Need for Speed? Top Five Oracle Performance Tuning Tips

Janis Griffin

Senior Sales Consultant

Quest

Who Am I?

Janis.Griffin@quest.com

Twitter® - @DoBoutAnything

- Current – 30+ Years in Oracle®, DB2®, ASE, SQL Server®, MySQL®
- DBA and Developer

Specialize in Performance Tuning

Customers Common Question: How do I tune it?

ORACLE ACE

Quest

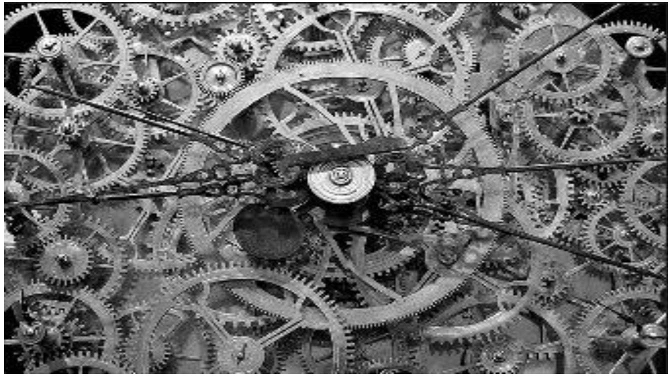
Agenda

- Challenges of Tuning
  - Monitor Wait Time
    - Find the right SQL statements to work on
    - Get Baseline Metrics
  - Review the Execution Plan
    - Know which Optimizer Features are being used
  - Gather Object Information
    - Review Table, Column, Index & Constraint information
    - Understand Column Selectivity & Statistics
  - Find the Driving Table
    - Consider SQL Diagramming
  - Engineer out the Stupid

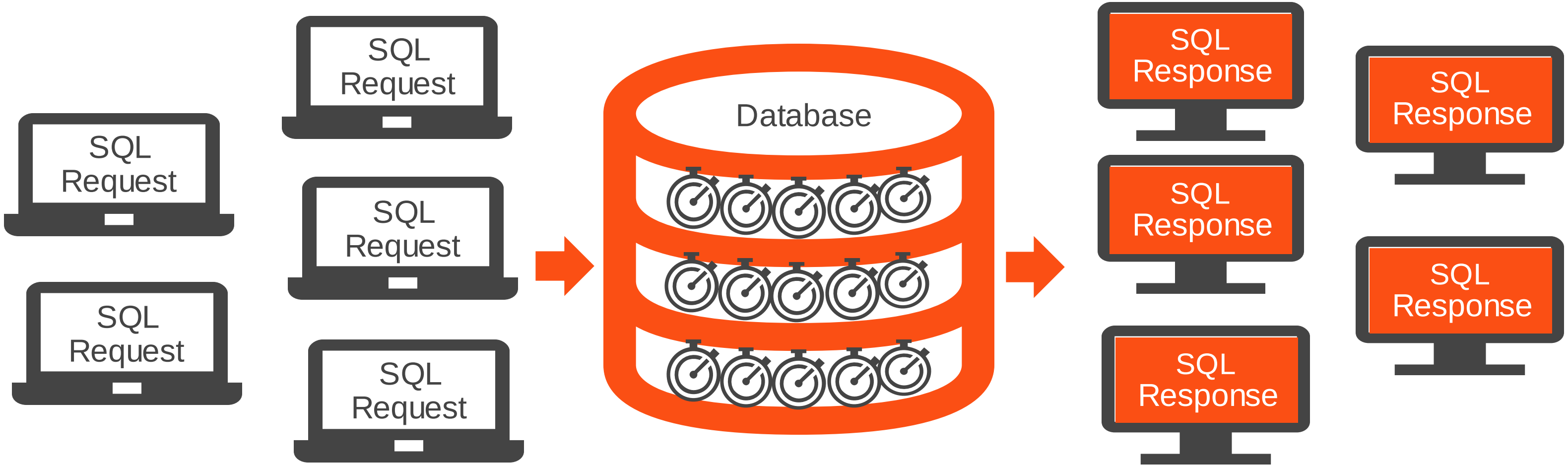
Challenges Of Tuning

- SQL Tuning is Hard
  - Who should tune – DBA or Developer
  - Which SQL to tune
- Requires Expertise in Many Areas
  - Technical – Plan, Data Access, SQL Design
  - Business – What is the Purpose of SQL?
- Tuning Takes Time
  - Large Number of SQL Statements
  - Each Statement is Different
- Low Priority in Some Companies
  - Vendor Applications
  - Focus on Hardware or System Issues

I THINK WE SHOULD BUILD AN SQL DATABASE.
UH-OH
E-mail: SCOTTADAMS@AOL.COM
S. Adams
DOES HE UNDERSTAND WHAT HE SAID OR IS IT SOMETHING HE SAW IN A TRADE MAGAZINE AD?
© 1996 United Feature Syndicate, Inc.(NYC)
WHAT COLOR DO YOU WANT THAT DATABASE?
I THINK MAUVE HAS THE MOST RAM.



1. Monitor Wait Time



- Identify Wait Time at every step and rank them by user impact
- Understand the total time a Query spends in Database
- Oracle helps by providing Wait Events

Wait Event Information

V$SESSION

SID
SERIAL#
USERNAME
MACHINE
PROGRAM
MODULE
ACTION
CLIENT_INFO
SQL_ID
SQL_CHILD_NUMBER
EVENT
P1TEXT
P1
P2TEXT
P2
P3TEXT
P3
STATE (WAITING, WAITED)
BLOCKING_SESSION

V$SQL

SQL_ID
SQL_FULLTEXT
PLAN_HASH_VALUE
CHILD_NUMBER
IS_BIND_SENSITIVE
IS_BIND_AWARE
IS_SHAREABLE
SQL_PROFILE
SQL_PATCH
SQL_PLAN_BASELINE
BIND_DATA
IS_REOPTIMIZABLE
IS_RESOLVED_ADAPTIVE_PLAN

V$SQLAREA

SQL_ID
EXECUTIONS
PARSE_CALLS
DISK_READS
BUFFER_GETS

V$SQL_PLAN

SQL_ID
PLAN_HASH_VALUE
CHILD_NUMBER
OPERATION
OBJECT_NAME
OTHER_XML

DBA_OBJECTS

OBJECT_ID
OBJECT_NAME
OBJECT_TYPE

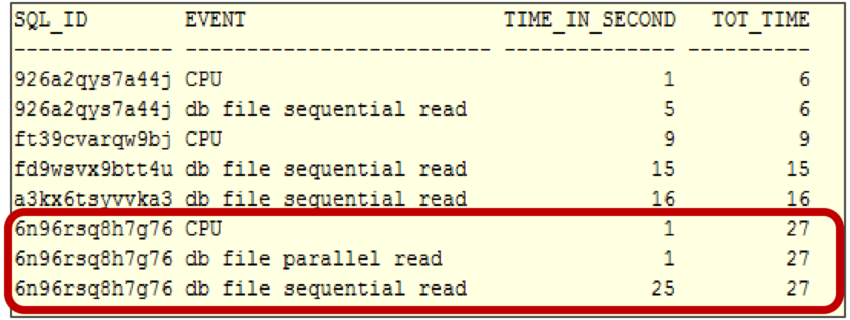
V$SQL_BIND_CAPTURE

SQL_ID
NAME
VALUE_STRING
DATATYPE_STRING
LAST_CAPTURED

Base Query - Not Rocket Science

```
INSERT INTO wta_data
SELECT
   sid, serial#, username, program, module, action,
   machine, osuser, sql_id, blocking_session,
   decode(state, 'WAITING', event, 'CPU') event,
   p1, p1text, p2, p2text, p3, p3text,
   SYSDATE date_time
FROM V$SESSION s
WHERE s.status = 'ACTIVE'
AND wait_class != 'Idle'
AND username != USER;
```

```
SELECT wta.sql_id, wta.event, COUNT(*)
time_in_second, tot_time
FROM wta_data wta,
       (SELECT sql_id, COUNT(*) tot_time
         FROM wta_data GROUP BY sql_id) tot
WHERE wta.sql_id = tot.sql_id
GROUP BY wta.sql_id,wta.event, tot_time
ORDER BY tot_time,wta.sql_id,
time_in_second;
```



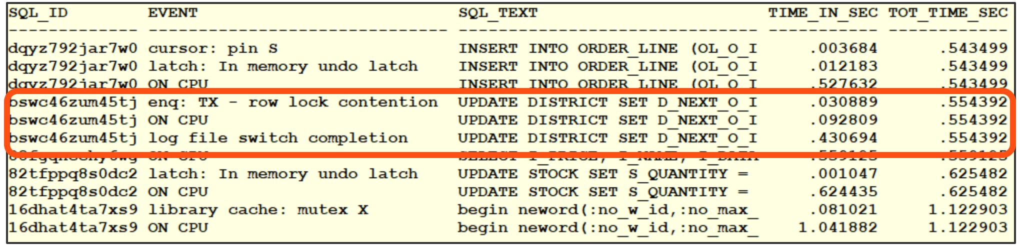
```
SQL_ID        EVENT                     TIME_IN_SECOND   TOT_TIME
------------- ------------------------- -------------- ----------
926a2qys7a44j CPU                                    1          6
926a2qys7a44j db file sequential read                5          6
ft39cvarqw9bj CPU                                    9          9
fd9wsvx9btt4u db file sequential read               15         15
a3kx6tsyvvka3 db file sequential read               16         16
6n96rsq8h7g76 CPU                                    1         27
6n96rsq8h7g76 db file parallel read                  1         27
6n96rsq8h7g76 db file sequential read               25         27
```

Active Session History (ASH)

- V$ACTIVE_SESSION_HISTORY
  - Data warehouse for session statistics
  - Oracle 10g and higher
  - Data is sampled every second
  - Holds at least one hour of history
  - Never bigger than:
    - 2% of SGA_TARGET
    - 5% of SHARED_POOL (if automatic sga sizing is turned off)

```
SELECT summary.sql_id, event, sql_text, event_time_in_seconds, tot_time_in_seconds
FROM (SELECT a.sql_id, DECODE(a.session_state, 'WAITING', a.event, 'ON CPU') event,
       SUBSTR(v.sql_text,1,30) sql_text,
       SUM(a.wait_time + a.time_waited)/1000000 event_time_in_seconds
      FROM v$active_session_history a, v$sqlarea v, dba_users u
      WHERE a.sample_time BETWEEN SYSDATE - 1 AND SYSDATE
      AND a.sql_id = v.sql_id AND a.user_id = u.user_id AND u.username <>'SYS'
      GROUP BY a.sql_id, DECODE(A.session_state, 'WAITING', a.event, 'ON CPU'),
              SUBSTR(v.sql_text,1,30)) detail,
     (SELECT sql_id, SUM(wait_time + time_waited)/1000000 tot_time_in_seconds
      FROM v$active_session_history
      WHERE sample_time BETWEEN SYSDATE - 1 AND SYSDATE GROUP BY sql_id) summary
WHERE detail.sql_id = summary.sql_id
ORDER by tot_time_in_seconds, sql_id, event_time_in_seconds
```

- WRH$_ACTIVE_SESSION_HISTORY
  - Above table gets flushed to this table
    - AKA – dba_hist_active_sess_history

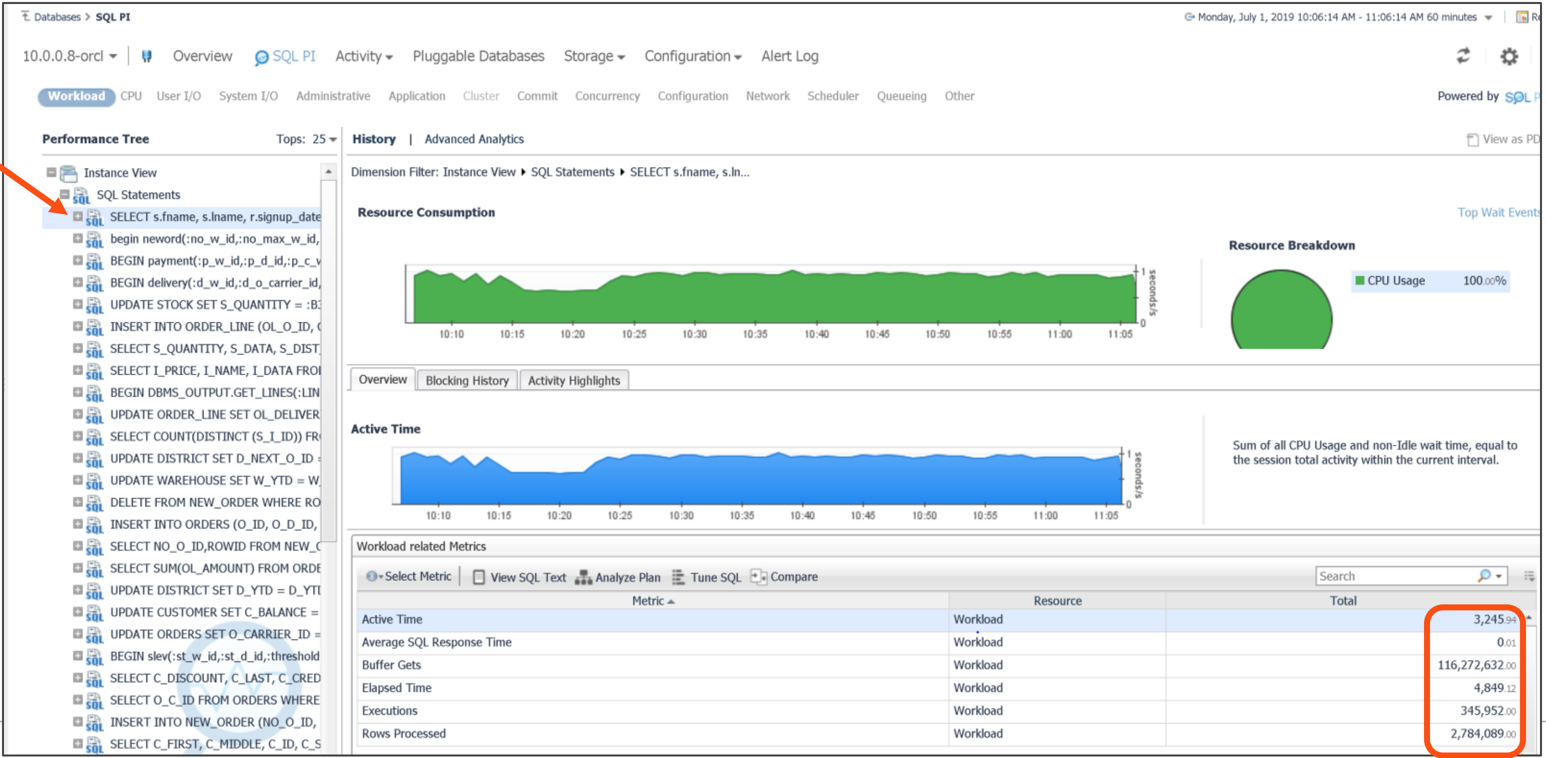


```
SQL_ID        EVENT                            SQL_TEXT                       TIME_IN_SEC TOT_TIME_SEC
------------- -------------------------------- ------------------------------ ----------- ------------
dqyz792jar7w0 cursor: pin S                    INSERT INTO ORDER_LINE (OL_O_I     .003684      .543499
dqyz792jar7w0 latch: In memory undo latch      INSERT INTO ORDER_LINE (OL_O_I     .012183      .543499
dqyz792jar7w0 ON CPU                           INSERT INTO ORDER_LINE (OL_O_I     .527632      .543499
bswc46zum45tj enq: TX - row lock contention    UPDATE DISTRICT SET D_NEXT_O_I     .030889      .554392
bswc46zum45tj ON CPU                           UPDATE DISTRICT SET D_NEXT_O_I     .092809      .554392
bswc46zum45tj log file switch completion       UPDATE DISTRICT SET D_NEXT_O_I     .430694      .554392
[illegible]   ON CPU                           SELECT I_PRICE, I_NAME, I_DATA     .559125      .559125
82tfppq8s0dc2 latch: In memory undo latch      UPDATE STOCK SET S_QUANTITY =      .001047      .625482
82tfppq8s0dc2 ON CPU                           UPDATE STOCK SET S_QUANTITY =      .624435      .625482
16dhat4ta7xs9 library cache: mutex X           begin neword(:no_w_id,:no_max_     .081021     1.122903
16dhat4ta7xs9 ON CPU                           begin neword(:no_w_id,:no_max_    1.041882     1.122903
```

- Need Tuning & Diagnostics Packs
  - On Enterprise Only

Wait Time Analysis

- Focus on SQL statements spending the most time in the database



Dimension Filter: Instance View ▸ SQL Statements ▸ SELECT s.fname, s.ln...

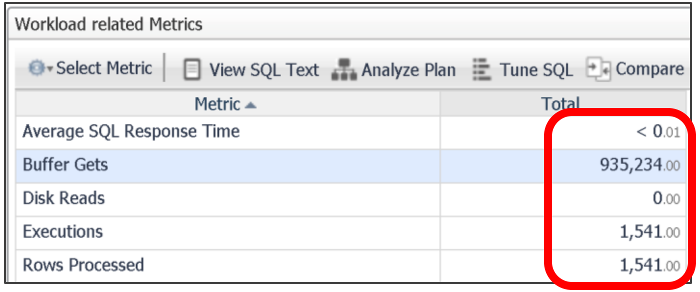
Instance View
SQL Statements
SELECT s.fname, s.lname, r.signup_date
begin neword(:no_w_id,:no_max_w_id,
BEGIN payment(:p_w_id,:p_d_id,:p_c_v
BEGIN delivery(:d_w_id,:d_o_carrier_id,
UPDATE STOCK SET S_QUANTITY = :B3
INSERT INTO ORDER_LINE (OL_O_ID, (
SELECT S_QUANTITY, S_DATA, S_DIST
SELECT I_PRICE, I_NAME, I_DATA FROI
BEGIN DBMS_OUTPUT.GET_LINES(:LIN
UPDATE ORDER_LINE SET OL_DELIVER
SELECT COUNT(DISTINCT (S_I_ID)) FR
UPDATE DISTRICT SET D_NEXT_O_ID =
UPDATE WAREHOUSE SET W_YTD = W
DELETE FROM NEW_ORDER WHERE RO
INSERT INTO ORDERS (O_ID, O_D_ID,
SELECT NO_O_ID,ROWID FROM NEW_C
SELECT SUM(OL_AMOUNT) FROM ORDE
UPDATE DISTRICT SET D_YTD = D_YTI
UPDATE CUSTOMER SET C_BALANCE =
UPDATE ORDERS SET O_CARRIER_ID =
BEGIN slev(:st_w_id,:st_d_id,:threshold
SELECT C_DISCOUNT, C_LAST, C_CRED
SELECT O_C_ID FROM ORDERS WHERE
INSERT INTO NEW_ORDER (NO_O_ID,
SELECT C_FIRST, C_MIDDLE, C_ID, C_S

Sum of all CPU Usage and non-Idle wait time, equal to the session total activity within the current interval.

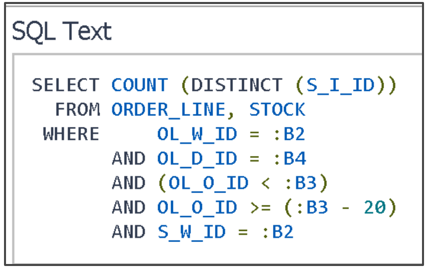
Metric ▲	Resource	Total
Active Time	Workload	3,245.94
Average SQL Response Time	Workload	0.01
Buffer Gets	Workload	116,272,632.00
Elapsed Time	Workload	4,849.12
Executions	Workload	345,952.00
Rows Processed	Workload	2,784,089.00

Benefits of Wait Time Analysis – Cont.

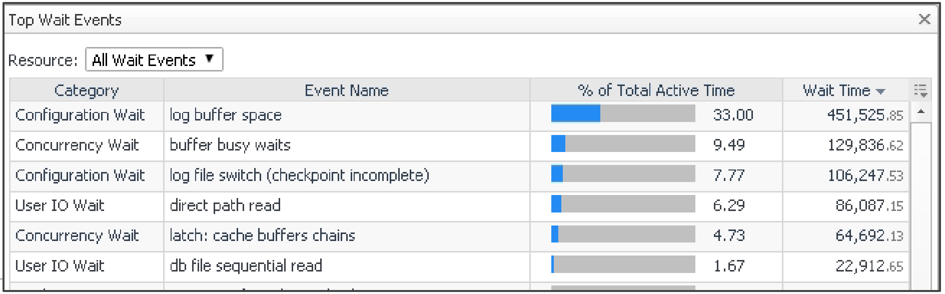
- Get baseline metrics
  - How long does it take now
  - What is acceptable (10 sec, 2 min, 1 hour)
  - Get number of Buffer Gets
    - Measurement to compare against while tuning
- Collect Wait Event Information
  - Locking / Blocking (enq)
  - I/O problem (db file sequential read)
  - Latch contention (latch)
  - Network slowdown (SQL*Net)
  - May be multiple issues
  - All have different resolutions


Workload related Metrics

Metric ▲	Total
Average SQL Response Time	< 0.01
Buffer Gets	935,234.00
Disk Reads	0.00
Executions	1,541.00
Rows Processed	1,541.00


SQL Text

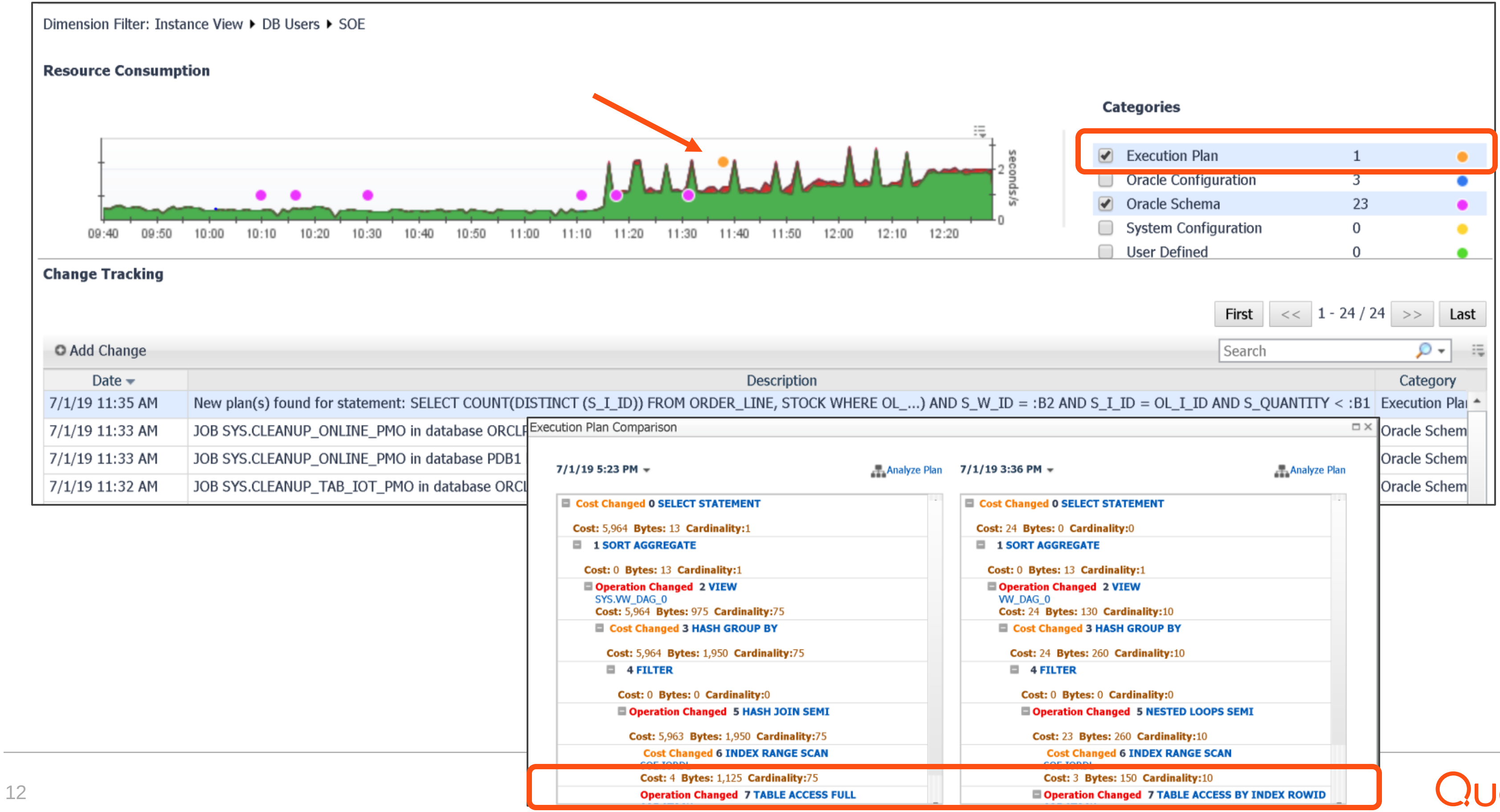
```
SELECT COUNT (DISTINCT (S_I_ID))
  FROM ORDER_LINE, STOCK
 WHERE      OL_W_ID = :B2
        AND OL_D_ID = :B4
        AND (OL_O_ID < :B3)
        AND OL_O_ID >= (:B3 - 20)
        AND S_W_ID = :B2
```


