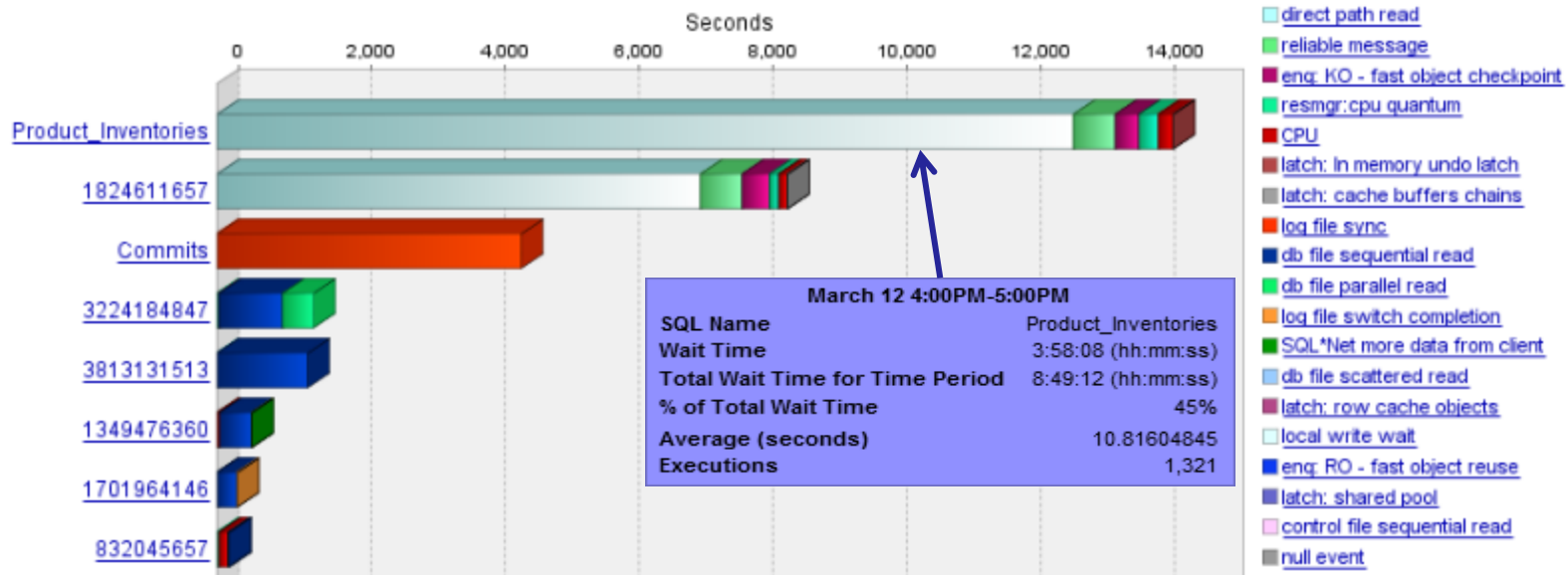
Home > Trend for CECE_JGRIFFIN-2(Oracle) > Mar 12 > 4PM-5PM

Day: Saturday - March 12, 2011 Time: 4:00PM to 5:00PM Refreshed on: 03/15/11 04:33:19 PM

Timeslice SQL Waits Programs DB Users O/S Users Machines Sessions Files Plans Objects Blockers

View Historical Charts for SQL: [Show SQL Text](#) [Email Chart](#)

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



Actual Plan

Execution Plan

Plan hash value: 1499855773

Id	Operation	Name	Rows	Bytes	Cost (%CPU)	Time
0	SELECT STATEMENT		1294	4122K	3016 (1)	00:00:37
* 1	COUNT STOPKEY					
* 2	HASH JOIN		1294	4122K	3016 (1)	00:00:37
* 3	HASH JOIN RIGHT OUTER		10	32230	24 (5)	00:00:01
* 4	TABLE ACCESS FULL	PRODUCT_DESCRIPTIONS	1	2070	18 (0)	00:00:01
* 5	TABLE ACCESS BY INDEX ROWID	PRODUCT_INFORMATION	10	11530	5 (0)	00:00:01
* 6	INDEX RANGE SCAN	PROD_CATEGORY_IX	4		1 (0)	00:00:01
* 7	TABLE ACCESS FULL	INVENTORIES	9575	364K	2992 (1)	00:00:36