Top Wait Events

Resource: All Wait Events ▼

Category	Event Name	% of Total Active Time	Wait Time ▼
Configuration Wait	log buffer space	33.00	451,525.85
Concurrency Wait	buffer busy waits	9.49	129,836.62
Configuration Wait	log file switch (checkpoint incomplete)	7.77	106,247.53
User IO Wait	direct path read	6.29	86,087.15
Concurrency Wait	latch: cache buffers chains	4.73	64,692.13
User IO Wait	db file sequential read	1.67	22,912.65

Other Benefits: Query Suddenly runs slower



Dimension Filter: Instance View ▸ DB Users ▸ SOE

Resource Consumption

Categories

☑	Execution Plan	1	●
□	Oracle Configuration	3	●
☑	Oracle Schema	23	●
□	System Configuration	0	●
□	User Defined	0	●

Change Tracking

First << 1 - 24 / 24 >> Last

Add Change Search

Date	Description	Category
7/1/19 11:35 AM	New plan(s) found for statement: SELECT COUNT(DISTINCT (S_I_ID)) FROM ORDER_LINE, STOCK WHERE OL_...) AND S_W_ID = :B2 AND S_I_ID = OL_I_ID AND S_QUANTITY < :B1	Execution Plai
7/1/19 11:33 AM	JOB SYS.CLEANUP_ONLINE_PMO in database ORCLP	Oracle Schem
7/1/19 11:33 AM	JOB SYS.CLEANUP_ONLINE_PMO in database PDB1	Oracle Schem
7/1/19 11:32 AM	JOB SYS.CLEANUP_TAB_IOT_PMO in database ORCL	Oracle Schem

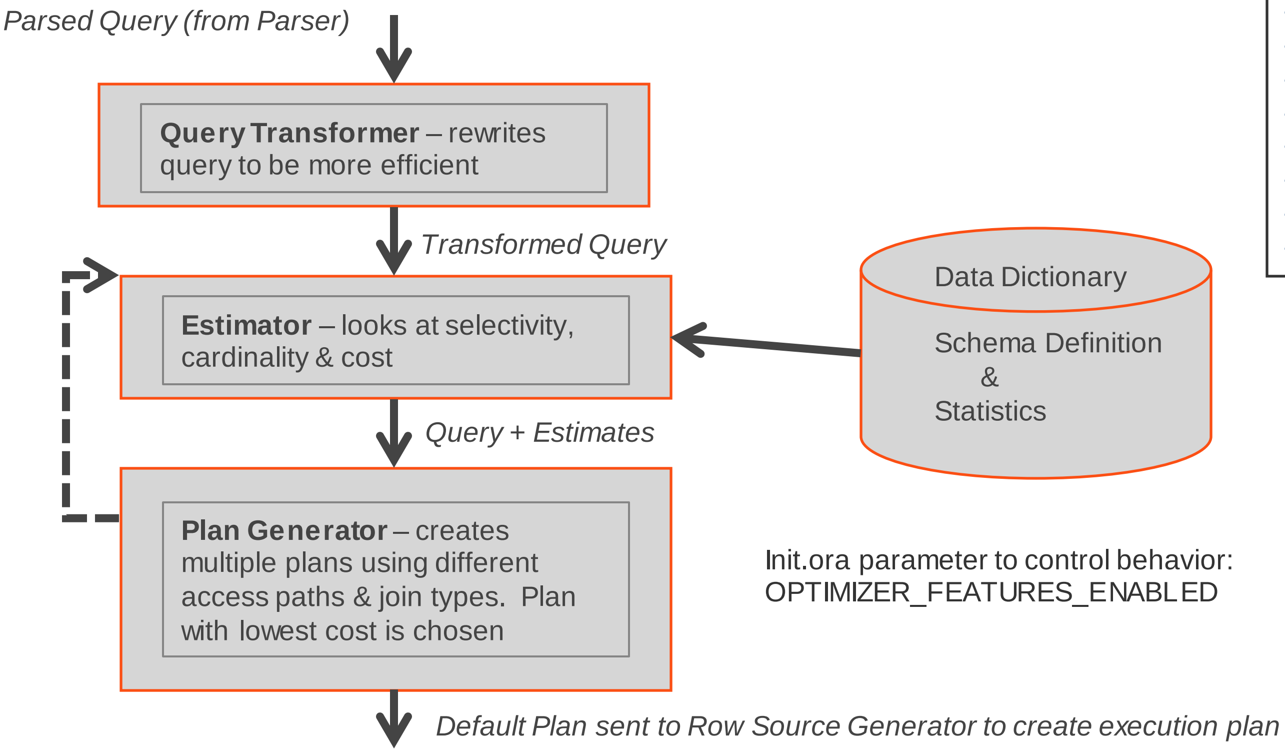
Execution Plan Comparison

7/1/19 5:23 PM	7/1/19 3:36 PM
Cost Changed 0 SELECT STATEMENT	Cost Changed 0 SELECT STATEMENT
Cost: 5,964 Bytes: 13 Cardinality:1	Cost: 24 Bytes: 0 Cardinality:0
1 SORT AGGREGATE	1 SORT AGGREGATE
Cost: 0 Bytes: 13 Cardinality:1	Cost: 0 Bytes: 13 Cardinality:1
Operation Changed 2 VIEW SYS.VW_DAG_0 Cost: 5,964 Bytes: 975 Cardinality:75	Operation Changed 2 VIEW VW_DAG_0 Cost: 24 Bytes: 130 Cardinality:10
Cost Changed 3 HASH GROUP BY	Cost Changed 3 HASH GROUP BY
Cost: 5,964 Bytes: 1,950 Cardinality:75	Cost: 24 Bytes: 260 Cardinality:10
4 FILTER	4 FILTER
Cost: 0 Bytes: 0 Cardinality:0	Cost: 0 Bytes: 0 Cardinality:0
Operation Changed 5 HASH JOIN SEMI	Operation Changed 5 NESTED LOOPS SEMI
Cost: 5,963 Bytes: 1,950 Cardinality:75	Cost: 23 Bytes: 260 Cardinality:10
Cost Changed 6 INDEX RANGE SCAN	Cost Changed 6 INDEX RANGE SCAN
Cost: 4 Bytes: 1,125 Cardinality:75	Cost: 3 Bytes: 150 Cardinality:10
Operation Changed 7 TABLE ACCESS FULL	Operation Changed 7 TABLE ACCESS BY INDEX ROWID

2. Review the Execution Plan

- EXPLAIN PLAN
  - Estimated plan - can be wrong for many reasons
    - Best Guess, Blind to Bind Variables or Data types
    - Explain Plan For … sql statement & DBMS_XPLAN.display
    - Set autotrace (on | trace | exp | stat | off)
- Tracing (all versions) / TKPROF
  - Get all sorts of good information
  - Works when you know a problem will occur
- V$SQL_PLAN (Oracle 9i+)
  - Actual execution plan
  - Use DBMS_XPLAN.display_cursor for display
- Historical Plans – AWR, Quest Foglight
  - Shows plan changes over time

How an Execution Plan is Created



Parsed Query (from Parser)

Query Transformer – rewrites query to be more efficient

Transformed Query

Estimator – looks at selectivity, cardinality & cost

Query + Estimates

Plan Generator – creates multiple plans using different access paths & join types. Plan with lowest cost is chosen

Default Plan sent to Row Source Generator to create execution plan

Data Dictionary

Schema Definition
&
Statistics

- OR Expansion
- View Merging
- Predicate Pushing
- Subquery Unnesting
- Query Rewrite with Materialized Views
- Star Transformation
- In-Memory Aggregation
- Table Expansion
- Join Factorization

Init.ora parameter to control behavior:
OPTIMIZER_FEATURES_ENABLED

Execution Plan Steps

- Show the sequence of operations performed to run SQL Statement
  - Order of the tables referenced in the statements
  - Access method for each table in the statement
    - INDEX
    - TABLE ACCESS
    - VIEW
  - Join method in statement accessing multiple tables
    - HASH JOIN
    - MERGE JOIN
    - NESTED LOOPS
  - Data manipulations
    - CONCATENATION
    - COUNT
    - FILTER
  - Statistic Collectors
    - New in 12C

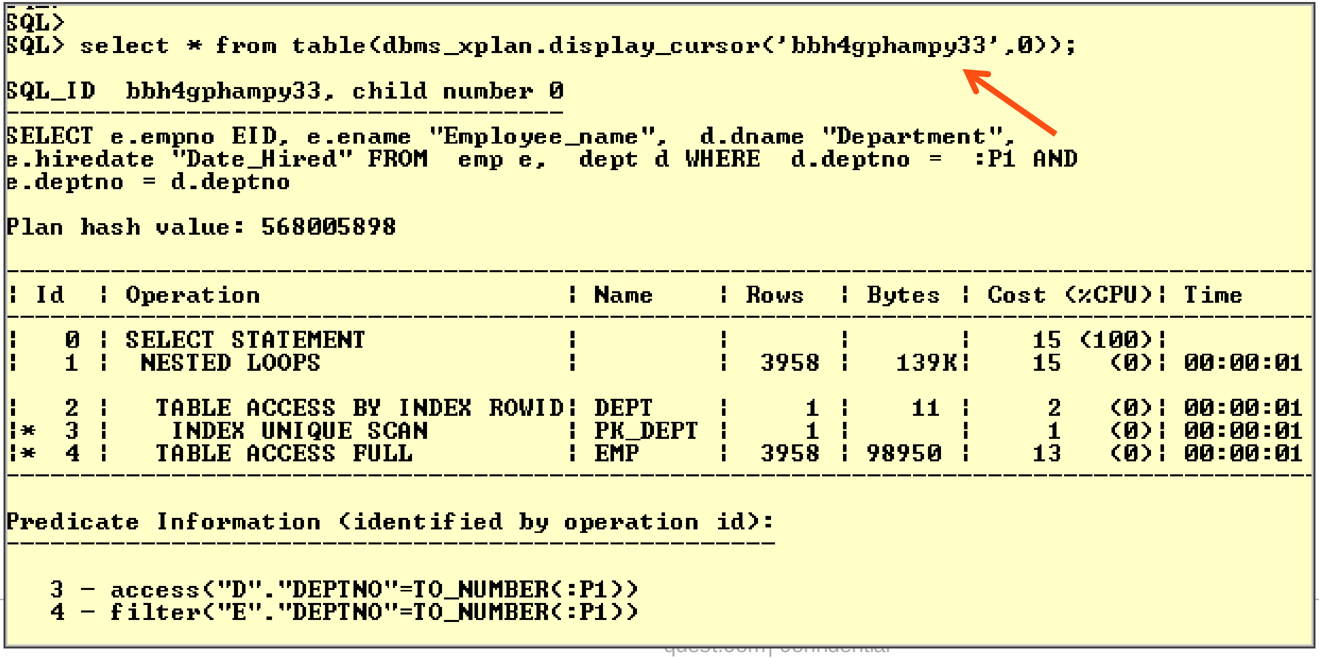
Examine the Execution Plan

- Find Expensive Operators
  - Examine cost, row counts and time of each step
  - Look for full table or index scans
- Review the Predicate Information
  - Know how bind variables are being interpreted
    - Review the data types
    - Implicit conversions
  - Know which step filtering predicate is applied
- Review the Join Methods
  - Nested Loops – good for large table / small table (lookup) joins
  - Hash Joins – good for large table / large table joins
- Check out the Notes Section
  - They are becoming increasingly important

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;

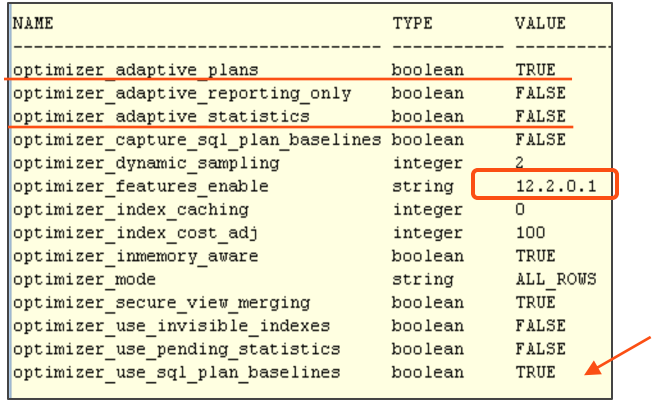
Actual Plan: V$SQL_PLAN using dbms_xplan.display_cursor



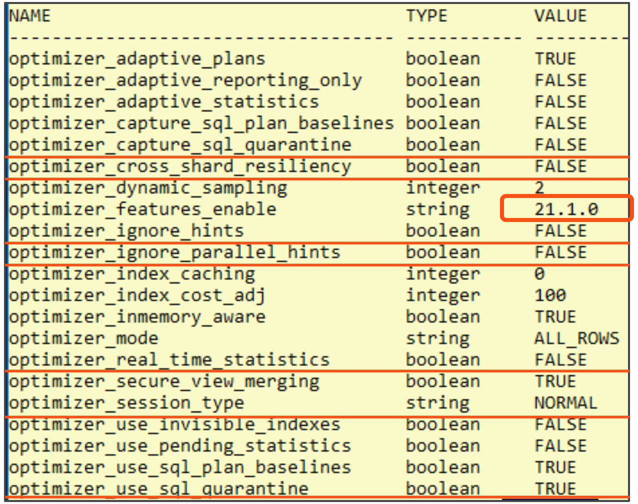
```
SQL>
SQL> select * from table(dbms_xplan.display_cursor('bbh4gphampy33',0));

SQL_ID  bbh4gphampy33, child number 0
-------------------------------------
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

Plan hash value: 568005898

---------------------------------------------------------------------------------------
| Id  | Operation                    | Name    | Rows  | Bytes | Cost (%CPU)| Time     |
---------------------------------------------------------------------------------------
|   0 | SELECT STATEMENT             |         |       |       |    15 (100)|          |
|   1 |  NESTED LOOPS                |         |  3958 |  139K|    15   (0)| 00:00:01 |

|   2 |   TABLE ACCESS BY INDEX ROWID| DEPT    |     1 |    11 |     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))
```

Know Which Optimizer Features You are Using

- Show parameter optimizer

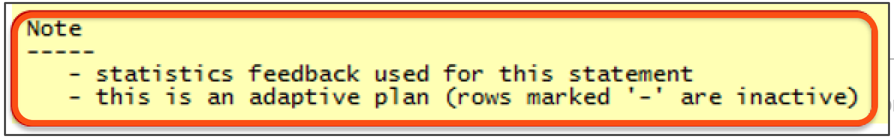


```
NAME                                 TYPE        VALUE
------------------------------------ ----------- ---------
optimizer_adaptive_plans             boolean     TRUE
optimizer_adaptive_reporting_only    boolean     FALSE
optimizer_adaptive_statistics        boolean     FALSE
optimizer_capture_sql_plan_baselines boolean     FALSE
optimizer_dynamic_sampling           integer     2
optimizer_features_enable            string      12.2.0.1
optimizer_index_caching              integer     0
optimizer_index_cost_adj             integer     100
optimizer_inmemory_aware             boolean     TRUE
optimizer_mode                       string      ALL_ROWS
optimizer_secure_view_merging        boolean     TRUE
optimizer_use_invisible_indexes      boolean     FALSE
optimizer_use_pending_statistics     boolean     FALSE
optimizer_use_sql_plan_baselines     boolean     TRUE
```

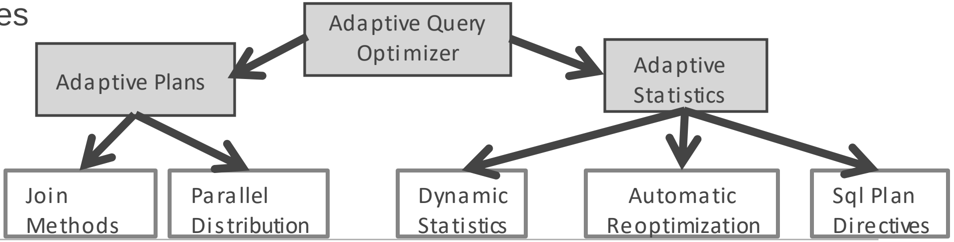


```
NAME                                 TYPE        VALUE
------------------------------------ ----------- --------
optimizer_adaptive_plans             boolean     TRUE
optimizer_adaptive_reporting_only    boolean     FALSE
optimizer_adaptive_statistics        boolean     FALSE
optimizer_capture_sql_plan_baselines boolean     FALSE
optimizer_capture_sql_quarantine     boolean     FALSE
optimizer_cross_shard_resiliency     boolean     FALSE
optimizer_dynamic_sampling           integer     2
optimizer_features_enable            string      21.1.0
optimizer_ignore_hints               boolean     FALSE
optimizer_ignore_parallel_hints      boolean     FALSE
optimizer_index_caching              integer     0
optimizer_index_cost_adj             integer     100
optimizer_inmemory_aware             boolean     TRUE
optimizer_mode                       string      ALL_ROWS
optimizer_real_time_statistics      boolean     FALSE
optimizer_secure_view_merging        boolean     TRUE
optimizer_session_type               string      NORMAL
optimizer_use_invisible_indexes      boolean     FALSE
optimizer_use_pending_statistics     boolean     FALSE
optimizer_use_sql_plan_baselines     boolean     TRUE
optimizer_use_sql_quarantine         boolean     TRUE
```

- What is supporting the Execution Plan
  - SQL Plan Management (Baselines) / Profiles / Outlines / Patches
  - Dynamic Statistics, Statistics Feedback or SQL Directives
  - Adaptive Cursor Sharing
  - Adaptive Plans
- Notes Section gives you clues

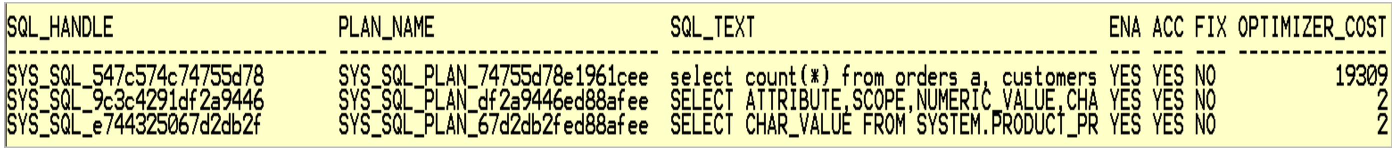


```
Note
-----
   - statistics feedback used for this statement
   - this is an adaptive plan (rows marked '-' are inactive)
```

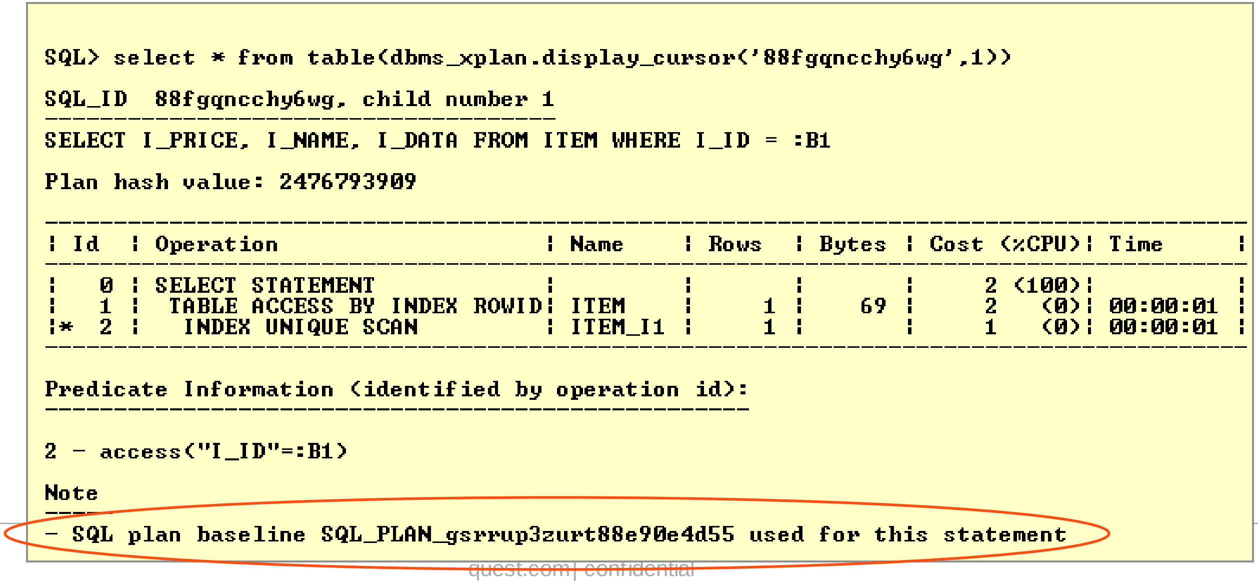


Execution Plan using Optimizer Feature: SPM (baselines)

- Select * from dba_sql_plan_baselines



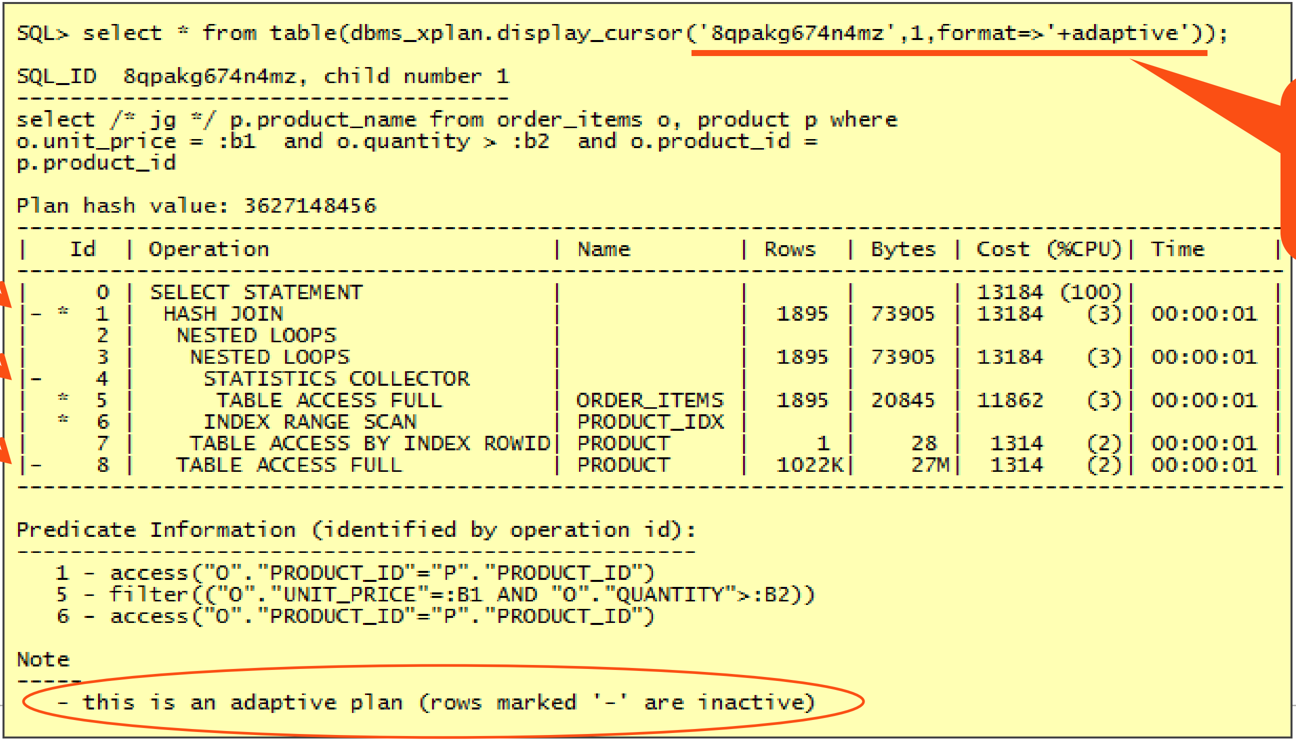
```
SQL_HANDLE                     PLAN_NAME                      SQL_TEXT                                 ENA ACC FIX OPTIMIZER_COST
------------------------------ ------------------------------ ---------------------------------------- --- --- --- --------------
SYS_SQL_547c574c74755d78       SYS_SQL_PLAN_74755d78e1961cee  select count(*) from orders a, customers YES YES NO           19309
SYS_SQL_9c3c4291df2a9446       SYS_SQL_PLAN_df2a9446ed88afee  SELECT ATTRIBUTE,SCOPE,NUMERIC_VALUE,CHA YES YES NO               2
SYS_SQL_e744325067d2db2f       SYS_SQL_PLAN_67d2db2fed88afee  SELECT CHAR_VALUE FROM SYSTEM.PRODUCT_PR YES YES NO               2
```



```
SQL> select * from table(dbms_xplan.display_cursor('88fgqncchy6wg',1))

SQL_ID  88fgqncchy6wg, child number 1
-------------------------------------
SELECT I_PRICE, I_NAME, I_DATA FROM ITEM WHERE I_ID = :B1

Plan hash value: 2476793909

-------------------------------------------------------------------------------------------
| Id  | Operation                   | Name     | Rows  | Bytes | Cost (%CPU)| Time     |
-------------------------------------------------------------------------------------------
|   0 | SELECT STATEMENT            |          |       |       |     2 (100)|          |
|   1 |  TABLE ACCESS BY INDEX ROWID| ITEM     |     1 |    69 |     2   (0)| 00:00:01 |
|*  2 |   INDEX UNIQUE SCAN         | ITEM_I1  |     1 |       |     1   (0)| 00:00:01 |
-------------------------------------------------------------------------------------------

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

2 - access("I_ID"=:B1)

Note
-----
- SQL plan baseline SQL_PLAN_gsrrup3zurt88e90e4d55 used for this statement
```