Predicate Information (identified by operation id):

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

Note

- dynamic sampling used for this statement

Statistics

```
49 recursive calls
0 db block gets
11348 consistent gets
88 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
6 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
```

Actual Plan – No Statistics

Execution Plan

Plan hash value: 1499855773

Id	Operation	Name	Rows	Bytes	Cost (%CPU)	Time
0	SELECT STATEMENT		1294	4122K	3016 (1)	00:00:37
* 1	COUNT STOPKEY					
* 2	HASH JOIN		1294	4122K	3016 (1)	00:00:37
* 3	HASH JOIN RIGHT OUTER		10	32230	24 (5)	00:00:01
* 4	TABLE ACCESS FULL	PRODUCT_DESCRIPTIONS	1	2070	18 (0)	00:00:01
5	TABLE ACCESS BY INDEX ROWID	PRODUCT_INFORMATION	10	11530	5 (0)	00:00:01
* 6	INDEX RANGE SCAN	PROD_CATEGORY_IX	4		1 (0)	00:00:01
* 7	TABLE ACCESS FULL	INVENTORIES	9575	364K	2992 (1)	00:00:36

Predicate Information (identified by operation id):

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

Note

- dynamic sampling used for this statement

Statistics

```
49 recursive calls
0 db block gets
11348 consistent gets
88 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
6 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
```

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
,      i.catalog_url
FROM product_information i
,      product_descriptions d
WHERE d.product_id (+) = i.product_id
AND   d.language_id (+) = sys_context('USERENV','LANG')
```

No Statistics

COLUMN_NAME	NUM_DISTINCT	NUM_NULLS	NUM_BUCKETS	DENSITY	SAMPLE_SIZE	HISTOGRAM
PRODUCT_ID						NONE
QUANTITY_ON_HAND						NONE
WAREHOUSE_ID						NONE

DBMS_STATS

```
exec dbms_stats.gather_schema_stats( -
ownname      => '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

Actual Plan – With Statistics

Plan hash value: 3363672010

Id	Operation	Name	Rows	Bytes	Cost (%CPU)	Time
0	SELECT STATEMENT				3007 (100)	
* 1	COUNT STOPKEY					
* 2	HASH JOIN OUTER		1	1183	3007 (1)	00:00:37
3	NESTED LOOPS					
4	NESTED LOOPS		1	1166	2988 (1)	00:00:36
* 5	TABLE ACCESS FULL	INVENTORIES	896	11648	2988 (1)	00:00:36
* 6	INDEX RANGE SCAN	PROD_CATEGORY_IX	1		0 (0)	
* 7	TABLE ACCESS BY INDEX ROWID	PRODUCT_INFORMATION	1	1153	0 (0)	
* 8	TABLE ACCESS FULL	PRODUCT_DESCRIPTIONS	10	170	18 (0)	00:00:01

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
```

How to tune ?

```
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

NAME	POSITION	DATATYPE_STRING	VALUE_STRING
:B3	1	NUMBER	136 - category_id
:B2	2	NUMBER	454 - warehouse_id
:B1	3	NUMBER	15 - rownum

INVENTORIES Table **Filter (warehouse_id=454) -1000 records**

COLUMN_NAME	NUM_DISTINCT	NUM_NULLS	NUM_BUCKETS	DENSITY	SAMPLE_SIZE	HISTOGRAM
PRODUCT_ID	1000	0	1	.001	894861	NONE
QUANTITY_ON_HAND	894861	0	1	1.1175E-06	894861	NONE
WAREHOUSE_ID	999	0	1	.001001001	894861	NONE

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

PCT_OF_INVENTORY

.11

How to tune ?

```
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

NAME	POSITION	DATATYPE_STRING	VALUE_STRING
:B3	1	NUMBER	136 - category_id

PRODUCT_INFORMATION Table

Filter (category_id = 136) – 6 records

COLUMN_NAME	NUM_DISTINCT	NUM_NULLS	NUM_BUCKETS	DENSITY	SAMPLE_SIZE	HISTOGRAM
CATALOG_URL	1000	0	1	.001	1000	NONE
CATEGORY_ID	196	0	196	.0005	1000	FREQUENCY
LIST_PRICE	910	0	1	.001098901	1000	NONE
MIN_PRICE	916	0	1	.001091703	1000	NONE
PRODUCT_DESCRIPTION	1000	0	1	.001	1000	NONE
PRODUCT_ID	1000	0	1	.001	1000	NONE

USER_TAB_HISTOGRAMS

```
select
  endpoint_value,
  endpoint_number,
  endpoint_number - nvl(prev_number,0) frequency
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
```

ENDPOINT_VALUE	ENDPOINT_NUMBER	FREQUENCY
134	676	4
135	683	7
136	689	6
137	693	4
138	697	4
139	699	2
140	706	7

New Plan

CREATE INDEX inventories_ix1 ON inventories(warehouse_id);

Plan hash value: 750880835

Id	Operation	Name	Rows	Bytes	Cost (%CPU)	Time
0	SELECT STATEMENT				927 (100)	
* 1	COUNT STOPKEY					
* 2	HASH JOIN		11	2266	927 (1)	00:00:12
* 3	HASH JOIN OUTER		7	1351	27 (4)	00:00:01
4	TABLE ACCESS BY INDEX ROWID	PRODUCT_INFORMATION	7	1232	8 (0)	00:00:01
* 5	INDEX RANGE SCAN	PROD_CATEGORY_IX	7		1 (0)	00:00:01