Adaptive Plan example

- Adapted on first execution

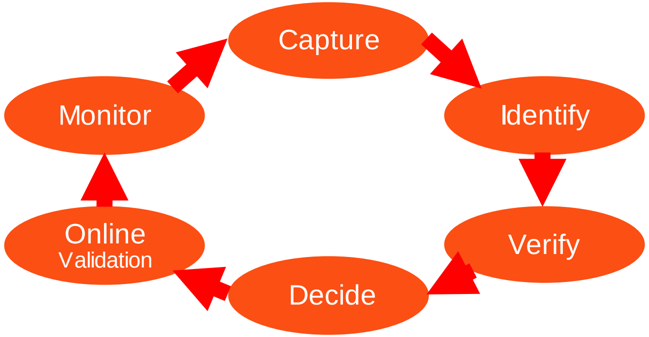


```
SQL> select * from table(dbms_xplan.display_cursor('8qpakg674n4mz',1,format=>'+adaptive'));

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

-----------------------------------------------------------------------------------------------
|   Id  | Operation                     | Name          | Rows  | Bytes | Cost (%CPU)| Time     |
-----------------------------------------------------------------------------------------------
|     0 | SELECT STATEMENT              |               |       |       | 13184 (100)|          |
|- *  1 |  HASH JOIN                    |               |  1895 | 73905 | 13184   (3)| 00:00:01 |
|     2 |   NESTED LOOPS                |               |       |       |            |          |
|     3 |    NESTED LOOPS               |               |  1895 | 73905 | 13184   (3)| 00:00:01 |
|-    4 |     STATISTICS COLLECTOR      |               |       |       |            |          |
|  *  5 |      TABLE ACCESS FULL        | ORDER_ITEMS   |  1895 | 20845 | 11862   (3)| 00:00:01 |
|  *  6 |     INDEX RANGE SCAN          | PRODUCT_IDX   |       |       |            |          |
|     7 |    TABLE ACCESS BY INDEX ROWID| PRODUCT       |     1 |    28 |  1314   (2)| 00:00:01 |
|-    8 |   TABLE ACCESS FULL           | PRODUCT       |  1022K|   27M |  1314   (2)| 00:00:01 |
-----------------------------------------------------------------------------------------------

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
-----
   - this is an adaptive plan (rows marked '-' are inactive)
```

New format options for dbms_xplan are: '+adaptive' – inactive steps '+report' – reporting_only

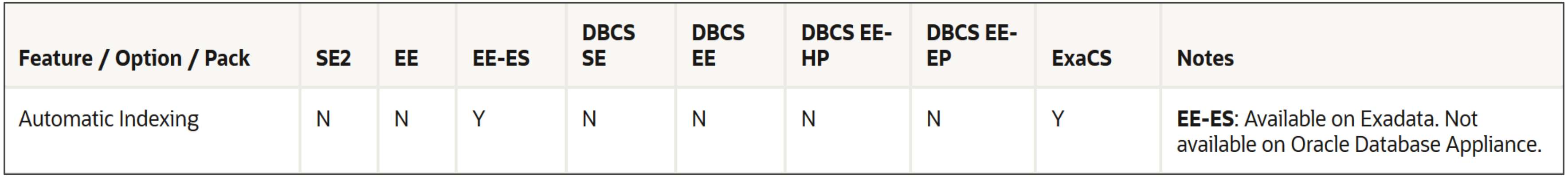
19c Automatic Indexing – What is it?

- Implements indexes based expert index tuning knowledge
  - Identifies ‘candidate indexes’ based on table column usage
  - Without DBA involvement
    - Except for DBA can set preferences
      - View report of indexes and their impact on the application
- Works incrementally
  - Needs to be iterative and continuous
  - Created as invisible
    - Uses ‘SYS_AI’ as the name prefix
  - Automatic indexes are tested
    - If improved performance – indexes made visible
    - If no improvement – indexes are marked unusable
      - Later removed



19c Automatic Indexing Requirements

- Feature is only available to Enterprise Edition on Engineered Systems
  - Exadata only



Feature / Option / Pack	SE2	EE	EE-ES	DBCS SE	DBCS EE	DBCS EE-HP	DBCS EE-EP	ExaCS	Notes
Automatic Indexing	N	N	Y	N	N	N	N	Y	**EE-ES**: Available on Exadata. Not available on Oracle Database Appliance.

  - Workaround for testing / development
    - ○ In CDB as sysdba

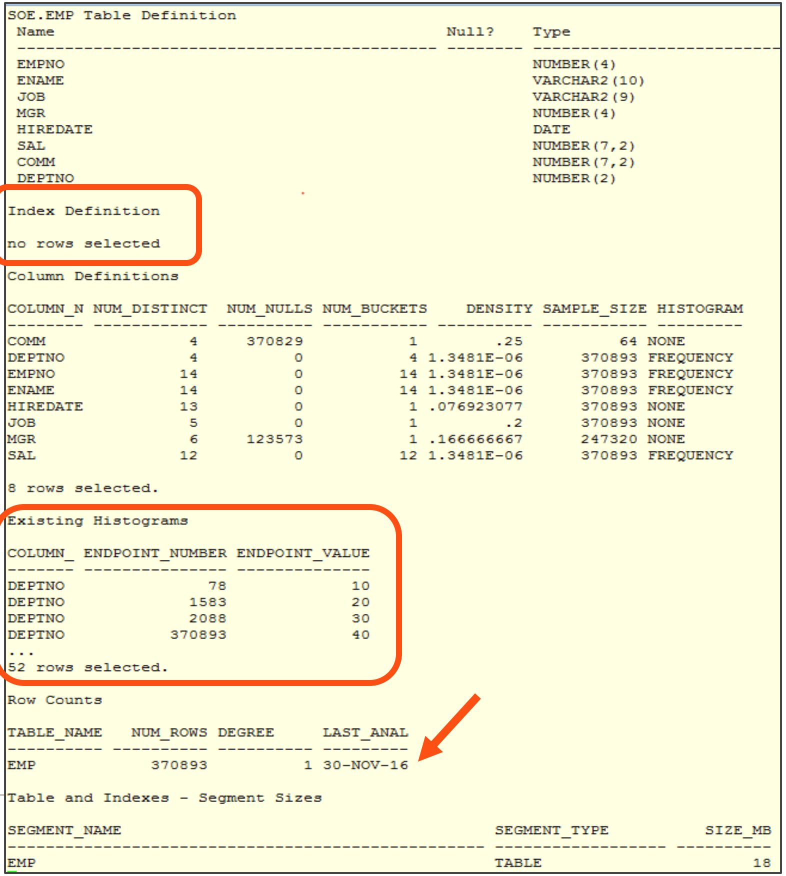
      Alter system set "_Exadata_feature_on"=true scope=spfile;

      Shutdown immediate;

      Startup
  - Unfortunately, this is not supported
    - ○ Don't use on real system

3. Gather Object Information

- Understand objects in execution plans
  - Table Definitions & Segment sizes
    - Is it a View?
      - Get underlying definition
    - Number of Rows / Partitioning
  - Examine Columns in Where Clause
    - Cardinality of columns
    - Data Skew / Histograms
  - Statistic Gathering
    - Tip: Out-of-date statistics can impact performance
- See tuning.sql script in appendix
  - Run it for expensive data access targets

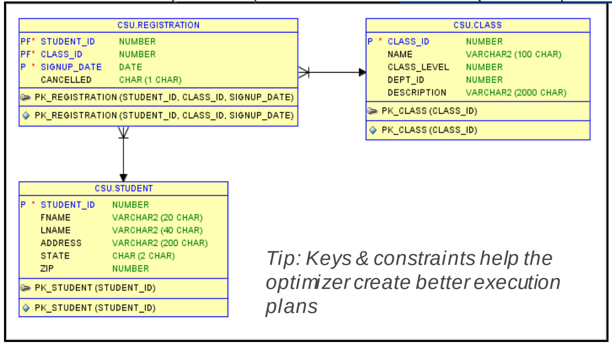


```
SOE.EMP Table Definition
 Name                                          Null?    Type
 -------------------------------------------- -------- ---------------------------
 EMPNO                                                  NUMBER(4)
 ENAME                                                  VARCHAR2(10)
 JOB                                                    VARCHAR2(9)
 MGR                                                    NUMBER(4)
 HIREDATE                                               DATE
 SAL                                                    NUMBER(7,2)
 COMM                                                   NUMBER(7,2)
 DEPTNO                                                 NUMBER(2)

Index Definition

no rows selected

Column Definitions

COLUMN_N NUM_DISTINCT  NUM_NULLS NUM_BUCKETS    DENSITY SAMPLE_SIZE HISTOGRAM
-------- ------------ ---------- ----------- ---------- ----------- ---------
COMM                4     370829           1        .25          64 NONE
DEPTNO              4          0           4 1.3481E-06      370893 FREQUENCY
EMPNO              14          0          14 1.3481E-06      370893 FREQUENCY
ENAME              14          0          14 1.3481E-06      370893 FREQUENCY
HIREDATE           13          0           1 .076923077      370893 NONE
JOB                 5          0           1         .2      370893 NONE
MGR                 6     123573           1 .166666667      247320 NONE
SAL                12          0          12 1.3481E-06      370893 FREQUENCY

8 rows selected.

Existing Histograms

COLUMN_ ENDPOINT_NUMBER ENDPOINT_VALUE
------- --------------- --------------
DEPTNO               78             10
DEPTNO             1583             20
DEPTNO             2088             30
DEPTNO           370893             40
...
52 rows selected.

Row Counts

TABLE_NAME   NUM_ROWS DEGREE     LAST_ANAL
---------- ---------- ---------- ---------
EMP            370893          1 30-NOV-16

Table and Indexes - Segment Sizes

SEGMENT_NAME                                      SEGMENT_TYPE          SIZE_MB
------------------------------------------------- ------------------ ----------
EMP                                               TABLE                      18
```

Review Indexes & Constraints

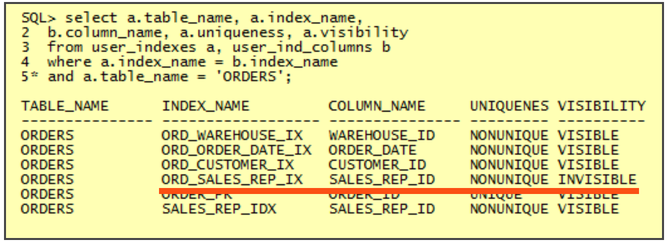
- Get Index definitions
  - Know the order of columns and their selectivity
- Review existing keys and constraints
  - Know Multi-Table Relationships (ERD)
    - Primary key and foreign definitions
  - Check and not null constraints
- Make sure the optimizer can use the index
  - Functions on indexed columns can turn off index
    - Consider a function index
  - Look for implicit conversions
    - Get sample bind variable values
  - Is the index INVISIBLE?

FREE - Oracle SQL Dev eloper Data Modeler: Oracle SQL Dev eloper Data Modeler



Tip: Keys & constraints help the optimizer create better execution plans

```
SELECT name, position, datatype_string, value_string
FROM v$sql_bind_capture
WHERE sql_id = '0zz5h1003f2dw';
```



```
SQL> select a.table_name, a.index_name,
2  b.column_name, a.uniqueness, a.visibility
3  from user_indexes a, user_ind_columns b
4  where a.index_name = b.index_name
5* and a.table_name = 'ORDERS';

TABLE_NAME      INDEX_NAME         COLUMN_NAME      UNIQUENES VISIBILITY
--------------- ------------------ ---------------- --------- ----------
ORDERS          ORD_WAREHOUSE_IX   WAREHOUSE_ID     NONUNIQUE VISIBLE
ORDERS          ORD_ORDER_DATE_IX  ORDER_DATE       NONUNIQUE VISIBLE
ORDERS          ORD_CUSTOMER_IX    CUSTOMER_ID      NONUNIQUE VISIBLE
ORDERS          ORD_SALES_REP_IX   SALES_REP_ID     NONUNIQUE INVISIBLE
ORDERS          ORDER_PK           ORDER_ID         UNIQUE    VISIBLE
ORDERS          SALES_REP_IDX      SALES_REP_ID     NONUNIQUE VISIBLE
```

Understand Statistics gathering

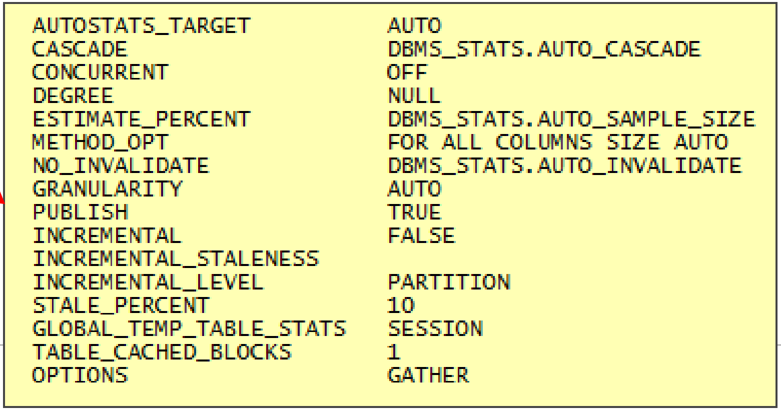
- GATHER_*_STATS procedures have many parameters
  - Should only set 2-4 parameters (per Tom Kyte)
    - SCHEMA NAME
    - TABLE NAME
    - PARTITION NAME
    - DOP
  - Defaults for: exec dbms_stats.gather_schema_stats(‘SOE’);

DBMS_STATS package

- Rewritten in 11g
  - A Faster & better AUTO_SAMPLE_SIZE
  - 100% in less time & more accurate than 10% estimate
- Avoid using ESTIMATE_PERCENT

New GET_PREFS function

select dbms_stats.get_prefs('ESTIMATE_PERCENT') from dual;



```
AUTOSTATS_TARGET             AUTO
CASCADE                      DBMS_STATS.AUTO_CASCADE
CONCURRENT                   OFF
DEGREE                       NULL
ESTIMATE_PERCENT             DBMS_STATS.AUTO_SAMPLE_SIZE
METHOD_OPT                   FOR ALL COLUMNS SIZE AUTO
NO_INVALIDATE                DBMS_STATS.AUTO_INVALIDATE
GRANULARITY                  AUTO
PUBLISH                      TRUE
INCREMENTAL                  FALSE
INCREMENTAL_STALENESS
INCREMENTAL_LEVEL            PARTITION
STALE_PERCENT                10
GLOBAL_TEMP_TABLE_STATS      SESSION
TABLE_CACHED_BLOCKS          1
OPTIONS                      GATHER
```

Optimizer tries to fix Statistics Mistakes

- Dynamic Statistics
  - Missing, Insufficient, Stale Statistics or Parallel Execution
  - New level 11 in 12c
    - alter session set OPTIMIZER_DYNAMIC_SAMPLING = 11;
- Statistics Feedback
  - Collectors sample statistics on 1st execution
    - Default stats compared with actual rows sampled
    - If they differ significantly, optimizer stores correct estimates for future use
      - Stored in OPT_ESTIMATE hints in V$SQL_REOPTIMIZATION_HINTS
- SQL Plan Directives
  - Additional info 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

What wrong with these pictures?

4. Find the Driving table

- Need to know the size of the actual data sets in each step
  - In Joins (Right, Left, Outer)
  - What are the filtering predicates
  - When is each filtering predicate applied
    - Try to filter earlier rather than later
- Compare size of final result set with # of data reads
- Find the driving table
    - To reduce buffer gets

```
SELECT s.fname, s.lname, r.signup_date
FROM   student s
   INNER JOIN registration r ON s.student_id = r.student_id
   INNER JOIN class c ON r.class_id = c.class_id
WHERE  c.name = 'SQL TUNING'
AND    r.signup_date BETWEEN :beg_date AND :beg_date +1
AND    r.cancelled = 'N'
```

Joins

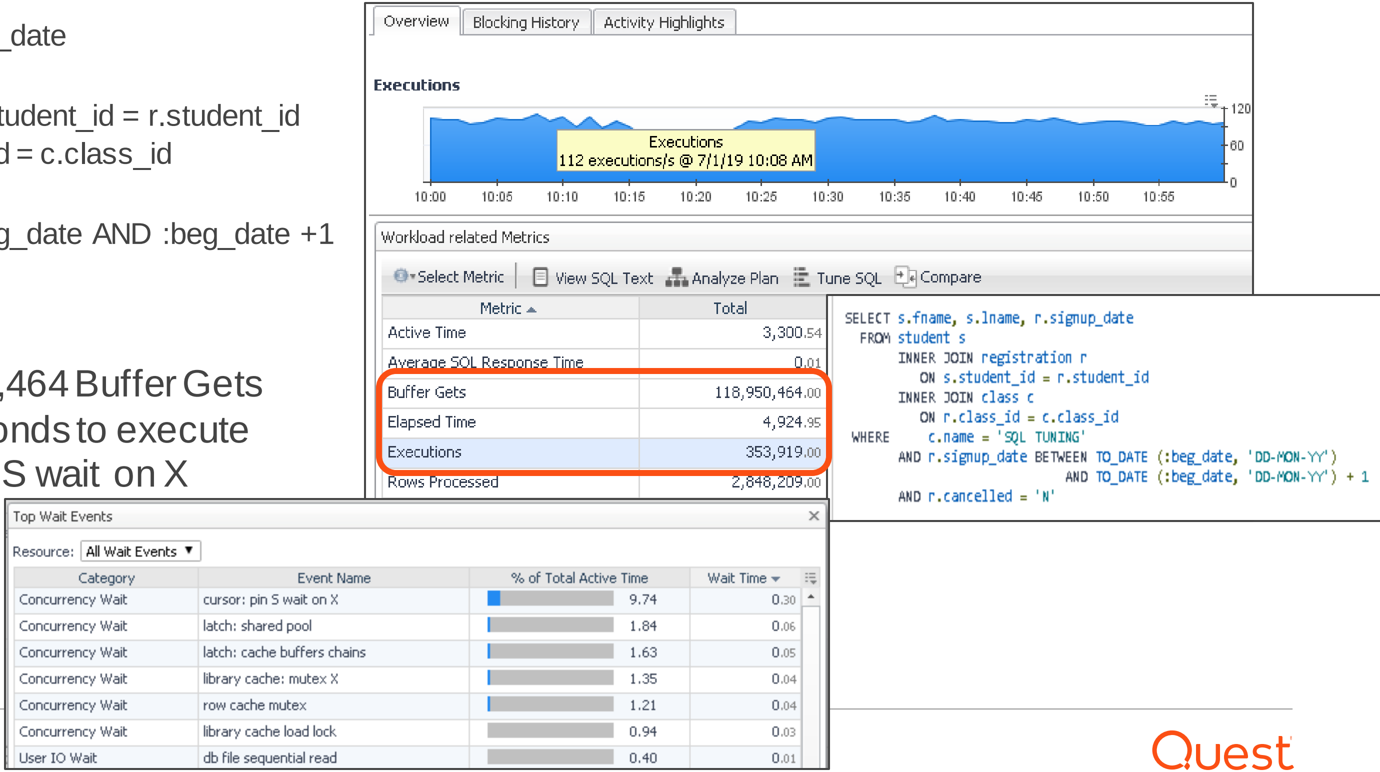
Filtering Predicates

Case Study

- Who registered yesterday for SQL Tuning?

```
SELECT s.fname, s.lname, r.signup_date
FROM   student s
   INNER  JOIN registration r ON s.student_id = r.student_id
   INNER  JOIN class c ON r.class_id = c.class_id
WHERE  c.name  = 'SQL TUNING'
AND    r.signup_date BETWEEN :beg_date AND :beg_date +1
AND    r.cancelled = ‘N‘
```

Execution Stats – 118,950,464 Buffer Gets
Execution Time – .01 seconds to execute
Wait Events – cursor: pin S wait on X
CPU – 57.46%



Metric ▲	Total
Active Time	3,300.54
Average SQL Response Time	0.01
Buffer Gets	118,950,464.00
Elapsed Time	4,924.95
Executions	353,919.00
Rows Processed	2,848,209.00

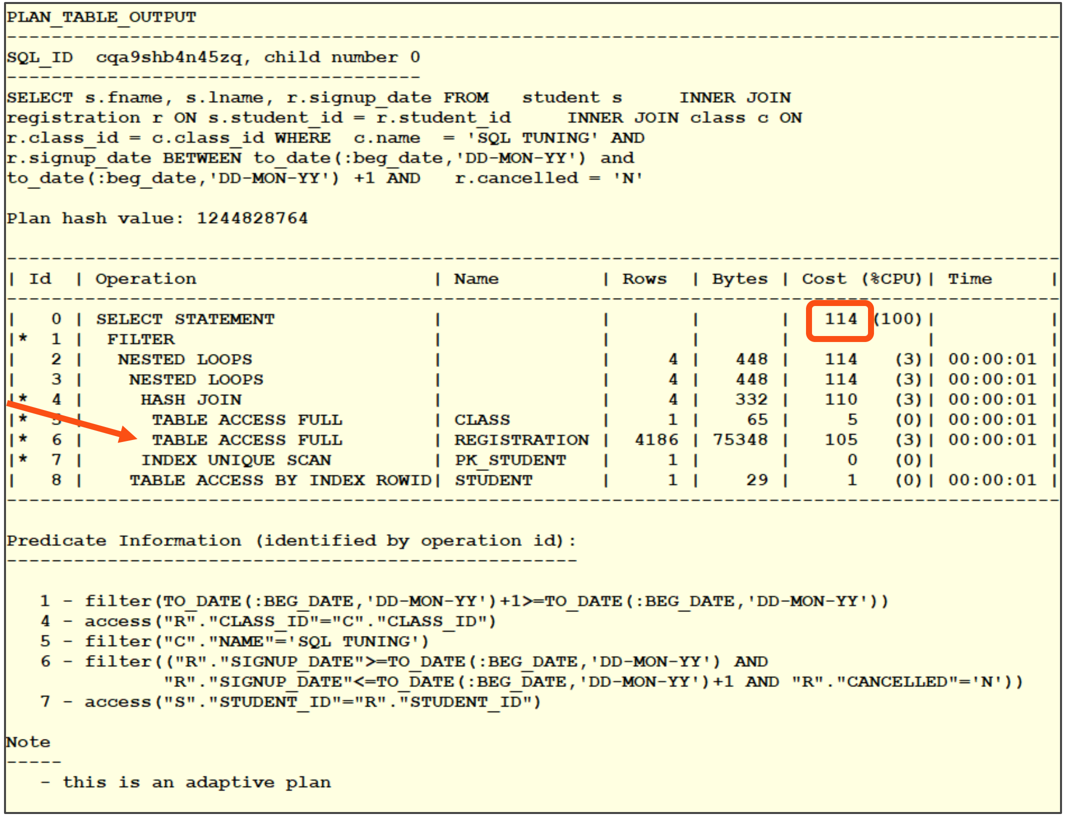
```
SELECT s.fname, s.lname, r.signup_date
  FROM student s
       INNER JOIN registration r
          ON s.student_id = r.student_id
       INNER JOIN class c
          ON r.class_id = c.class_id
 WHERE     c.name = 'SQL TUNING'
       AND r.signup_date BETWEEN TO_DATE (:beg_date, 'DD-MON-YY')
                             AND TO_DATE (:beg_date, 'DD-MON-YY') + 1
       AND r.cancelled = 'N'
```