* 6	TABLE ACCESS FULL	PRODUCT_DESCRIPTIONS	10	170	18 (0)	00:00:01
7	TABLE ACCESS BY INDEX ROWID	INVENTORIES	896	11648	900 (0)	00:00:11
* 8	INDEX RANGE SCAN	INVENTORIES_IX1	896		4 (0)	00:00:01

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")
- 5 - access("I"."CATEGORY_ID"=:B3)
- 6 - filter("D"."LANGUAGE_ID"=SYS_CONTEXT('USERENV','LANG'))
- 8 - access("INVENTORIES"."WAREHOUSE_ID"=:B2)

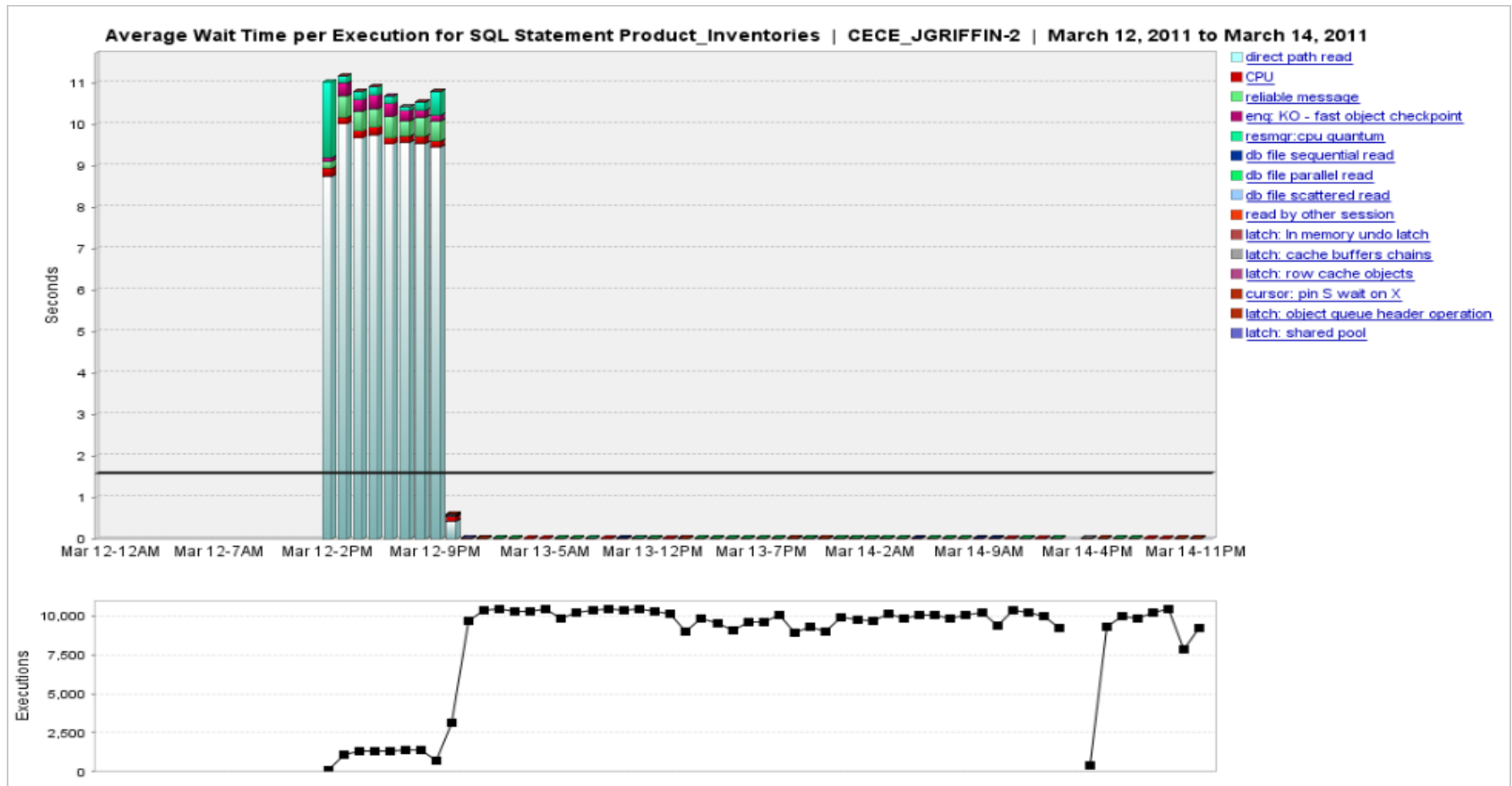
March 13 4:00AM-5:00AM

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

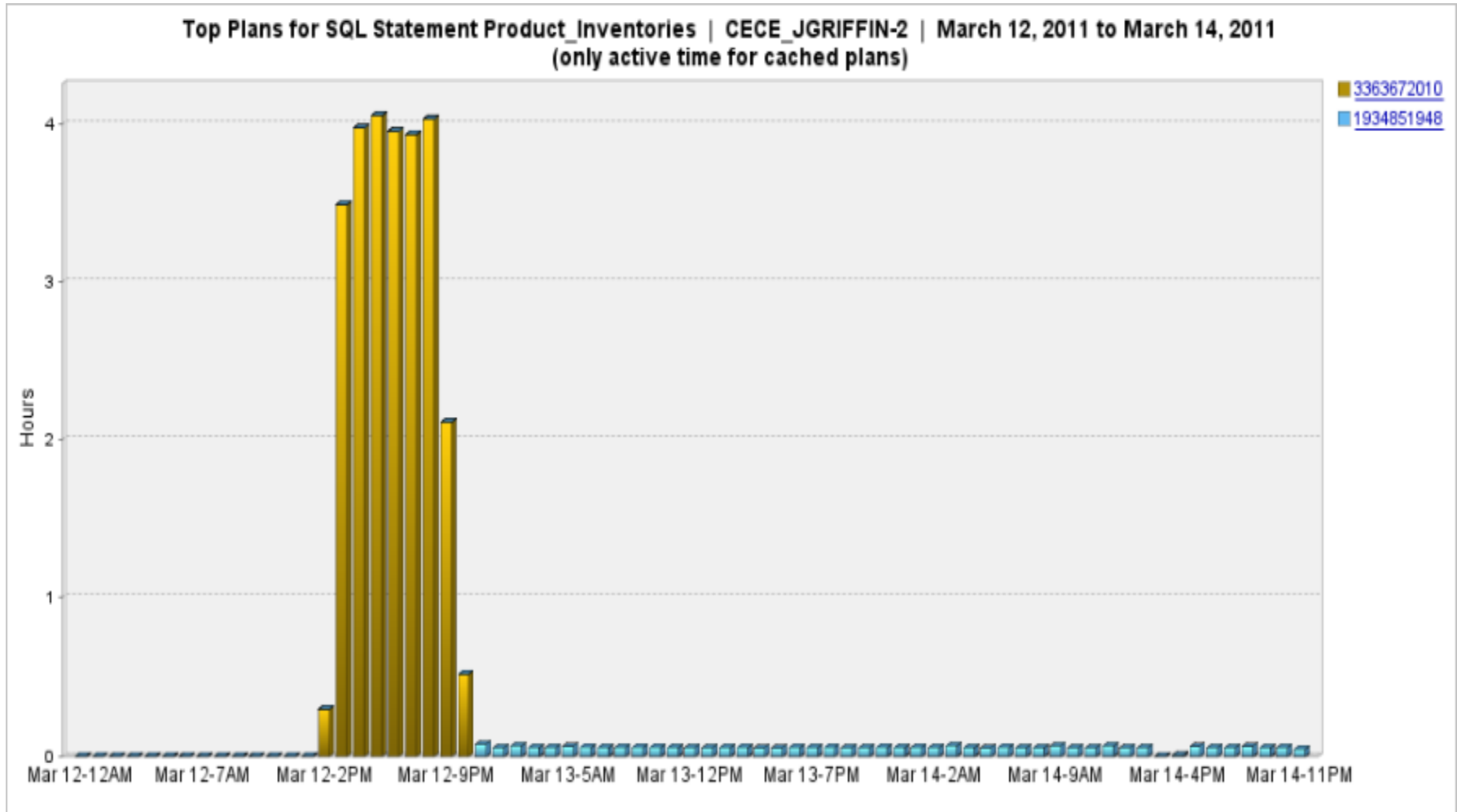
100% on CPU

Did the performance improve?

CREATE INDEX inventories_ix1 ON inventories(warehouse_id);



Which is the better Plan?



- Execution Plans show internal steps for statements
 - Shows how data is accessed, manipulated and joined
 - Shows expensive steps with Cost and Time spent.
 - Gives information on Bind Variables (predicate information)
- Viewing actual plan is better than EXPLAIN PLAN
 - V\$SQL_PLAN using DBMS_XPLAN.display_cursor('&sql_id', 0)
 - Tracing / TKPROF
- Gather additional data when tuning execution plans
 - Table sizes, Index selectivity and column details
 - How Statistic Gathering is performed
 - Bind values from V\$SQL_BIND_CAPTURE
- Tune the execution plans that make a difference
 - Monitor response time to find the biggest problems - Use Ignite

- Wait-Based Performance Tools
- Ignite8
 - Ignite for SQL Server, Oracle, DB2, Sybase
- Helps show which SQL to tune
- Based in Colorado, worldwide customers
- Free trial at www.confio.com
- [_http://www.ignitefree.com](http://www.ignitefree.com) – Free Current View

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

■ New Functions 11g

BUILD_PLAN_XML	Return the last plan, or a named plan, explained as XML
DISPLAY	Shows the last plan explained – EXPLAIN PLAN ** Only FUNCTION in Oracle 9i
DISPLAY_AWR	Format & display the plan of a stored SQL statement in AWR
DISPLAY_CURSOR	Format & display the execution plan of any loaded cursor
DISPLAY_PLAN	Return the last plan, or a named plan, explained as a CLOB
DISPLAY_SQLSET	Format & display the execution plan of statements stored in a SQL tuning set
DISPLAY_SQL_PLAN_BASELINE	Displays one or more plans for the specified SQL statement
FORMAT_NUMBER	Returns a number as a string
FORMAT_NUMBER2	Returns a number as a string formatted with a leading space (CHR(32))
FORMAT_SIZE	Undocumented
FORMAT_SIZE2	Undocumented
FORMAT_TIME_S	Undocumented
PREPARE_PLAN_XML_QUERY	- 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
PREPARE_RECORDS	Used Internally
VALIDATE_FORMAT	Used Internally

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