Top Wait Events

Resource: All Wait Events ▼

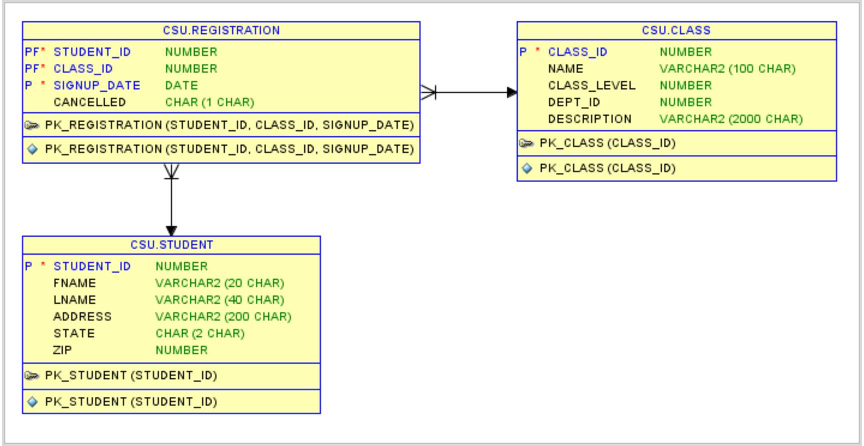
Category	Event Name	% of Total Active Time	Wait Time ▼
Concurrency Wait	cursor: pin S wait on X	9.74	0.30
Concurrency Wait	latch: shared pool	1.84	0.06
Concurrency Wait	latch: cache buffers chains	1.63	0.05
Concurrency Wait	library cache: mutex X	1.35	0.04
Concurrency Wait	row cache mutex	1.21	0.04
Concurrency Wait	library cache load lock	0.94	0.03
User IO Wait	db file sequential read	0.40	0.01

Execution Plan



```
PLAN_TABLE_OUTPUT
------------------------------------------------------------------------------------------------
SQL_ID  cqa9shb4n45zq, child number 0
-------------------------------------
SELECT s.fname, s.lname, r.signup_date FROM   student s     INNER JOIN
registration r ON s.student_id = r.student_id     INNER JOIN class c ON
r.class_id = c.class_id WHERE  c.name  = 'SQL TUNING' AND
r.signup_date BETWEEN to_date(:beg_date,'DD-MON-YY') and
to_date(:beg_date,'DD-MON-YY') +1 AND   r.cancelled = 'N'

Plan hash value: 1244828764

------------------------------------------------------------------------------------------------
| Id  | Operation                    | Name           | Rows  | Bytes | Cost (%CPU)| Time     |
------------------------------------------------------------------------------------------------
|   0 | SELECT STATEMENT             |                |       |       |   114 (100)|          |
|*  1 |  FILTER                      |                |       |       |            |          |
|   2 |   NESTED LOOPS               |                |     4 |   448 |   114   (3)| 00:00:01 |
|   3 |    NESTED LOOPS              |                |     4 |   448 |   114   (3)| 00:00:01 |
|*  4 |     HASH JOIN                |                |     4 |   332 |   110   (3)| 00:00:01 |
|*  5 |      TABLE ACCESS FULL       | CLASS          |     1 |    65 |     5   (0)| 00:00:01 |
|*  6 |      TABLE ACCESS FULL       | REGISTRATION   |  4186 | 75348 |   105   (3)| 00:00:01 |
|*  7 |     INDEX UNIQUE SCAN        | PK_STUDENT     |     1 |       |     0   (0)|          |
|   8 |    TABLE ACCESS BY INDEX ROWID| STUDENT       |     1 |    29 |     1   (0)| 00:00:01 |
------------------------------------------------------------------------------------------------

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

   1 - filter(TO_DATE(:BEG_DATE,'DD-MON-YY')+1>=TO_DATE(:BEG_DATE,'DD-MON-YY'))
   4 - access("R"."CLASS_ID"="C"."CLASS_ID")
   5 - filter("C"."NAME"='SQL TUNING')
   6 - filter(("R"."SIGNUP_DATE">=TO_DATE(:BEG_DATE,'DD-MON-YY') AND
              "R"."SIGNUP_DATE"<=TO_DATE(:BEG_DATE,'DD-MON-YY')+1 AND "R"."CANCELLED"='N'))
   7 - access("S"."STUDENT_ID"="R"."STUDENT_ID")

Note
-----
   - this is an adaptive plan
```

Relationship Diagram



CSU.REGISTRATION

PF* STUDENT_ID NUMBER
PF* CLASS_ID NUMBER
P * SIGNUP_DATE DATE
CANCELLED CHAR (1 CHAR)
PK_REGISTRATION (STUDENT_ID, CLASS_ID, SIGNUP_DATE)
PK_REGISTRATION (STUDENT_ID, CLASS_ID, SIGNUP_DATE)

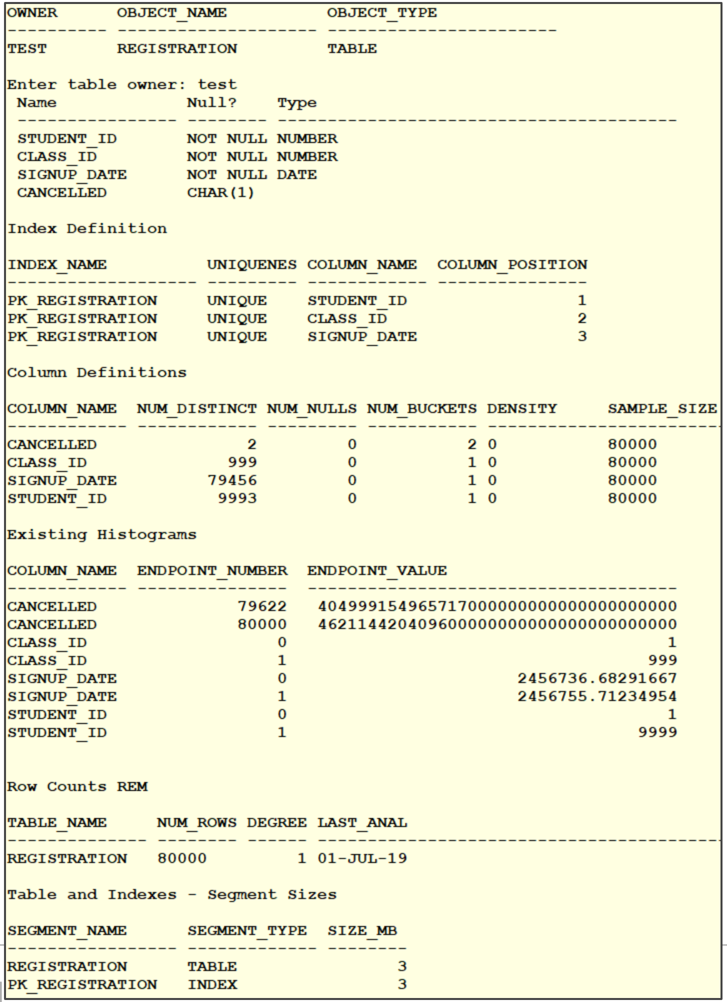
CSU.CLASS

P * CLASS_ID NUMBER
NAME VARCHAR2 (100 CHAR)
CLASS_LEVEL NUMBER
DEPT_ID NUMBER
DESCRIPTION VARCHAR2 (2000 CHAR)
PK_CLASS (CLASS_ID)
PK_CLASS (CLASS_ID)

CSU.STUDENT

P * STUDENT_ID NUMBER
FNAME VARCHAR2 (20 CHAR)
LNAME VARCHAR2 (40 CHAR)
ADDRESS VARCHAR2 (200 CHAR)
STATE CHAR (2 CHAR)
ZIP NUMBER
PK_STUDENT (STUDENT_ID)
PK_STUDENT (STUDENT_ID)

- Registration – 80,000
- Student – 10,000
- Class – 1,000

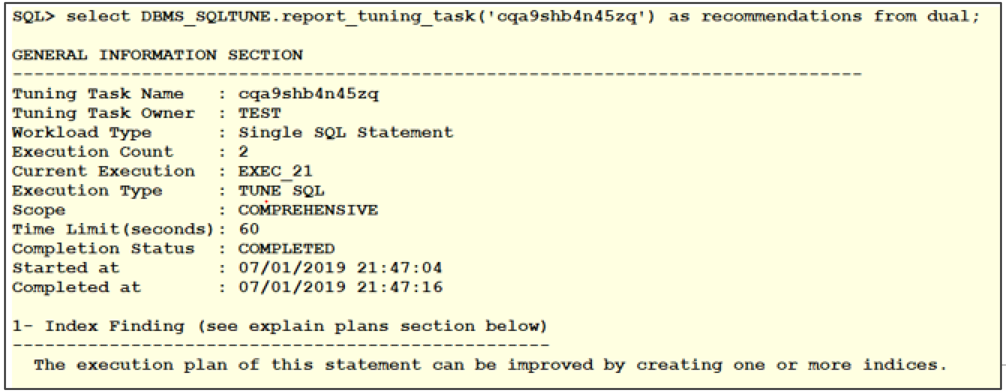


```
OWNER      OBJECT_NAME          OBJECT_TYPE
---------- -------------------- -----------------------
TEST       REGISTRATION         TABLE

Enter table owner: test
 Name                             Null?    Type
 -------------------------------- -------- ----------------------------------------
 STUDENT_ID                       NOT NULL NUMBER
 CLASS_ID                         NOT NULL NUMBER
 SIGNUP_DATE                      NOT NULL DATE
 CANCELLED                                 CHAR(1)

Index Definition

INDEX_NAME          UNIQUENES COLUMN_NAME  COLUMN_POSITION
------------------- --------- ------------ ---------------
PK_REGISTRATION     UNIQUE    STUDENT_ID                 1
PK_REGISTRATION     UNIQUE    CLASS_ID                   2
PK_REGISTRATION     UNIQUE    SIGNUP_DATE                3

Column Definitions

COLUMN_NAME  NUM_DISTINCT NUM_NULLS NUM_BUCKETS DENSITY     SAMPLE_SIZE
------------ ------------ --------- ----------- ------------------------
CANCELLED               2         0           2 0           80000
CLASS_ID              999         0           1 0           80000
SIGNUP_DATE         79456         0           1 0           80000
STUDENT_ID           9993         0           1 0           80000

Existing Histograms

COLUMN_NAME  ENDPOINT_NUMBER  ENDPOINT_VALUE
------------ ---------------  --------------------------------------
CANCELLED              79622   4049991549657170000000000000000000000
CANCELLED              80000   4621144204096000000000000000000000000
CLASS_ID                   0                                       1
CLASS_ID                   1                                     999
SIGNUP_DATE                0                        2456736.68291667
SIGNUP_DATE                1                        2456755.71234954
STUDENT_ID                 0                                       1
STUDENT_ID                 1                                    9999


Row Counts REM

TABLE_NAME     NUM_ROWS DEGREE LAST_ANAL
-------------- -------- ------ ----------------------------------------
REGISTRATION   80000         1 01-JUL-19

Table and Indexes - Segment Sizes

SEGMENT_NAME      SEGMENT_TYPE  SIZE_MB
----------------- ------------- --------
REGISTRATION      TABLE                3
PK_REGISTRATION   INDEX                3
```

Tuning Advisor

- Recommends – 2 new indexes

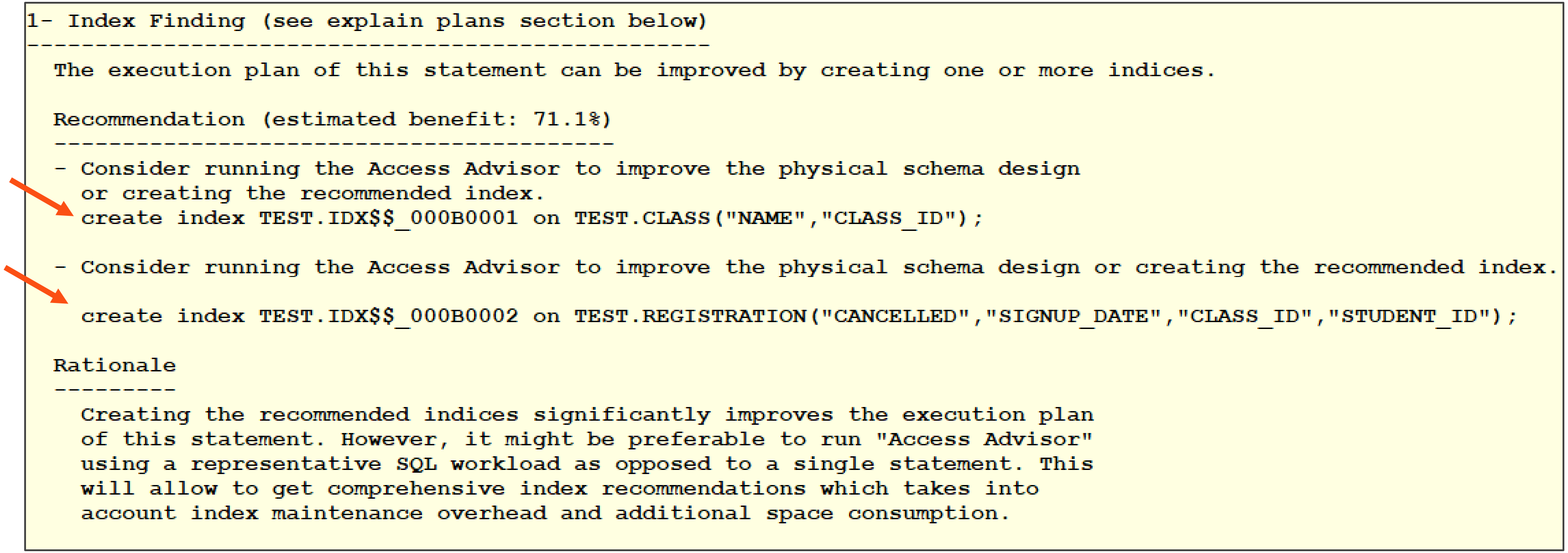
```
DECLARE
 l_sql_tune_task_id  VARCHAR2(100);
BEGIN
 l_sql_tune_task_id := DBMS_SQLTUNE.create_tuning_task ( sql_id => '&sql_id',
  scope => DBMS_SQLTUNE.scope_comprehensive, time_limit  => 60,
  task_name => '&sql_id', description => 'Tuning task for class registration query');
 DBMS_OUTPUT.put_line('l_sql_tune_task_id: ' || l_sql_tune_task_id);
END;
/

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



```
SQL> select DBMS_SQLTUNE.report_tuning_task('cqa9shb4n45zq') as recommendations from dual;

GENERAL INFORMATION SECTION
-------------------------------------------------------------------------------
Tuning Task Name   : cqa9shb4n45zq
Tuning Task Owner  : TEST
Workload Type      : Single SQL Statement
Execution Count    : 2
Current Execution  : EXEC_21
Execution Type     : TUNE SQL
Scope              : COMPREHENSIVE
Time Limit(seconds): 60
Completion Status  : COMPLETED
Started at         : 07/01/2019 21:47:04
Completed at       : 07/01/2019 21:47:16

1- Index Finding (see explain plans section below)
--------------------------------------------------
  The execution plan of this statement can be improved by creating one or more indices.
```

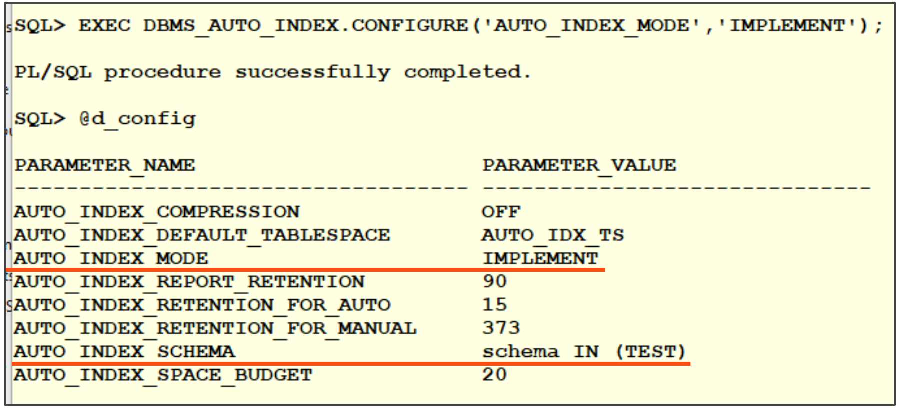
Tuning Advisor

- Recommends – 2 new indexes
  - Select DBMS_SQLTUNE.report_tuning_task(‘&task_name’) from dual;

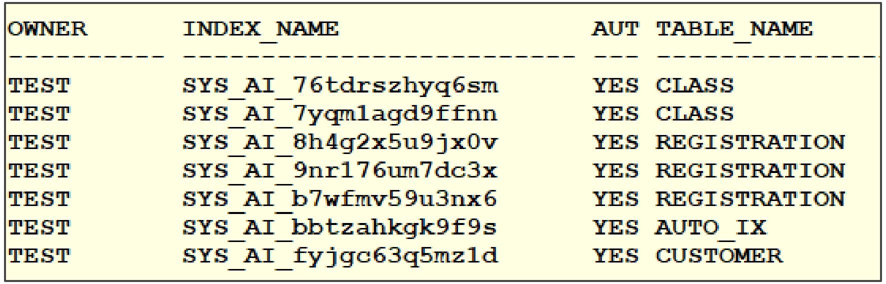


```
1- Index Finding (see explain plans section below)
--------------------------------------------------
  The execution plan of this statement can be improved by creating one or more indices.

  Recommendation (estimated benefit: 71.1%)
  -----------------------------------------
  - Consider running the Access Advisor to improve the physical schema design
    or creating the recommended index.
    create index TEST.IDX$$_000B0001 on TEST.CLASS("NAME","CLASS_ID");

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

    create index TEST.IDX$$_000B0002 on TEST.REGISTRATION("CANCELLED","SIGNUP_DATE","CLASS_ID","STUDENT_ID");

  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.
```

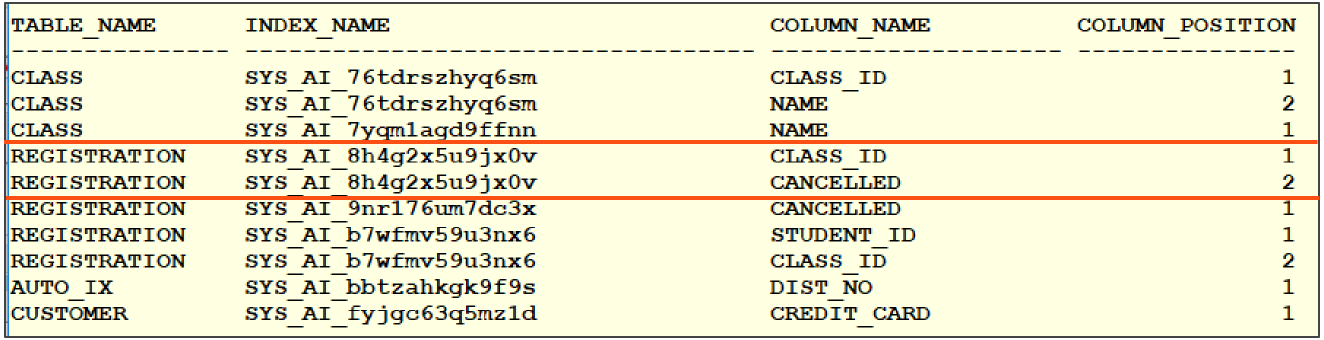
19c Automatic Indexes Enabled for Schema ‘Test’



```
SQL> EXEC DBMS_AUTO_INDEX.CONFIGURE('AUTO_INDEX_MODE','IMPLEMENT');

PL/SQL procedure successfully completed.

SQL> @d_config

PARAMETER_NAME                      PARAMETER_VALUE
----------------------------------- ------------------------------
AUTO_INDEX_COMPRESSION              OFF
AUTO_INDEX_DEFAULT_TABLESPACE       AUTO_IDX_TS
AUTO_INDEX_MODE                     IMPLEMENT
AUTO_INDEX_REPORT_RETENTION         90
AUTO_INDEX_RETENTION_FOR_AUTO       15
AUTO_INDEX_RETENTION_FOR_MANUAL     373
AUTO_INDEX_SCHEMA                   schema IN (TEST)
AUTO_INDEX_SPACE_BUDGET             20
```



```
OWNER        INDEX_NAME                     AUT TABLE_NAME
------------ ------------------------------ --- --------------
TEST         SYS_AI_76tdrszhyq6sm           YES CLASS
TEST         SYS_AI_7yqmlagd9ffnn           YES CLASS
TEST         SYS_AI_8h4g2x5u9jx0v           YES REGISTRATION
TEST         SYS_AI_9nr176um7dc3x           YES REGISTRATION
TEST         SYS_AI_b7wfmv59u3nx6           YES REGISTRATION
TEST         SYS_AI_bbtzahkgk9f9s           YES AUTO_IX
TEST         SYS_AI_fyjgc63q5mz1d           YES CUSTOMER
```

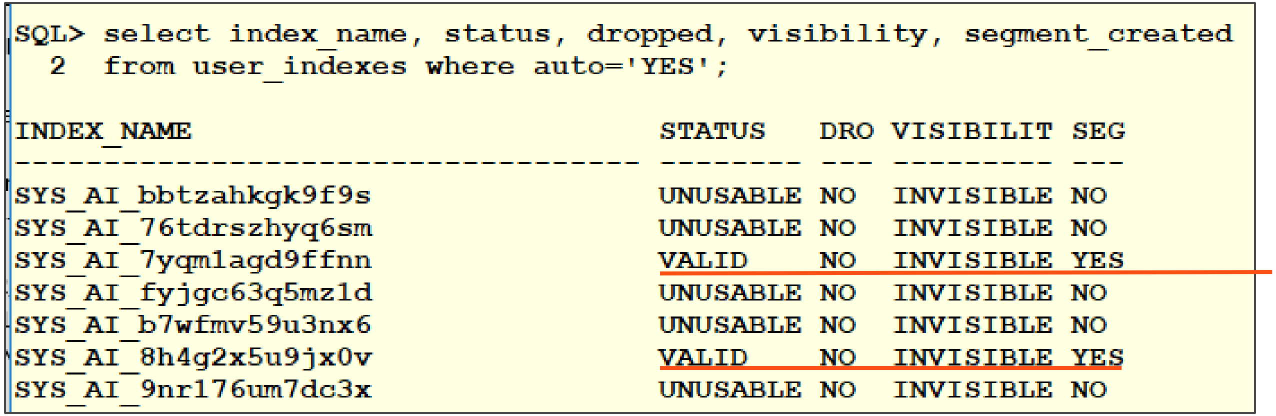