```
- <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>00:00:01 </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

ALTER_SQL_PLAN_BASELINE	Changes an attribute of a single plan or all plans associated with a SQL statement using the attribute name/value format
CONFIGURE	Set configuration options for the SQL Management Base (SMB) as well as the maintenance of SQL plan baselines
CREATE_STGTAB_BASELINE	Creates a staging table that will be used for the purpose of transporting SQL plan baselines from one system to another
DROP_SQL_PLAN_BASELINE	Drops a single plan, or all plans associated with a SQL statement
EVOLVE_SQL_PLAN_BASELINE	Evolves SQL plan baselines associated with one or more SQL statements
LOAD_PLANS_FROM_CURSOR_CACHE	Loads one or more plans present in the cursor cache for a SQL statement
LOAD_PLANS_FROM_SQLSET	Loads plans stored in a SQL tuning set (STS) into SQL plan baselines
PACK_STGTAB_BASELINE	Packs (exports) SQL plan baselines from SQL management base into a staging table
UNPACK_STGTAB_BASELINE	Unpacks (imports) SQL plan baselines from a staging table into SQL management base

■ Without Statistics

Select	Type	Findings	Recommendations	Rationale	Benefit (%)	New Explain Plan	Compare Explain Plans
<input checked="" type="radio"/>	Statistics	Table "CSU"."CLASS" was not analyzed.	Consider collecting optimizer statistics for this table.	The optimizer requires up-to-date statistics for the table in order to select a good execution plan.			
<input type="radio"/>	Statistics	Table "CSU"."REGISTRATION" was not analyzed.	Consider collecting optimizer statistics for this table.	The optimizer requires up-to-date statistics for the table in order to select a good execution plan.			
<input type="radio"/>	Index	The execution plan of this statement can be improved by creating one or more indices.	Consider running the Access Advisor to improve the physical schema design or creating the recommended index. CSU.CLASS(UPPER("NAME"),"CLASS_LEVEL") CSU.REGISTRATION("SIGNUP_DATE") CSU.REGISTRATION("CLASS_ID","STUDENT_ID") CSU.REGISTRATION("STUDENT_ID","CLASS_ID","CANCELLED")	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.	94.12		

■ With Statistics

Select	Type	Findings	Recommendations	Rationale	Benefit (%)	New Explain Plan	Compare Explain Plans
<input checked="" type="radio"/>	Index	The execution plan of this statement can be improved by creating one or more indices.	Consider running the Access Advisor to improve the physical schema design or creating the recommended index. CSU.CLASS(UPPER("NAME"),"CLASS_LEVEL") CSU.REGISTRATION("SIGNUP_DATE") CSU.REGISTRATION("CLASS_ID","STUDENT_ID") CSU.REGISTRATION("STUDENT_ID","CLASS_ID")	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.	91.46		

Manual Run - Works in 10g/11g

```
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. ');
  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

```
Tuning Task Name      : cpn6dwxd743s1
Tuning Task Owner    : SYS
Workload Type        : Single SQL Statement
Scope                : COMPREHENSIVE
Time Limit(seconds)  : 60
Completion Status     : COMPLETED
Started at           : 11/10/2010 16:29:44
Completed at         : 11/10/2010 16:29:44
```

```
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.
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.

Oracle - Sql Tuning Advice



New index contains same columns but reversed.

```
EXPLAIN PLANS SECTION
-----
1- Original
Plan hash value: 642510383
-----
| Id | Operation | Name | Rows | Byte
s | Cost (%CPU)| Time | |
-----
| 0 | SELECT STATEMENT | | 36 | 1004
4 | 39 (<0>)| 00:00:01 | |
| 1 | TABLE ACCESS BY INDEX ROWID | CON_ALERT_DB_STATUS_HISTORY | 36 | 1004
4 | 39 (<0>)| 00:00:01 | |
!* 2 | INDEX RANGE SCAN | SYS_C0013034 | 36 |
| 4 (<0>)| 00:00:01 |
-----
Predicate Information (identified by operation id):
-----
2 - access("ALERTID"=:1 AND "DBID"=:2)

2- Using New Indices
Plan hash value: 2465160697
-----
| Id | Operation | Name | Rows | Byte
s | Cost (%CPU)| Time | |
-----
| 0 | SELECT STATEMENT | | 36 | 1004
4 | 17 (<0>)| 00:00:01 | |
| 1 | TABLE ACCESS BY INDEX ROWID | CON_ALERT_DB_STATUS_HISTORY | 36 | 1004
4 | 17 (<0>)| 00:00:01 | |
!* 2 | INDEX RANGE SCAN | IDX$$_11570001 | 36 |
| 1 (<0>)| 00:00:01 |
-----
Predicate Information (identified by operation id):
-----
2 - access("DBID"=:2 AND "ALERTID"=:1)
```

Select	Type	Findings	Recommendations	Rationale	Benefit (%)	New Explain Plan	Compare Explain Plans
<input checked="" type="radio"/>	Statistics	Table "CSU"."CLASS" was not analyzed.	Consider collecting optimizer statistics for this table.	The optimizer requires up-to-date statistics for the table in order to select a good execution plan.			
<input type="radio"/>	Statistics	Table "CSU"."REGISTRATION" was not analyzed.	Consider collecting optimizer statistics for this table.	The optimizer requires up-to-date statistics for the table in order to select a good execution plan.			
<input type="radio"/>	Index	The execution plan of this statement can be improved by creating one or more indices.	Consider running the Access Advisor to improve the physical schema design or creating the recommended index. CSU.CLASS(UPPER("NAME"),"CLASS_LEVEL") CSU.REGISTRATION("SIGNUP_DATE") CSU.REGISTRATION("CLASS_ID","STUDENT_ID") CSU.REGISTRATION("STUDENT_ID","CLASS_ID","CANCELLED")	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.	94.12		

■ With Statistics

Select	Type	Findings	Recommendations	Rationale	Benefit (%)	New Explain Plan	Compare Explain Plans
<input checked="" type="radio"/>	Index	The execution plan of this statement can be improved by creating one or more indices.	Consider running the Access Advisor to improve the physical schema design or creating the recommended index. CSU.CLASS(UPPER("NAME"),"CLASS_LEVEL") CSU.REGISTRATION("SIGNUP_DATE") CSU.REGISTRATION("CLASS_ID","STUDENT_ID") CSU.REGISTRATION("STUDENT_ID","CLASS_ID")	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.	91.46		

ADDM Commands – only in 11g



(In 10g, must run the addmrpt.sql in ?/rdbms/admin)

```
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 bsnap,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, bsnap, esnap);
```

```
END;
```

```
/
```

```
SET LONG 100000
```

```
SET PAGESIZE 50000
```

```
SELECT DBMS_ADDM.GET_REPORT(:tname) FROM DUAL;
```

ADDM Example – Current Hour

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 4108810298.  
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  
-----  


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


```

- Wait-Based Performance Tools
- Ignite8
 - Ignite for SQL Server, Oracle, DB2, Sybase
- Helps show which SQL to tune
- Based in Colorado, worldwide customers
- Free trial at www.confio.com
- [_http://www.ignitefree.com](http://www.ignitefree.com) – Free Current View