```
TABLE_NAME       INDEX_NAME                               COLUMN_NAME           COLUMN_POSITION
---------------- ---------------------------------------- --------------------- ---------------
CLASS            SYS_AI_76tdrszhyq6sm                     CLASS_ID                            1
CLASS            SYS_AI_76tdrszhyq6sm                     NAME                                2
CLASS            SYS_AI_7yqmlagd9ffnn                     NAME                                1
REGISTRATION     SYS_AI_8h4g2x5u9jx0v                     CLASS_ID                            1
REGISTRATION     SYS_AI_8h4g2x5u9jx0v                     CANCELLED                           2
REGISTRATION     SYS_AI_9nr176um7dc3x                     CANCELLED                           1
REGISTRATION     SYS_AI_b7wfmv59u3nx6                     STUDENT_ID                          1
REGISTRATION     SYS_AI_b7wfmv59u3nx6                     CLASS_ID                            2
AUTO_IX          SYS_AI_bbtzahkgk9f9s                     DIST_NO                             1
CUSTOMER         SYS_AI_fyjgc63q5mz1d                     CREDIT_CARD                         1
```

Created 2 indexes –
1 on Class
1 on Registration

Auto Indexes Created

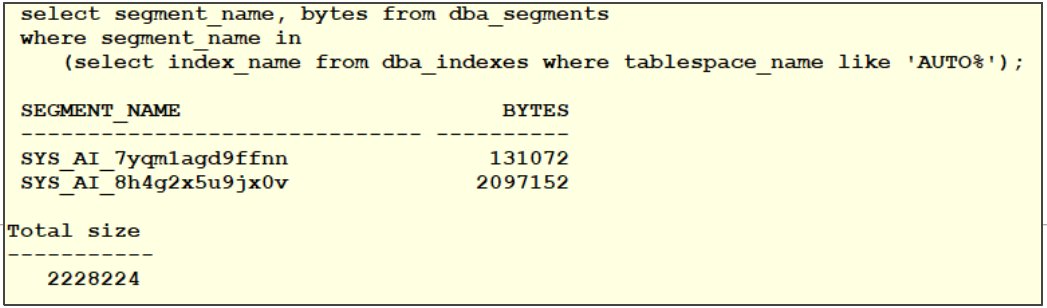
- Shows status of indexes
  - 2 indexes are taking up space



```
SQL> select index_name, status, dropped, visibility, segment_created
  2  from user_indexes where auto='YES';

INDEX_NAME                          STATUS   DRO VISIBILIT SEG
----------------------------------- -------- --- --------- ---
SYS_AI_bbtzahkgk9f9s                UNUSABLE NO  INVISIBLE NO
SYS_AI_76tdrszhyq6sm                UNUSABLE NO  INVISIBLE NO
SYS_AI_7yqm1agd9ffnn                VALID    NO  INVISIBLE YES
SYS_AI_fyjgc63q5mz1d                UNUSABLE NO  INVISIBLE NO
SYS_AI_b7wfmv59u3nx6                UNUSABLE NO  INVISIBLE NO
SYS_AI_8h4g2x5u9jx0v                VALID    NO  INVISIBLE YES
SYS_AI_9nr176um7dc3x                UNUSABLE NO  INVISIBLE NO
```

Class.name

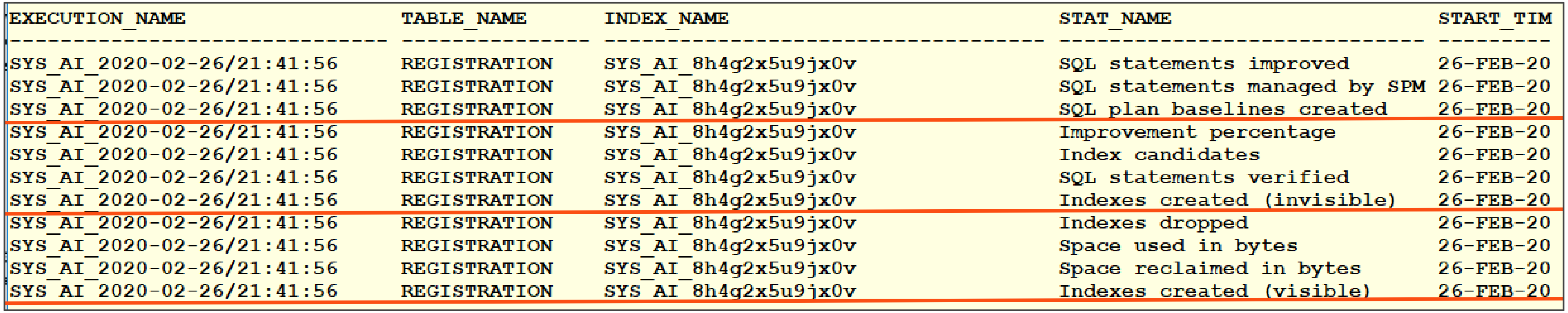
Reg.Class_id,Canceled



```
 select segment_name, bytes from dba_segments
 where segment_name in
    (select index_name from dba_indexes where tablespace_name like 'AUTO%');

 SEGMENT_NAME                         BYTES
 ------------------------------ ----------
 SYS_AI_7yqm1agd9ffnn               131072
 SYS_AI_8h4g2x5u9jx0v              2097152

Total size
-----------
   2228224
```

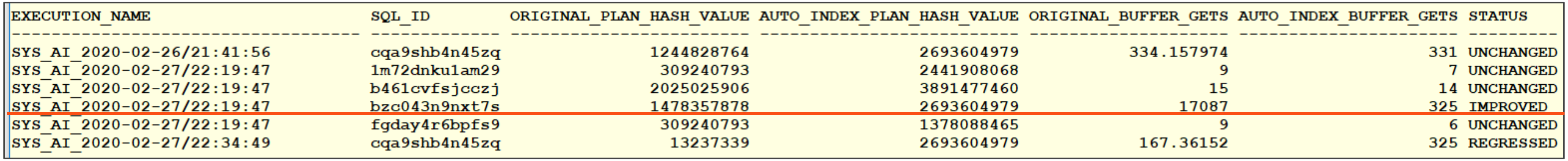
Auto Index Rational of Registration (class_id, canceled)

```
SELECT a.execution_name, a.table_name,
      a.index_name, b.stat_name, a.start_time
FROM dba_auto_index_ind_actions a, dba_auto_index_statistics b
WHERE a.execution_name = b.execution_name
ORDER BY 5,3;
```



```
EXECUTION_NAME                 TABLE_NAME      INDEX_NAME                          STAT_NAME                     START_TIM
------------------------------ --------------- ----------------------------------- ----------------------------- ---------
SYS_AI_2020-02-26/21:41:56     REGISTRATION    SYS_AI_8h4g2x5u9jx0v                SQL statements improved       26-FEB-20
SYS_AI_2020-02-26/21:41:56     REGISTRATION    SYS_AI_8h4g2x5u9jx0v                SQL statements managed by SPM 26-FEB-20
SYS_AI_2020-02-26/21:41:56     REGISTRATION    SYS_AI_8h4g2x5u9jx0v                SQL plan baselines created    26-FEB-20
SYS_AI_2020-02-26/21:41:56     REGISTRATION    SYS_AI_8h4g2x5u9jx0v                Improvement percentage        26-FEB-20
SYS_AI_2020-02-26/21:41:56     REGISTRATION    SYS_AI_8h4g2x5u9jx0v                Index candidates              26-FEB-20
SYS_AI_2020-02-26/21:41:56     REGISTRATION    SYS_AI_8h4g2x5u9jx0v                SQL statements verified       26-FEB-20
SYS_AI_2020-02-26/21:41:56     REGISTRATION    SYS_AI_8h4g2x5u9jx0v                Indexes created (invisible)   26-FEB-20
SYS_AI_2020-02-26/21:41:56     REGISTRATION    SYS_AI_8h4g2x5u9jx0v                Indexes dropped               26-FEB-20
SYS_AI_2020-02-26/21:41:56     REGISTRATION    SYS_AI_8h4g2x5u9jx0v                Space used in bytes           26-FEB-20
SYS_AI_2020-02-26/21:41:56     REGISTRATION    SYS_AI_8h4g2x5u9jx0v                Space reclaimed in bytes      26-FEB-20
SYS_AI_2020-02-26/21:41:56     REGISTRATION    SYS_AI_8h4g2x5u9jx0v                Indexes created (visible)     26-FEB-20
```

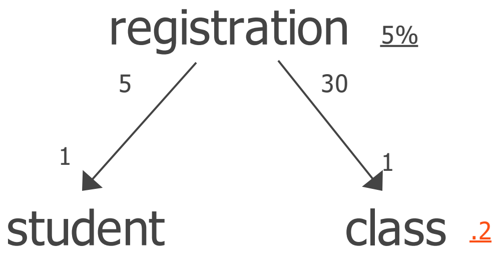
DBA_AUTO_INDEX_VERIFICATIONS



```
EXECUTION_NAME                      SQL_ID        ORIGINAL_PLAN_HASH_VALUE AUTO_INDEX_PLAN_HASH_VALUE ORIGINAL_BUFFER_GETS AUTO_INDEX_BUFFER_GETS STATUS
----------------------------------- ------------- ------------------------ -------------------------- -------------------- ---------------------- ---------
SYS_AI_2020-02-26/21:41:56          cqa9shb4n45zq               1244828764                 2693604979           334.157974                    331 UNCHANGED
SYS_AI_2020-02-27/22:19:47          1m72dnku1am29                309240793                 2441908068                    9                      7 UNCHANGED
SYS_AI_2020-02-27/22:19:47          b461cvfsjcczj               2025025906                 3891477460                   15                     14 UNCHANGED
SYS_AI_2020-02-27/22:19:47          bzc043n9nxt7s               1478357878                 2693604979                17087                    325 IMPROVED
SYS_AI_2020-02-27/22:19:47          fgday4r6bpfs9                309240793                 1378088465                    9                      6 UNCHANGED
SYS_AI_2020-02-27/22:34:49          cqa9shb4n45zq                 13237339                 2693604979            167.36152                    325 REGRESSED
```

SQL Diagramming

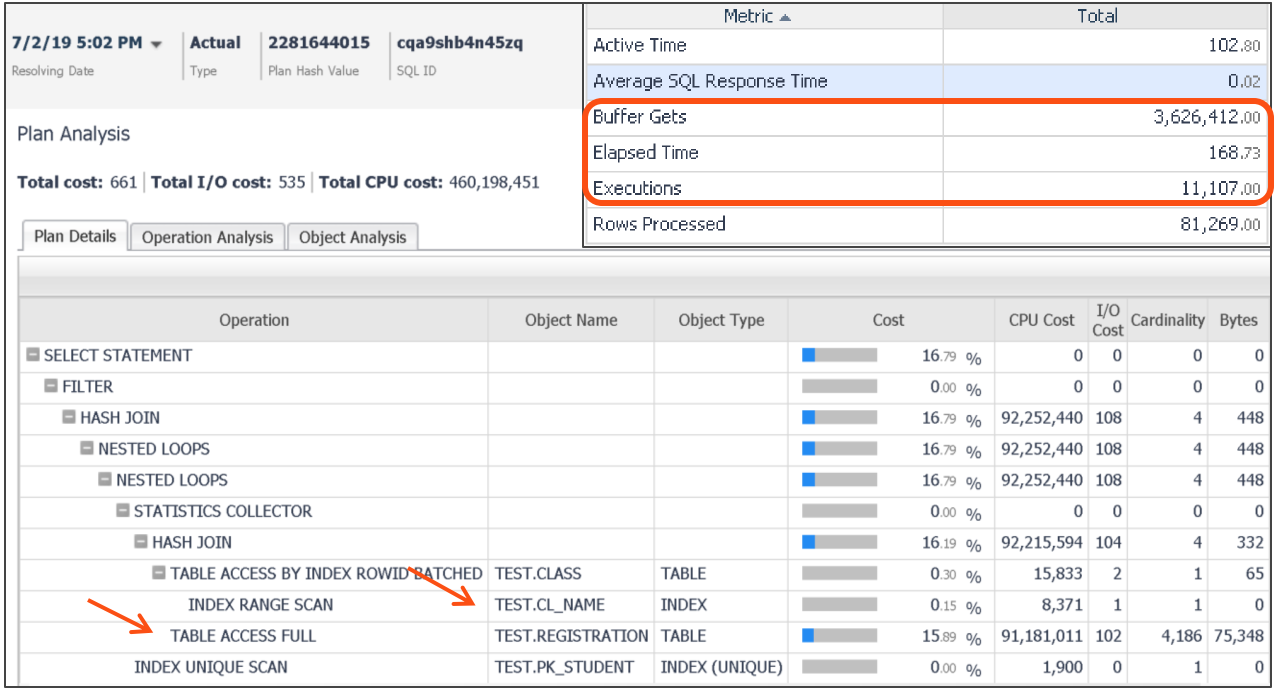
- Great Book “SQL Tuning” by Dan Tow
  - Oldie but a goodie that teaches SQL Diagramming
  - http://www.singingsql.com



```
select count(1) from registration where cancelled = 'N'
and signup_date between '2016-12-10 00:00' and '2016-12-11 00:00'

4344 / 80000 * 100  = 5.43%
                                                5.43

select count(1) from class where name = 'SQL TUNING'

    2 / 1000 * 100 = .2
```

New Execution Plan

- CREATE INDEX cl_name ON class(name);



7/2/19 5:02 PM Resolving Date | **Actual** Type | **2281644015** Plan Hash Value | **cqa9shb4n45zq** SQL ID

Metric ▲	Total
Active Time	102.80
Average SQL Response Time	0.02
Buffer Gets	3,626,412.00
Elapsed Time	168.73
Executions	11,107.00
Rows Processed	81,269.00

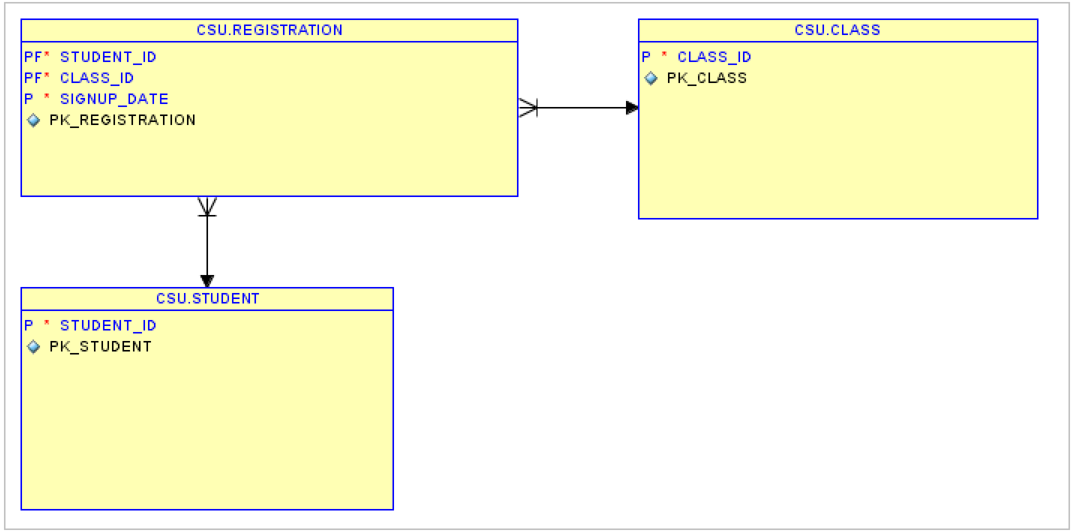
Plan Analysis

Total cost: 661 | **Total I/O cost:** 535 | **Total CPU cost:** 460,198,451

Operation	Object Name	Object Type	Cost	CPU Cost	I/O Cost	Cardinality	Bytes
SELECT STATEMENT			16.79 %	0	0	0	0
FILTER			0.00 %	0	0	0	0
HASH JOIN			16.79 %	92,252,440	108	4	448
NESTED LOOPS			16.79 %	92,252,440	108	4	448
NESTED LOOPS			16.79 %	92,252,440	108	4	448
STATISTICS COLLECTOR			0.00 %	0	0	0	0
HASH JOIN			16.19 %	92,215,594	104	4	332
TABLE ACCESS BY INDEX ROWID BATCHED	TEST.CLASS	TABLE	0.30 %	15,833	2	1	65
INDEX RANGE SCAN	TEST.CL_NAME	INDEX	0.15 %	8,371	1	1	0
TABLE ACCESS FULL	TEST.REGISTRATION	TABLE	15.89 %	91,181,011	102	4,186	75,348
INDEX UNIQUE SCAN	TEST.PK_STUDENT	INDEX (UNIQUE)	0.00 %	1,900	0	1	0

Review Index Order

- CLASS_ID not left leading in index



CSU.REGISTRATION

PF* STUDENT_ID
PF* CLASS_ID
P * SIGNUP_DATE
◆ PK_REGISTRATION

CSU.CLASS

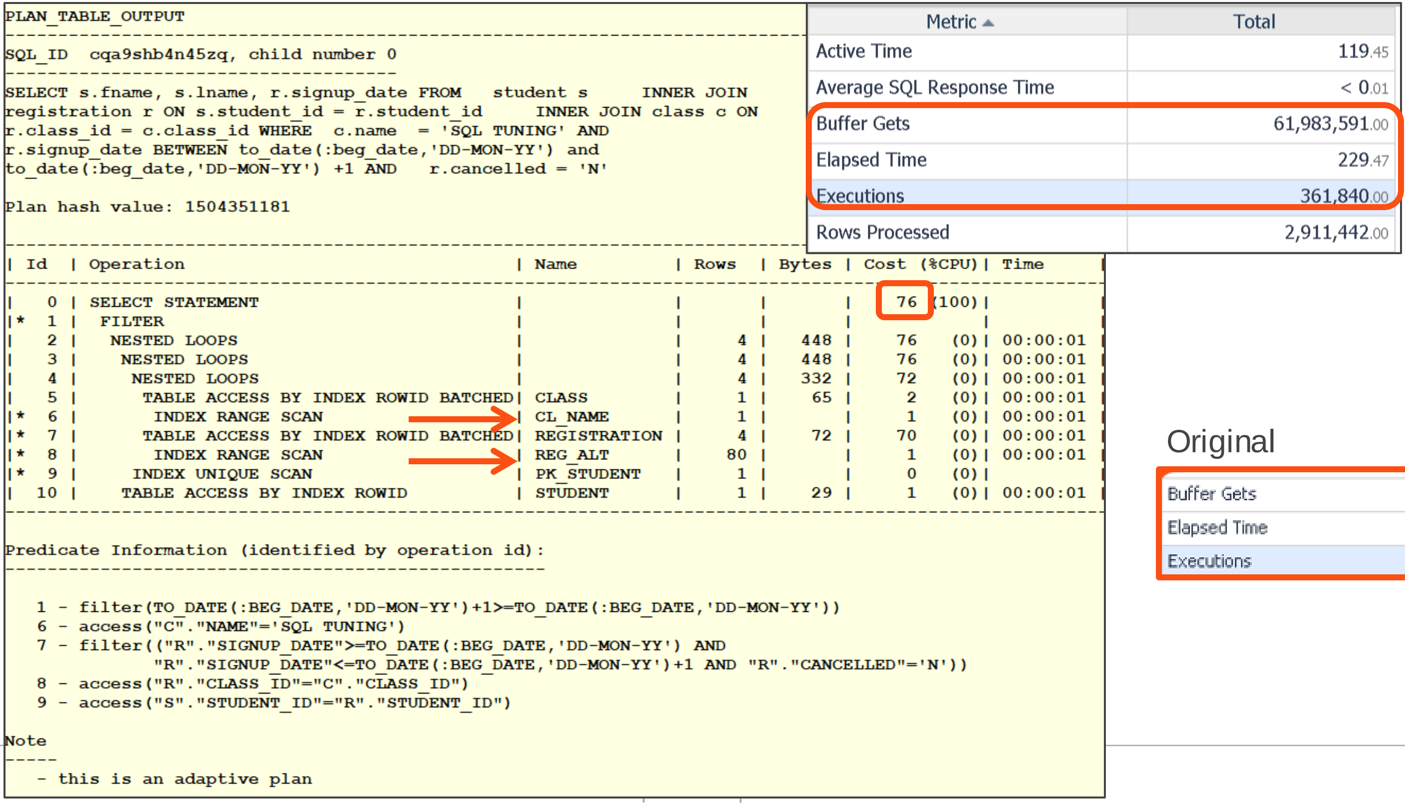
P * CLASS_ID
◆ PK_CLASS

CSU.STUDENT

P * STUDENT_ID
◆ PK_STUDENT

New Execution Plan

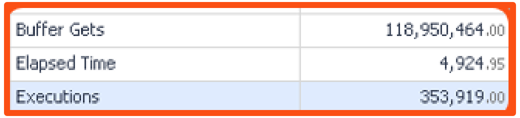
- CREATE INDEX reg_alt ON registration(class_id);



```
PLAN_TABLE_OUTPUT
------------------------------------------------------------------------------------------
SQL_ID  cqa9shb4n45zq, child number 0
-------------------------------------
SELECT s.fname, s.lname, r.signup_date FROM   student s     INNER JOIN
registration r ON s.student_id = r.student_id     INNER JOIN class c ON
r.class_id = c.class_id WHERE  c.name  = 'SQL TUNING' AND
r.signup_date BETWEEN to_date(:beg_date,'DD-MON-YY') and
to_date(:beg_date,'DD-MON-YY') +1 AND   r.cancelled = 'N'

Plan hash value: 1504351181

------------------------------------------------------------------------------------------------------------
| Id  | Operation                            | Name           | Rows  | Bytes | Cost (%CPU)| Time     |
------------------------------------------------------------------------------------------------------------
|   0 | SELECT STATEMENT                     |                |       |       |    76 (100)|          |
|*  1 |  FILTER                              |                |       |       |            |          |
|   2 |   NESTED LOOPS                       |                |     4 |   448 |    76   (0)| 00:00:01 |
|   3 |    NESTED LOOPS                      |                |     4 |   448 |    76   (0)| 00:00:01 |
|   4 |     NESTED LOOPS                     |                |     4 |   332 |    72   (0)| 00:00:01 |
|   5 |      TABLE ACCESS BY INDEX ROWID BATCHED| CLASS       |     1 |    65 |     2   (0)| 00:00:01 |
|*  6 |       INDEX RANGE SCAN               | CL_NAME        |     1 |       |     1   (0)| 00:00:01 |
|*  7 |      TABLE ACCESS BY INDEX ROWID BATCHED| REGISTRATION |     4 |    72 |    70   (0)| 00:00:01 |
|*  8 |       INDEX RANGE SCAN               | REG_ALT        |    80 |       |     1   (0)| 00:00:01 |
|*  9 |     INDEX UNIQUE SCAN                | PK_STUDENT     |     1 |       |     0   (0)|          |
|  10 |    TABLE ACCESS BY INDEX ROWID       | STUDENT        |     1 |    29 |     1   (0)| 00:00:01 |
------------------------------------------------------------------------------------------------------------

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

   1 - filter(TO_DATE(:BEG_DATE,'DD-MON-YY')+1>=TO_DATE(:BEG_DATE,'DD-MON-YY'))
   6 - access("C"."NAME"='SQL TUNING')
   7 - filter(("R"."SIGNUP_DATE">=TO_DATE(:BEG_DATE,'DD-MON-YY') AND
              "R"."SIGNUP_DATE"<=TO_DATE(:BEG_DATE,'DD-MON-YY')+1 AND "R"."CANCELLED"='N'))
   8 - access("R"."CLASS_ID"="C"."CLASS_ID")
   9 - access("S"."STUDENT_ID"="R"."STUDENT_ID")

Note
-----
   - this is an adaptive plan
```

Metric ▲	Total
Active Time	119.45
Average SQL Response Time	< 0.01
Buffer Gets	61,983,591.00
Elapsed Time	229.47
Executions	361,840.00
Rows Processed	2,911,442.00

Original

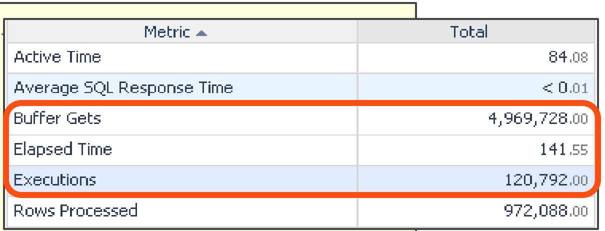


Buffer Gets	118,950,464.00
Elapsed Time	4,924.95
Executions	353,919.00

Tuning Advisor Suggested Index

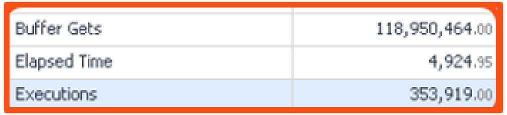
create index REG_CANCEL_SIGNUP on registration (cancelled, signup_date,class_id, student_id);

```
PLAN_TABLE_OUTPUT
-----------------------------------------------------------------------
SQL_ID  cqa9shb4n45zq, child number 0
-------------------------------------
SELECT s.fname, s.lname, r.signup_date FROM   student s     INNER JOIN
registration r ON s.student_id = r.student_id     INNER JOIN class c ON
r.class_id = c.class_id WHERE  c.name  = 'SQL TUNING' AND
r.signup_date BETWEEN to_date(:beg_date,'DD-MON-YY') and
to_date(:beg_date,'DD-MON-YY') +1 AND   r.cancelled = 'N'

Plan hash value: 1192206169

-----------------------------------------------------------------------------------------------------------
| Id  | Operation                          | Name               | Rows  | Bytes | Cost (%CPU)| Time     |
-----------------------------------------------------------------------------------------------------------
|   0 | SELECT STATEMENT                   |                    |       |       |    23 (100)|          |
|*  1 |  FILTER                            |                    |       |       |            |          |
|   2 |   NESTED LOOPS                     |                    |     4 |   448 |    23   (0)| 00:00:01 |
|   3 |    NESTED LOOPS                    |                    |     4 |   448 |    23   (0)| 00:00:01 |
|*  4 |     HASH JOIN                      |                    |     4 |   332 |    19   (0)| 00:00:01 |
|   5 |      TABLE ACCESS BY INDEX ROWID BATCHED| CLASS         |     1 |    65 |     2   (0)| 00:00:01 |
|*  6 |       INDEX RANGE SCAN             | CL_NAME            |     1 |       |     1   (0)| 00:00:01 |
|*  7 |      INDEX RANGE SCAN              | REG_CANCEL_SIGNUP  |     4 |    72 |    17   (0)| 00:00:01 |
|*  8 |     INDEX UNIQUE SCAN              | PK_STUDENT         |     1 |       |     0   (0)|          |
|   9 |    TABLE ACCESS BY INDEX ROWID     | STUDENT            |     1 |    29 |     1   (0)| 00:00:01 |
-----------------------------------------------------------------------------------------------------------

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

   1 - filter(TO_DATE(:BEG_DATE,'DD-MON-YY')+1>=TO_DATE(:BEG_DATE,'DD-MON-YY'))
   4 - access("R"."CLASS_ID"="C"."CLASS_ID")
   6 - access("C"."NAME"='SQL TUNING')
   7 - access("R"."CANCELLED"='N' AND "R"."SIGNUP_DATE">=TO_DATE(:BEG_DATE,'DD-MON-YY') AND
              "R"."SIGNUP_DATE"<=TO_DATE(:BEG_DATE,'DD-MON-YY')+1)
   8 - access("S"."STUDENT_ID"="R"."STUDENT_ID")

Note
-----
   - this is an adaptive plan
```



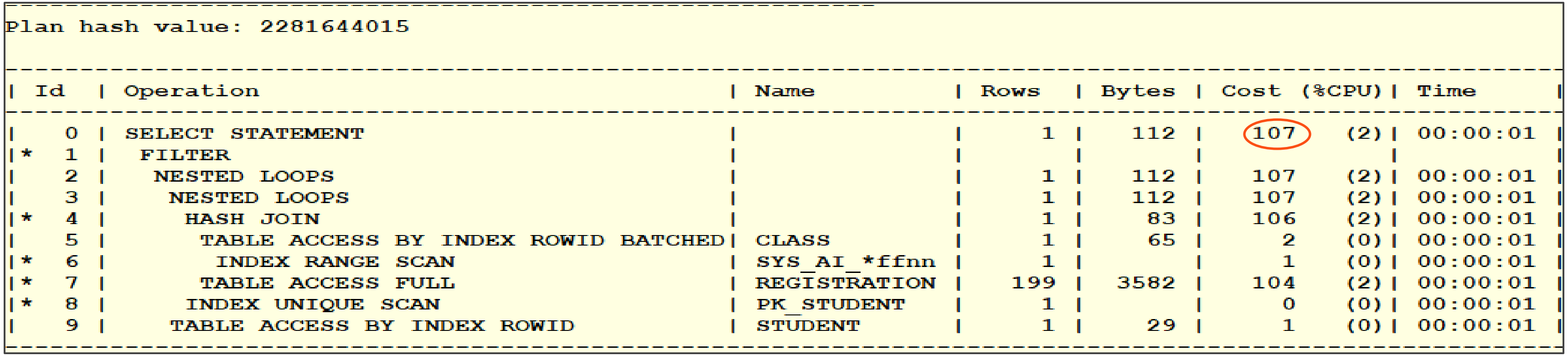
Metric ▲	Total
Active Time	84.08
Average SQL Response Time	< 0.01
Buffer Gets	4,969,728.00
Elapsed Time	141.55
Executions	120,792.00
Rows Processed	972,088.00

Original

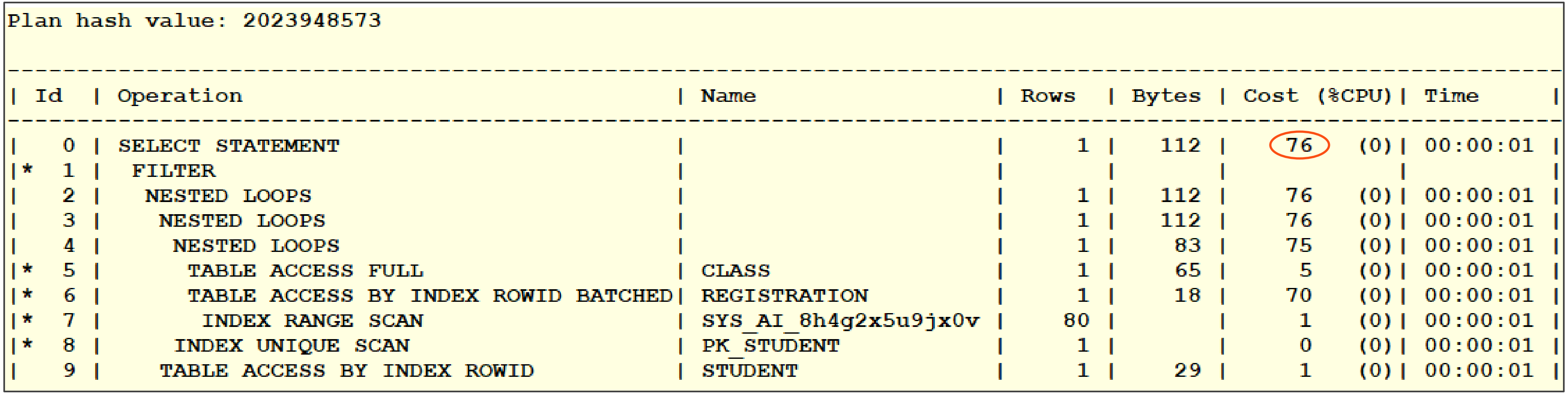


Buffer Gets	118,950,464.00
Elapsed Time	4,924.95
Executions	353,919.00

Auto Indexes on Class Or Registration (Not Both)



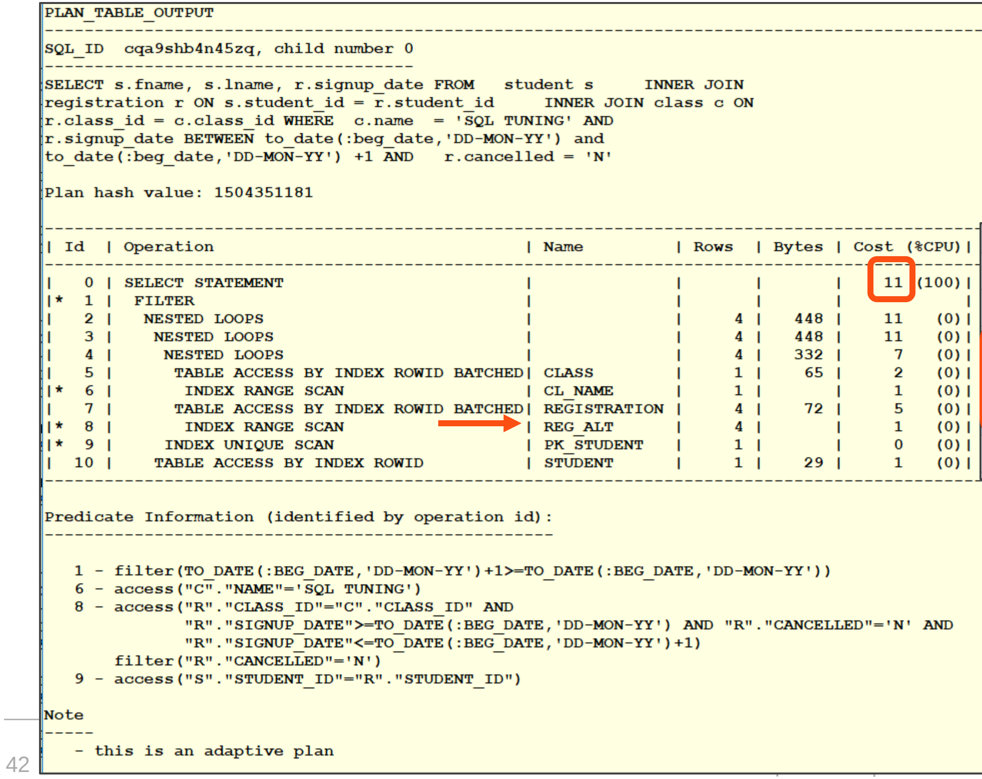
```
------------------------------------------------------------
Plan hash value: 2281644015

----------------------------------------------------------------------------------------------------------
| Id  | Operation                               | Name           | Rows  | Bytes | Cost (%CPU)| Time     |
----------------------------------------------------------------------------------------------------------
|   0 | SELECT STATEMENT                        |                |     1 |   112 |   107   (2)| 00:00:01 |
|*  1 |  FILTER                                 |                |       |       |            |          |
|   2 |   NESTED LOOPS                          |                |     1 |   112 |   107   (2)| 00:00:01 |
|   3 |    NESTED LOOPS                         |                |     1 |   112 |   107   (2)| 00:00:01 |
|*  4 |     HASH JOIN                           |                |     1 |    83 |   106   (2)| 00:00:01 |
|   5 |      TABLE ACCESS BY INDEX ROWID BATCHED| CLASS          |     1 |    65 |     2   (0)| 00:00:01 |
|*  6 |       INDEX RANGE SCAN                  | SYS_AI_*ffnn   |     1 |       |     1   (0)| 00:00:01 |
|*  7 |      TABLE ACCESS FULL                  | REGISTRATION   |   199 |  3582 |   104   (2)| 00:00:01 |
|*  8 |     INDEX UNIQUE SCAN                   | PK_STUDENT     |     1 |       |     0   (0)| 00:00:01 |
|   9 |    TABLE ACCESS BY INDEX ROWID          | STUDENT        |     1 |    29 |     1   (0)| 00:00:01 |
----------------------------------------------------------------------------------------------------------
```



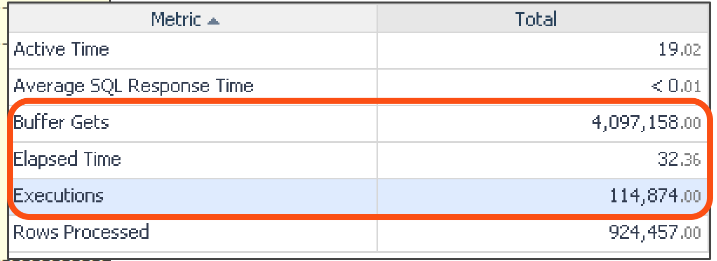
```
Plan hash value: 2023948573

-----------------------------------------------------------------------------------------------------------------
| Id  | Operation                               | Name                  | Rows  | Bytes | Cost (%CPU)| Time     |
-----------------------------------------------------------------------------------------------------------------
|   0 | SELECT STATEMENT                        |                       |     1 |   112 |    76   (0)| 00:00:01 |
|*  1 |  FILTER                                 |                       |       |       |            |          |
|   2 |   NESTED LOOPS                          |                       |     1 |   112 |    76   (0)| 00:00:01 |
|   3 |    NESTED LOOPS                         |                       |     1 |   112 |    76   (0)| 00:00:01 |
|   4 |     NESTED LOOPS                        |                       |     1 |    83 |    75   (0)| 00:00:01 |
|*  5 |      TABLE ACCESS FULL                  | CLASS                 |     1 |    65 |     5   (0)| 00:00:01 |
|*  6 |      TABLE ACCESS BY INDEX ROWID BATCHED| REGISTRATION          |     1 |    18 |    70   (0)| 00:00:01 |
|*  7 |       INDEX RANGE SCAN                  | SYS_AI_8h4g2x5u9jx0v  |    80 |       |     1   (0)| 00:00:01 |
|*  8 |     INDEX UNIQUE SCAN                   | PK_STUDENT            |     1 |       |     0   (0)| 00:00:01 |
|   9 |    TABLE ACCESS BY INDEX ROWID          | STUDENT               |     1 |    29 |     1   (0)| 00:00:01 |
```

Better Execution Plan – DBA Intervention

CREATE INDEX reg_alt ON registration(class_id,signup_date, cancelled);



```
PLAN_TABLE_OUTPUT
-----------------------------------------------------------------------------------------------
SQL_ID  cqa9shb4n45zq, child number 0
-----------------------------------
SELECT s.fname, s.lname, r.signup_date FROM   student s     INNER JOIN
registration r ON s.student_id = r.student_id      INNER JOIN class c ON
r.class_id = c.class_id WHERE  c.name  = 'SQL TUNING' AND
r.signup_date BETWEEN to_date(:beg_date,'DD-MON-YY') and
to_date(:beg_date,'DD-MON-YY') +1 AND   r.cancelled = 'N'

Plan hash value: 1504351181

-----------------------------------------------------------------------------------------------
| Id  | Operation                              | Name          | Rows  | Bytes | Cost (%CPU)|
-----------------------------------------------------------------------------------------------
|   0 | SELECT STATEMENT                       |               |       |       |    11 (100)|
|*  1 |  FILTER                                |               |       |       |            |
|   2 |   NESTED LOOPS                         |               |     4 |   448 |    11   (0)|
|   3 |    NESTED LOOPS                        |               |     4 |   448 |    11   (0)|
|   4 |     NESTED LOOPS                       |               |     4 |   332 |     7   (0)|
|   5 |      TABLE ACCESS BY INDEX ROWID BATCHED| CLASS        |     1 |    65 |     2   (0)|
|*  6 |       INDEX RANGE SCAN                 | CL_NAME       |     1 |       |     1   (0)|
|   7 |      TABLE ACCESS BY INDEX ROWID BATCHED| REGISTRATION |     4 |    72 |     5   (0)|
|*  8 |       INDEX RANGE SCAN                 | REG_ALT       |     4 |       |     1   (0)|
|*  9 |     INDEX UNIQUE SCAN                  | PK_STUDENT    |     1 |       |     0   (0)|
|  10 |    TABLE ACCESS BY INDEX ROWID         | STUDENT       |     1 |    29 |     1   (0)|
-----------------------------------------------------------------------------------------------

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

   1 - filter(TO_DATE(:BEG_DATE,'DD-MON-YY')+1>=TO_DATE(:BEG_DATE,'DD-MON-YY'))
   6 - access("C"."NAME"='SQL TUNING')
   8 - access("R"."CLASS_ID"="C"."CLASS_ID" AND
              "R"."SIGNUP_DATE">=TO_DATE(:BEG_DATE,'DD-MON-YY') AND "R"."CANCELLED"='N' AND
              "R"."SIGNUP_DATE"<=TO_DATE(:BEG_DATE,'DD-MON-YY')+1)
       filter("R"."CANCELLED"='N')
   9 - access("S"."STUDENT_ID"="R"."STUDENT_ID")

Note
-----
   - this is an adaptive plan
```

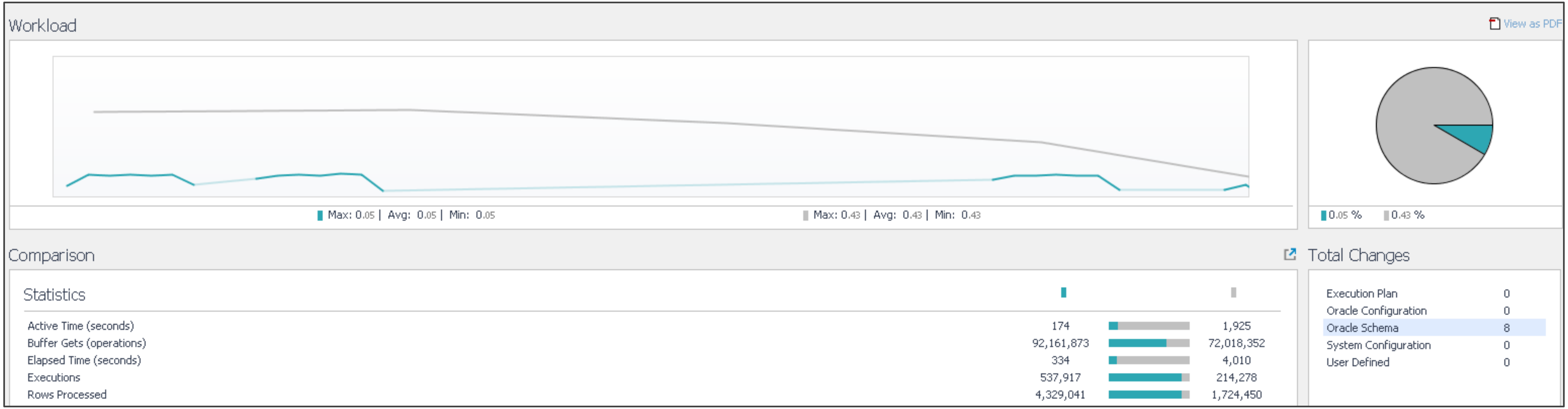


Metric ▲	Total
Active Time	19.02
Average SQL Response Time	< 0.01
Buffer Gets	4,097,158.00
Elapsed Time	32.36
Executions	114,874.00
Rows Processed	924,457.00

DBA Index on Registration Wins

- Original Plan cost 114
- Tuning Advisor on Class(name, class_id), Registration(cancelled, signup_date, class_id, student_id)
  - Cost 23
- Auto Index on Class(name) cost 107
- Auto Index on Registration(class_id, canceled) cost 76
- DBA Index on Class(name), Registration(class_id, signup_date, cancelled)
  - Cost **11**

Performance Improved?



Workload

🟦 Max: 0.05 | Avg: 0.05 | Min: 0.05

⬜ Max: 0.43 | Avg: 0.43 | Min: 0.43

🟦 0.05 %

⬜ 0.43 %

Comparison

Statistics

	🟦	⬜
Active Time (seconds)	174	1,925
Buffer Gets (operations)	92,161,873	72,018,352
Elapsed Time (seconds)	334	4,010
Executions	537,917	214,278
Rows Processed	4,329,041	1,724,450

Total Changes

Execution Plan	0
Oracle Configuration	0
Oracle Schema	8
System Configuration	0
User Defined	0

5. Engineer out the Stupid

- Look for Performance Inhibitors
  - Cursor or row by row processing
  - Parallel processing
    - Don’t use in an OLTP environment
    - Use only when accessing large data sets and additional resources can be allocated
  - Nested views that use db_links
  - Abuse of Wild Cards (*) or No Where Clause
    - Select ONLY those columns in a query which are required.
    - Extra columns cause more I/O on the database & increase network traffic
    - Code-based SQL Generators (e.g. Hibernate)
  - Using functions on indexed columns (SUBSTR, TO_CHAR, UPPER, TRUNC)
    - Optimizer can’t use the index
    - Instead move the function to the constant or variable side of equation
    - Consider creating a function based index
  - Hard-coded Hints

```
select… where upper(last_name) = ‘GRIFFIN’
Better way:  select … where last_name = upper(:b1);
```

More Do’s and Don’ts

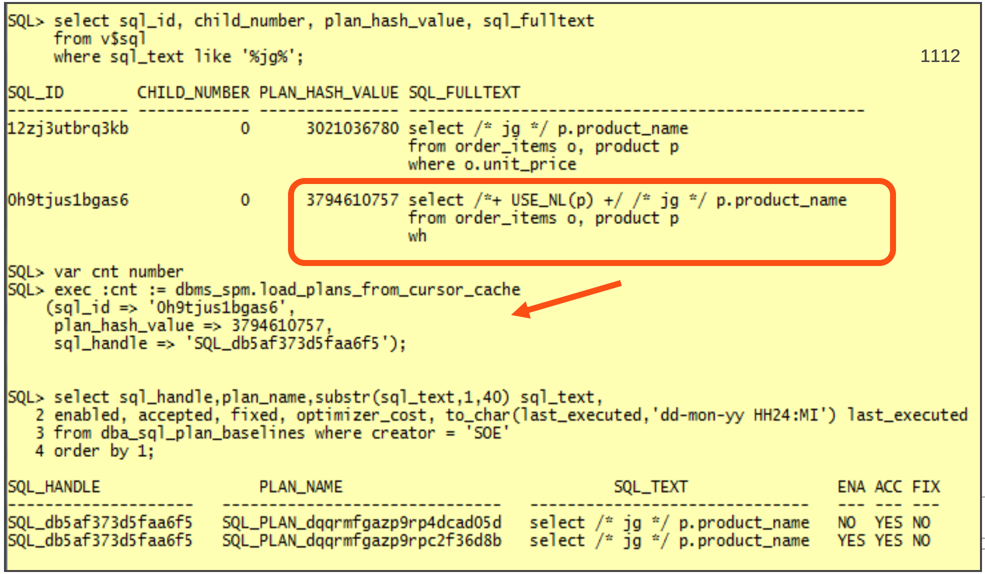
- Reduce SORT operations as they slow down your queries
  - Don’t use the UNION operator if you can use UNION ALL
  - Don’t use the DISTINCT keyword if you don’t need it
- When using a composite/multi-column index, access the left-leading column (in WHERE)
  - An INDEX SKIP SCAN may occur which is often no better than a FULL TABLE SCAN
- Try to avoid Cartesian product queries
- Use bind variables instead of literal values
  - To reduce repeated parsing of the same statement
- If using sub-queries, make use of the EXISTS operator when possible
  - Optimizer will stop with a match and avoid a FULL TABLE SCAN
- Try to use an index if less than 5% of the data needs to be accessed
  - Exception: small table are best accessed through a FULL TABLE SCAN
    - Consider keeping in memory

Avoid Common Pitfalls

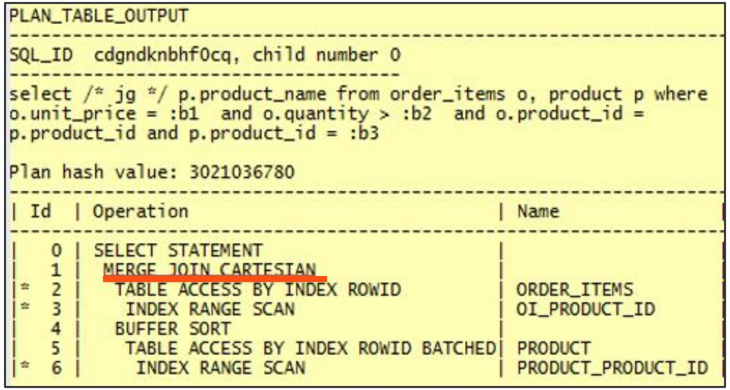
- Use equi-joins whenever possible
  - Try not to use ‘not in’, !=, <>, not null, etc…
  - Optimizer has more choices to choose from
- Avoid complex expressions such as NVL(col1,0), TO_DATE(), TO_NUMBER(), etc…
  - They prevent the optimizer from assigning valid cardinality or selectivity estimates
  - Can affect the overall plan and the join methods
- Avoid joining complex views
  - May instantiate all views to run query against (reading too much data)
  - Querying views requires all tables from the view to be accessed
    - If they aren’t required, then don’t use the view
- Use the partition key in the ‘WHERE’ clause if querying a partitioned table
  - Partition pruning will be used to reduce the amount of data read

When you need to uses hints

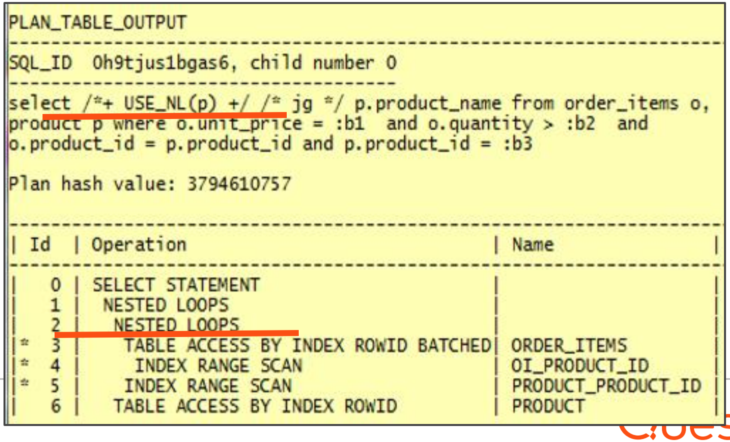
- If you can hint it, baseline it (per Tom Kyte)
  - Alternative to using hints
    - Hints difficult to manage over time
    - Once added, usually forgotten about
  - 3rd Party Software – can’t modify code



```
SQL> select sql_id, child_number, plan_hash_value, sql_fulltext
     from v$sql
     where sql_text like '%jg%';                                                  1112

SQL_ID        CHILD_NUMBER PLAN_HASH_VALUE SQL_FULLTEXT
------------- ------------ --------------- ----------------------------------------------
12zj3utbrq3kb            0      3021036780 select /* jg */ p.product_name
                                           from order_items o, product p
                                           where o.unit_price

0h9tjus1bgas6            0      3794610757 select /*+ USE_NL(p) +/ /* jg */ p.product_name
                                           from order_items o, product p
                                           wh

SQL> var cnt number
SQL> exec :cnt := dbms_spm.load_plans_from_cursor_cache
    (sql_id => '0h9tjus1bgas6',
     plan_hash_value => 3794610757,
     sql_handle => 'SQL_db5af373d5faa6f5');


SQL> select sql_handle,plan_name,substr(sql_text,1,40) sql_text,
   2 enabled, accepted, fixed, optimizer_cost, to_char(last_executed,'dd-mon-yy HH24:MI') last_executed
   3 from dba_sql_plan_baselines where creator = 'SOE'
   4 order by 1;

SQL_HANDLE              PLAN_NAME                        SQL_TEXT                     ENA ACC FIX
--------------------   ------------------------------   ------------------------------ --- --- ---
SQL_db5af373d5faa6f5   SQL_PLAN_dqqrmfgazp9rp4dcad05d   select /* jg */ p.product_name NO  YES NO
SQL_db5af373d5faa6f5   SQL_PLAN_dqqrmfgazp9rpc2f36d8b   select /* jg */ p.product_name YES YES NO
```



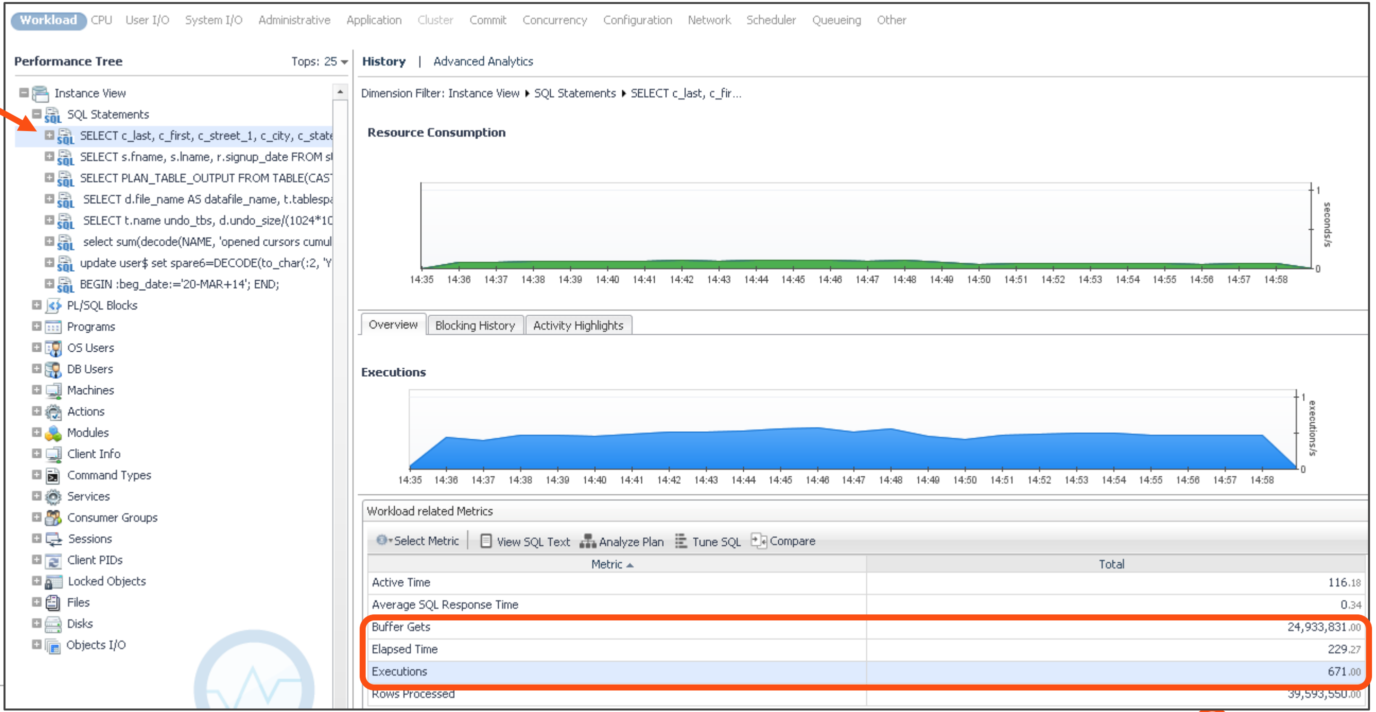
```
PLAN_TABLE_OUTPUT
-------------------------------------------------------------------------------------
SQL_ID  cdgndknbhf0cq, child number 0
-------------------------------------
select /* jg */ p.product_name from order_items o, product p where
o.unit_price = :b1  and o.quantity > :b2  and o.product_id =
p.product_id and p.product_id = :b3

Plan hash value: 3021036780

-------------------------------------------------------------------------------------
| Id  | Operation                             | Name
-------------------------------------------------------------------------------------
|   0 | SELECT STATEMENT                      |
|   1 |  MERGE JOIN CARTESIAN                 |
|*  2 |   TABLE ACCESS BY INDEX ROWID         | ORDER_ITEMS
|*  3 |    INDEX RANGE SCAN                   | OI_PRODUCT_ID
|   4 |   BUFFER SORT                         |
|   5 |    TABLE ACCESS BY INDEX ROWID BATCHED| PRODUCT
|*  6 |     INDEX RANGE SCAN                  | PRODUCT_PRODUCT_ID
```



```
PLAN_TABLE_OUTPUT
-------------------------------------------------------------------------------------
SQL_ID  0h9tjus1bgas6, child number 0
-------------------------------------
select /*+ USE_NL(p) +/ /* 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 and p.product_id = :b3

Plan hash value: 3794610757


-------------------------------------------------------------------------------------
| Id  | Operation                             | Name               |
-------------------------------------------------------------------------------------
|   0 | SELECT STATEMENT                      |                    |
|   1 |  NESTED LOOPS                         |                    |
|   2 |   NESTED LOOPS                        |                    |
|*  3 |    TABLE ACCESS BY INDEX ROWID BATCHED| ORDER_ITEMS        |
|*  4 |     INDEX RANGE SCAN                  | OI_PRODUCT_ID      |
|*  5 |    INDEX RANGE SCAN                   | PRODUCT_PRODUCT_ID |
|   6 |   TABLE ACCESS BY INDEX ROWID         | PRODUCT            |
```

Case Study 2 – Orders by Customer Last Name

```
SELECT c_last, c_first, c_street_1, c_city, c_state, c_zip,
       c_phone, o_entry_d, d_name, ol_delivery_d, ol_quantity, ol_amount
FROM order_line, orders, district, customer, stock
WHERE o_id = ol_o_id
AND o_c_id=c_id
AND s_i_id = ol_i_id
AND d_id = ol_d_id
AND ol_w_id = :B2
AND ol_d_id = :B4
AND (ol_o_id < :B3 )
AND ol_o_id >= (:B3 - 20)
AND s_w_id = :B2
AND s_quantity < :B1
AND d_id = :B4
AND c_last like :B5 ;
```



Metric ▲	Total
Active Time	116.18
Average SQL Response Time	0.34
Buffer Gets	24,933,831.00
Elapsed Time	229.27
Executions	671.00
Rows Processed	39,593,550.00

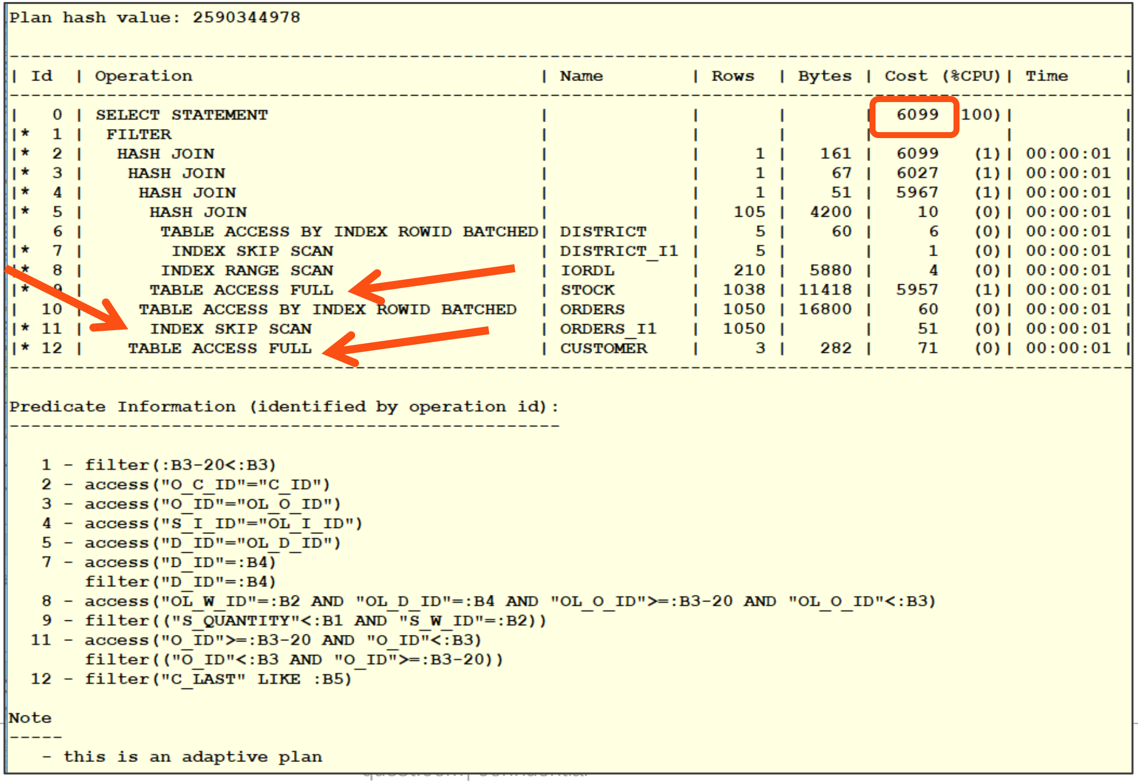
Review the Execution Plan

select * from table (dbms_xplan.display_cursor(null,null, format=> '+report'));

Buffer Gets: 25m

Executions: 671

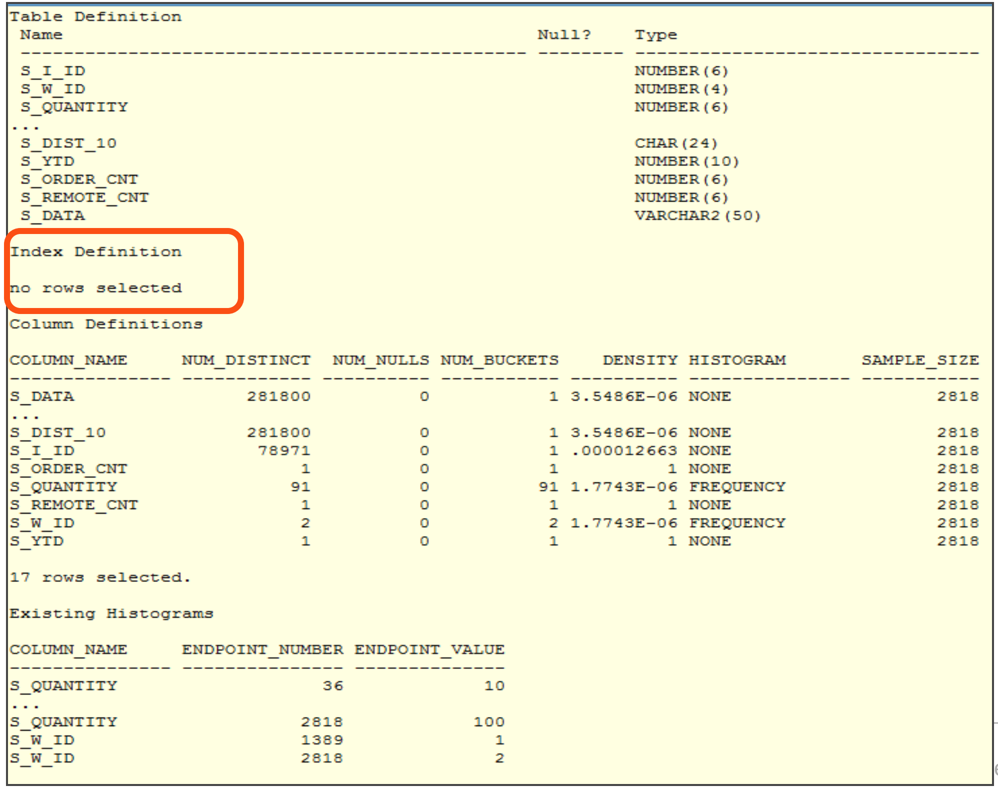
Elapsed Time: 229 secs



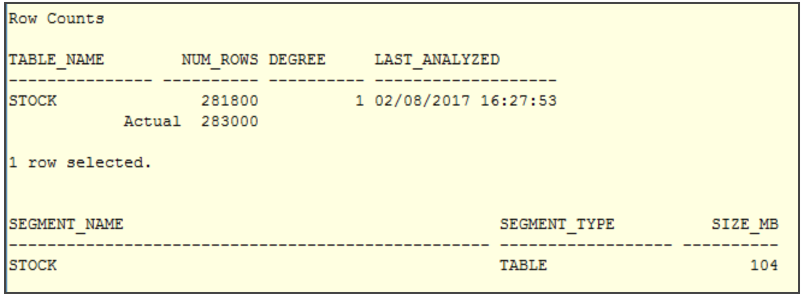
```
Plan hash value: 2590344978

------------------------------------------------------------------------------------------------------
| Id  | Operation                               | Name          | Rows  | Bytes | Cost (%CPU)| Time     |
------------------------------------------------------------------------------------------------------
|   0 | SELECT STATEMENT                        |               |       |       |  6099 100)|          |
|*  1 |  FILTER                                 |               |       |       |            |          |
|*  2 |   HASH JOIN                             |               |     1 |   161 |  6099   (1)| 00:00:01 |
|*  3 |    HASH JOIN                            |               |     1 |    67 |  6027   (1)| 00:00:01 |
|*  4 |     HASH JOIN                           |               |     1 |    51 |  5967   (1)| 00:00:01 |
|*  5 |      HASH JOIN                          |               |   105 |  4200 |    10   (0)| 00:00:01 |
|   6 |       TABLE ACCESS BY INDEX ROWID BATCHED| DISTRICT     |     5 |    60 |     6   (0)| 00:00:01 |
|*  7 |        INDEX SKIP SCAN                  | DISTRICT_I1   |     5 |       |     1   (0)| 00:00:01 |
|*  8 |       INDEX RANGE SCAN                  | IORDL         |   210 |  5880 |     4   (0)| 00:00:01 |
|*  9 |      TABLE ACCESS FULL                  | STOCK         |  1038 | 11418 |  5957   (1)| 00:00:01 |
|  10 |     TABLE ACCESS BY INDEX ROWID BATCHED | ORDERS        |  1050 | 16800 |    60   (0)| 00:00:01 |
|* 11 |      INDEX SKIP SCAN                    | ORDERS_I1     |  1050 |       |    51   (0)| 00:00:01 |
|* 12 |    TABLE ACCESS FULL                    | CUSTOMER      |     3 |   282 |    71   (0)| 00:00:01 |
------------------------------------------------------------------------------------------------------

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

   1 - filter(:B3-20<:B3)
   2 - access("O_C_ID"="C_ID")
   3 - access("O_ID"="OL_O_ID")
   4 - access("S_I_ID"="OL_I_ID")
   5 - access("D_ID"="OL_D_ID")
   7 - access("D_ID"=:B4)
       filter("D_ID"=:B4)
   8 - access("OL_W_ID"=:B2 AND "OL_D_ID"=:B4 AND "OL_O_ID">=:B3-20 AND "OL_O_ID"<:B3)
   9 - filter(("S_QUANTITY"<:B1 AND "S_W_ID"=:B2))
  11 - access("O_ID">=:B3-20 AND "O_ID"<:B3)
       filter(("O_ID"<:B3 AND "O_ID">=:B3-20))
  12 - filter("C_LAST" LIKE :B5)

Note
-----
   - this is an adaptive plan
```

Get Object Information

- Stock:



```
Table Definition
 Name                                            Null?    Type
 ----------------------------------------------- -------- --------------------------------
 S_I_ID                                                   NUMBER(6)
 S_W_ID                                                   NUMBER(4)
 S_QUANTITY                                               NUMBER(6)
...
 S_DIST_10                                                CHAR(24)
 S_YTD                                                    NUMBER(10)
 S_ORDER_CNT                                              NUMBER(6)
 S_REMOTE_CNT                                             NUMBER(6)
 S_DATA                                                   VARCHAR2(50)

Index Definition

no rows selected

Column Definitions

COLUMN_NAME     NUM_DISTINCT  NUM_NULLS NUM_BUCKETS    DENSITY HISTOGRAM       SAMPLE_SIZE
--------------- ------------ ---------- ----------- ---------- --------------- -----------
S_DATA                281800          0           1 3.5486E-06 NONE                   2818
...
S_DIST_10             281800          0           1 3.5486E-06 NONE                   2818
S_I_ID                 78971          0           1 .000012663 NONE                   2818
S_ORDER_CNT                1          0           1          1 NONE                   2818
S_QUANTITY                91          0          91 1.7743E-06 FREQUENCY              2818
S_REMOTE_CNT               1          0           1          1 NONE                   2818
S_W_ID                     2          0           2 1.7743E-06 FREQUENCY              2818
S_YTD                      1          0           1          1 NONE                   2818

17 rows selected.

Existing Histograms

COLUMN_NAME     ENDPOINT_NUMBER ENDPOINT_VALUE
--------------- --------------- --------------
S_QUANTITY                   36             10
...
S_QUANTITY                 2818            100
S_W_ID                     1389              1
S_W_ID                     2818              2
```

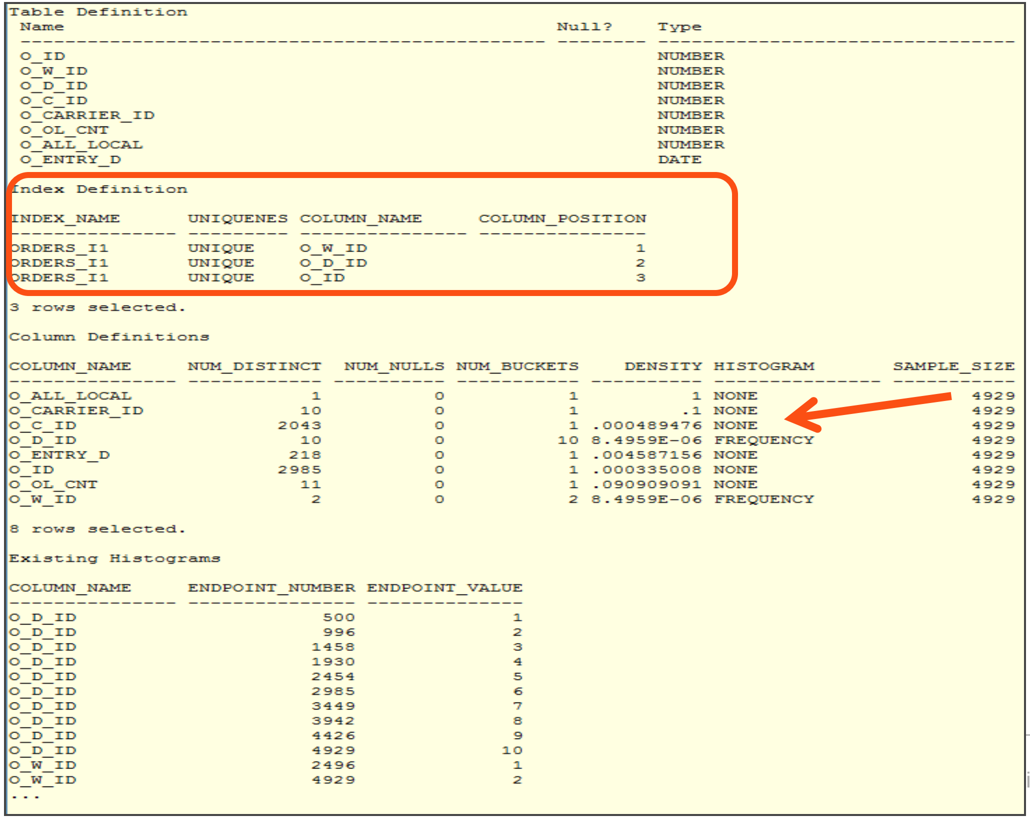


```
Row Counts

TABLE_NAME        NUM_ROWS     DEGREE LAST_ANALYZED
--------------- ---------- ---------- -------------------
STOCK               281800          1 02/08/2017 16:27:53
            Actual  283000

1 row selected.

SEGMENT_NAME                                       SEGMENT_TYPE          SIZE_MB
-------------------------------------------------- ------------------ ----------
STOCK                                              TABLE                     104
```

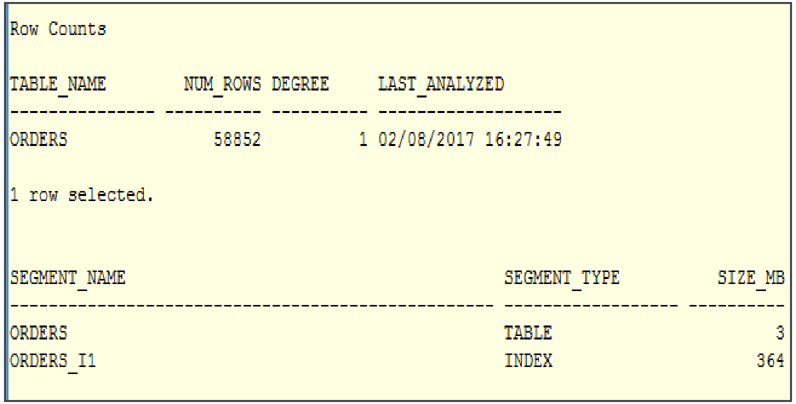
create index stock_idx
on stock
(s_i_id, s_w_id,
s_quantity);

Get Object Information

- Orders:



```
Table Definition
 Name                                            Null?    Type
 ----------------------------------------------- -------- --------------------------------
 O_ID                                                     NUMBER
 O_W_ID                                                   NUMBER
 O_D_ID                                                   NUMBER
 O_C_ID                                                   NUMBER
 O_CARRIER_ID                                             NUMBER
 O_OL_CNT                                                 NUMBER
 O_ALL_LOCAL                                              NUMBER
 O_ENTRY_D                                                DATE

Index Definition

INDEX_NAME      UNIQUENES COLUMN_NAME     COLUMN_POSITION
--------------- --------- --------------- ---------------
ORDERS_I1       UNIQUE    O_W_ID                        1
ORDERS_I1       UNIQUE    O_D_ID                        2
ORDERS_I1       UNIQUE    O_ID                          3

3 rows selected.

Column Definitions

COLUMN_NAME     NUM_DISTINCT  NUM_NULLS NUM_BUCKETS    DENSITY HISTOGRAM       SAMPLE_SIZE
--------------- ------------ ---------- ----------- ---------- --------------- -----------
O_ALL_LOCAL                1          0           1          1 NONE                   4929
O_CARRIER_ID              10          0           1         .1 NONE                   4929
O_C_ID                  2043          0           1 .000489476 NONE                   4929
O_D_ID                    10          0          10 8.4959E-06 FREQUENCY              4929
O_ENTRY_D                218          0           1 .004587156 NONE                   4929
O_ID                    2985          0           1 .000335008 NONE                   4929
O_OL_CNT                  11          0           1 .090909091 NONE                   4929
O_W_ID                     2          0           2 8.4959E-06 FREQUENCY              4929

8 rows selected.

Existing Histograms

COLUMN_NAME     ENDPOINT_NUMBER ENDPOINT_VALUE
--------------- --------------- --------------
O_D_ID                      500              1
O_D_ID                      996              2
O_D_ID                     1458              3
O_D_ID                     1930              4
O_D_ID                     2454              5
O_D_ID                     2985              6
O_D_ID                     3449              7
O_D_ID                     3942              8
O_D_ID                     4426              9
O_D_ID                     4929             10
O_W_ID                     2496              1
O_W_ID                     4929              2
...
```

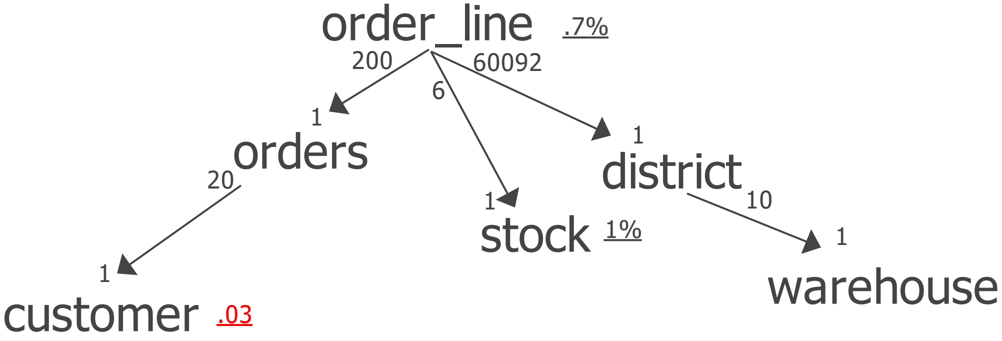


```
Row Counts

TABLE_NAME        NUM_ROWS DEGREE     LAST_ANALYZED
--------------- ---------- ---------- -------------------
ORDERS               58852          1 02/08/2017 16:27:49

1 row selected.


SEGMENT_NAME                                       SEGMENT_TYPE          SIZE_MB
-------------------------------------------------- ------------------ ----------
ORDERS                                             TABLE                       3
ORDERS_I1                                          INDEX                     364
```

Actual Rows = 60,000

Find the Driving Table

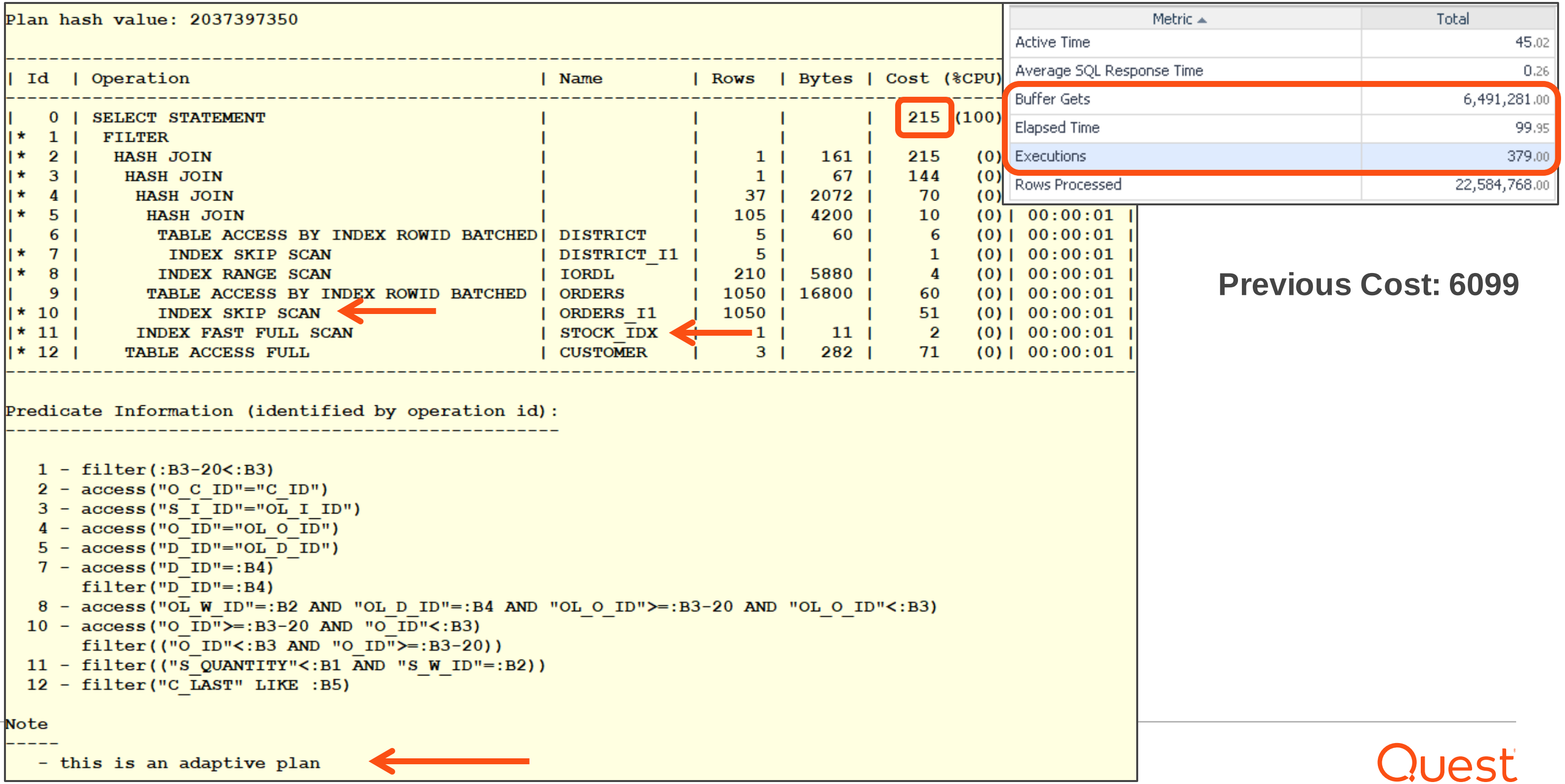
```
WHERE o_id = ol_o_id
AND o_c_id=c_id
AND s_i_id = ol_i_id
AND d_id = ol_d_id
AND ol_w_id = :B2
AND ol_d_id = :B4
AND (ol_o_id < :B3 )
AND ol_o_id >= (:B3 - 20)
AND s_w_id = :B2
AND s_quantity < :B1
AND d_id = :B4
AND c_last like :B5 ;
```



```
select count(*) from order_line
where ol_o_id < 200 and ol_o_id >= 200-20;

3941 / 600916  * 100 = .6558%

select avg(cnt) from (select c_last, count(*) cnt
from customer group by c_last);

20 / 60000 * 100 = .03333%

Filter on Stock: 3109 / 283000 * 100 = 1%
```

Engineer Out The Stupid

create index stock_idx on stock (s_i_id, s_w_id, s_quantity);

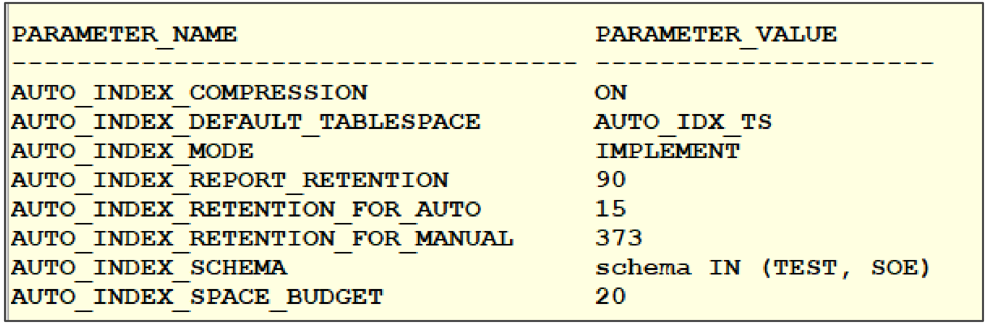


```
Plan hash value: 2037397350

---------------------------------------------------------------------------------------------
| Id  | Operation                               | Name         | Rows  | Bytes | Cost (%CPU)|
---------------------------------------------------------------------------------------------
|   0 | SELECT STATEMENT                        |              |       |       |   215 (100)|
|*  1 |  FILTER                                 |              |       |       |            |
|*  2 |   HASH JOIN                             |              |     1 |   161 |   215   (0)|
|*  3 |    HASH JOIN                            |              |     1 |    67 |   144   (0)|
|*  4 |     HASH JOIN                           |              |    37 |  2072 |    70   (0)|
|*  5 |      HASH JOIN                          |              |   105 |  4200 |    10   (0)| 00:00:01 |
|   6 |       TABLE ACCESS BY INDEX ROWID BATCHED| DISTRICT    |     5 |    60 |     6   (0)| 00:00:01 |
|*  7 |        INDEX SKIP SCAN                  | DISTRICT_I1  |     5 |       |     1   (0)| 00:00:01 |
|*  8 |       INDEX RANGE SCAN                  | IORDL        |   210 |  5880 |     4   (0)| 00:00:01 |
|   9 |      TABLE ACCESS BY INDEX ROWID BATCHED | ORDERS      |  1050 | 16800 |    60   (0)| 00:00:01 |
|* 10 |       INDEX SKIP SCAN                   | ORDERS_I1    |  1050 |       |    51   (0)| 00:00:01 |
|* 11 |     INDEX FAST FULL SCAN                | STOCK_IDX    |     1 |    11 |     2   (0)| 00:00:01 |
|* 12 |    TABLE ACCESS FULL                    | CUSTOMER     |     3 |   282 |    71   (0)| 00:00:01 |
-------------------------------------------------------------------------------------------------------

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

   1 - filter(:B3-20<:B3)
   2 - access("O_C_ID"="C_ID")
   3 - access("S_I_ID"="OL_I_ID")
   4 - access("O_ID"="OL_O_ID")
   5 - access("D_ID"="OL_D_ID")
   7 - access("D_ID"=:B4)
       filter("D_ID"=:B4)
   8 - access("OL_W_ID"=:B2 AND "OL_D_ID"=:B4 AND "OL_O_ID">=:B3-20 AND "OL_O_ID"<:B3)
  10 - access("O_ID">=:B3-20 AND "O_ID"<:B3)
       filter(("O_ID"<:B3 AND "O_ID">=:B3-20))
  11 - filter(("S_QUANTITY"<:B1 AND "S_W_ID"=:B2))
  12 - filter("C_LAST" LIKE :B5)

Note
-----
   - this is an adaptive plan
```

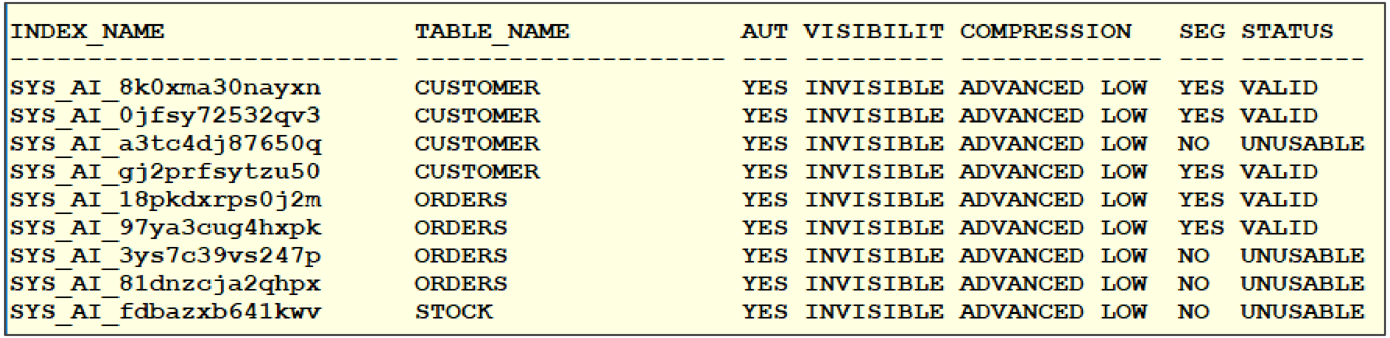
Metric ▲	Total
Active Time	45.02
Average SQL Response Time	0.26
Buffer Gets	6,491,281.00
Elapsed Time	99.95
Executions	379.00
Rows Processed	22,584,768.00

Previous Cost: 6099

Try Auto Indexing – Include SOE Schema



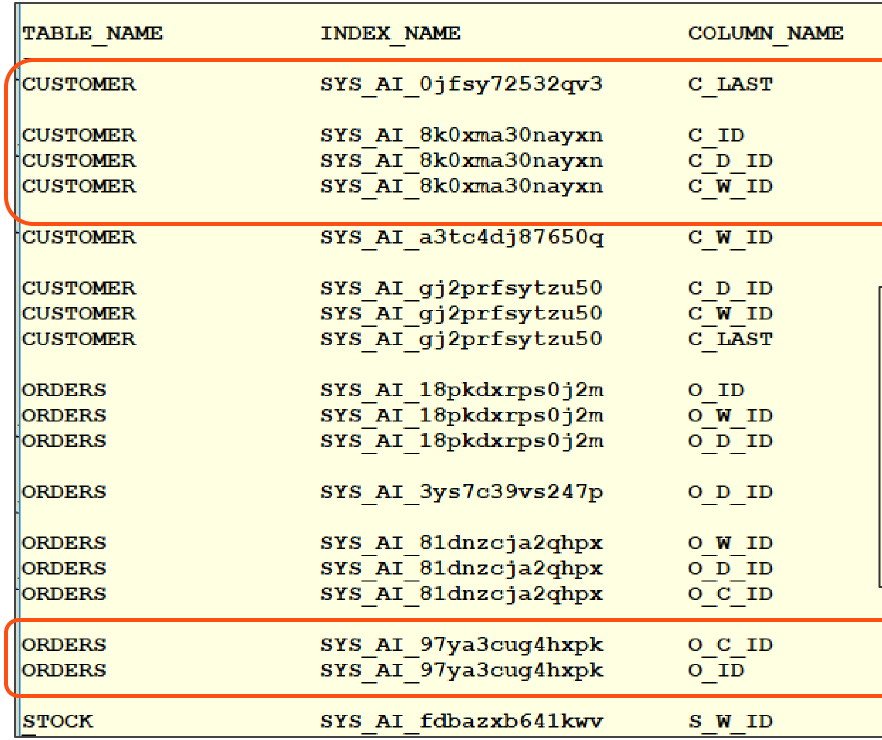
```
PARAMETER_NAME                      PARAMETER_VALUE
----------------------------------- ---------------------
AUTO_INDEX_COMPRESSION              ON
AUTO_INDEX_DEFAULT_TABLESPACE       AUTO_IDX_TS
AUTO_INDEX_MODE                     IMPLEMENT
AUTO_INDEX_REPORT_RETENTION         90
AUTO_INDEX_RETENTION_FOR_AUTO       15
AUTO_INDEX_RETENTION_FOR_MANUAL     373
AUTO_INDEX_SCHEMA                   schema IN (TEST, SOE)
AUTO_INDEX_SPACE_BUDGET             20
```



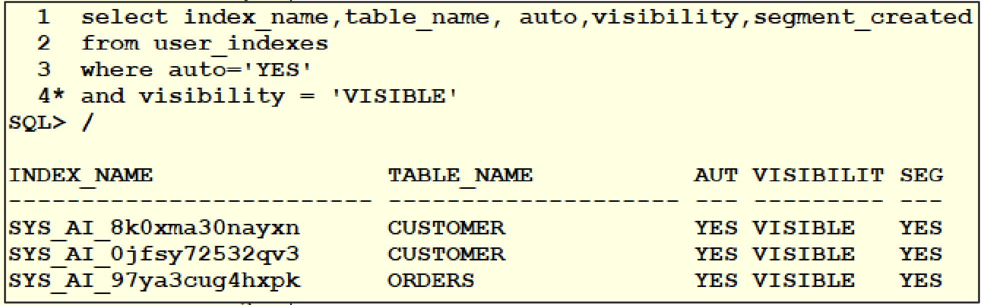
```
INDEX_NAME                TABLE_NAME           AUT VISIBILIT COMPRESSION   SEG STATUS
------------------------- -------------------- --- --------- ------------- --- --------
SYS_AI_8k0xma30nayxn      CUSTOMER             YES INVISIBLE ADVANCED LOW  YES VALID
SYS_AI_0jfsy72532qv3      CUSTOMER             YES INVISIBLE ADVANCED LOW  YES VALID
SYS_AI_a3tc4dj87650q      CUSTOMER             YES INVISIBLE ADVANCED LOW  NO  UNUSABLE
SYS_AI_gj2prfsytzu50      CUSTOMER             YES INVISIBLE ADVANCED LOW  YES VALID
SYS_AI_18pkdxrps0j2m      ORDERS               YES INVISIBLE ADVANCED LOW  YES VALID
SYS_AI_97ya3cug4hxpk      ORDERS               YES INVISIBLE ADVANCED LOW  YES VALID
SYS_AI_3ys7c39vs247p      ORDERS               YES INVISIBLE ADVANCED LOW  NO  UNUSABLE
SYS_AI_81dnzcja2qhpx      ORDERS               YES INVISIBLE ADVANCED LOW  NO  UNUSABLE
SYS_AI_fdbazxb641kwv      STOCK                YES INVISIBLE ADVANCED LOW  NO  UNUSABLE
```

```
SELECT index_name,table_name,
    auto,visibility, compression,
    segment_created, status
FROM user_indexes
WHERE auto='YES';
```

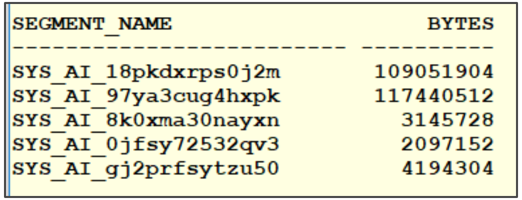
Automatic Indexes



```
TABLE_NAME           INDEX_NAME                 COLUMN_NAME     COLUMN_POSITION
CUSTOMER             SYS_AI_0jfsy72532qv3       C_LAST                        1

CUSTOMER             SYS_AI_8k0xma30nayxn       C_ID                          1
CUSTOMER             SYS_AI_8k0xma30nayxn       C_D_ID                        2
CUSTOMER             SYS_AI_8k0xma30nayxn       C_W_ID                        3

CUSTOMER             SYS_AI_a3tc4dj87650q       C_W_ID                        1

CUSTOMER             SYS_AI_gj2prfsytzu50       C_D_ID                        1
CUSTOMER             SYS_AI_gj2prfsytzu50       C_W_ID
CUSTOMER             SYS_AI_gj2prfsytzu50       C_LAST

ORDERS               SYS_AI_18pkdxrps0j2m       O_ID
ORDERS               SYS_AI_18pkdxrps0j2m       O_W_ID
ORDERS               SYS_AI_18pkdxrps0j2m       O_D_ID

ORDERS               SYS_AI_3ys7c39vs247p       O_D_ID

ORDERS               SYS_AI_81dnzcja2qhpx       O_W_ID
ORDERS               SYS_AI_81dnzcja2qhpx       O_D_ID
ORDERS               SYS_AI_81dnzcja2qhpx       O_C_ID                        3

ORDERS               SYS_AI_97ya3cug4hxpk       O_C_ID                        1
ORDERS               SYS_AI_97ya3cug4hxpk       O_ID                          2

STOCK                SYS_AI_fdbazxb641kwv       S_W_ID                        1
```



```
  1  select index_name,table_name, auto,visibility,segment_created
  2  from user_indexes
  3  where auto='YES'
  4* and visibility = 'VISIBLE'
SQL> /

INDEX_NAME                    TABLE_NAME                AUT VISIBILIT SEG
----------------------------- ------------------------- --- --------- ---
SYS_AI_8k0xma30nayxn          CUSTOMER                  YES VISIBLE   YES
SYS_AI_0jfsy72532qv3          CUSTOMER                  YES VISIBLE   YES
SYS_AI_97ya3cug4hxpk          ORDERS                    YES VISIBLE   YES
```

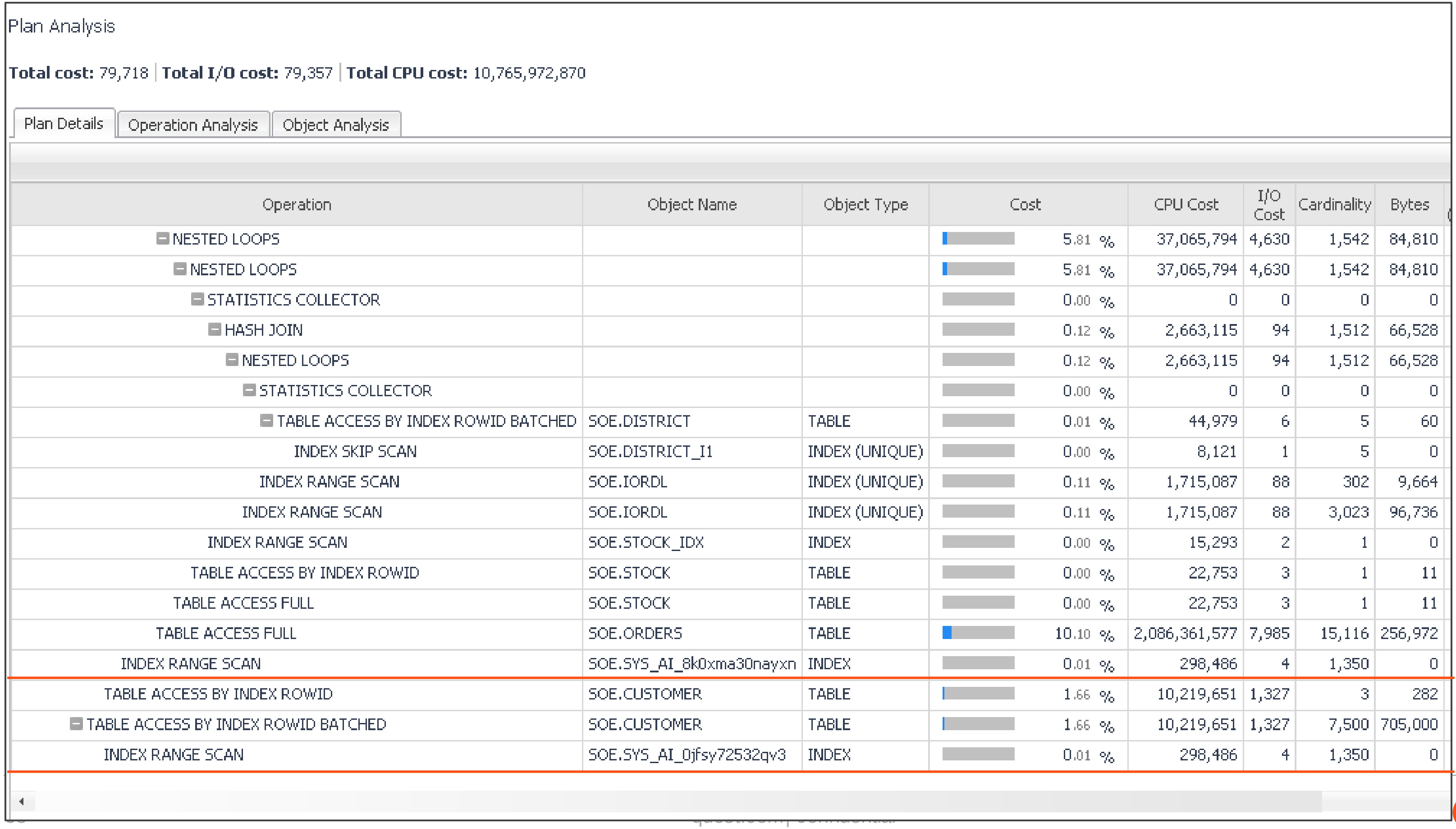


```
SEGMENT_NAME                  BYTES
------------------------- ----------
SYS_AI_18pkdxrps0j2m       109051904
SYS_AI_97ya3cug4hxpk       117440512
SYS_AI_8k0xma30nayxn         3145728
SYS_AI_0jfsy72532qv3         2097152
SYS_AI_gj2prfsytzu50         4194304
```

Total Space: 225m

Visible: 9m

New Execution Plan

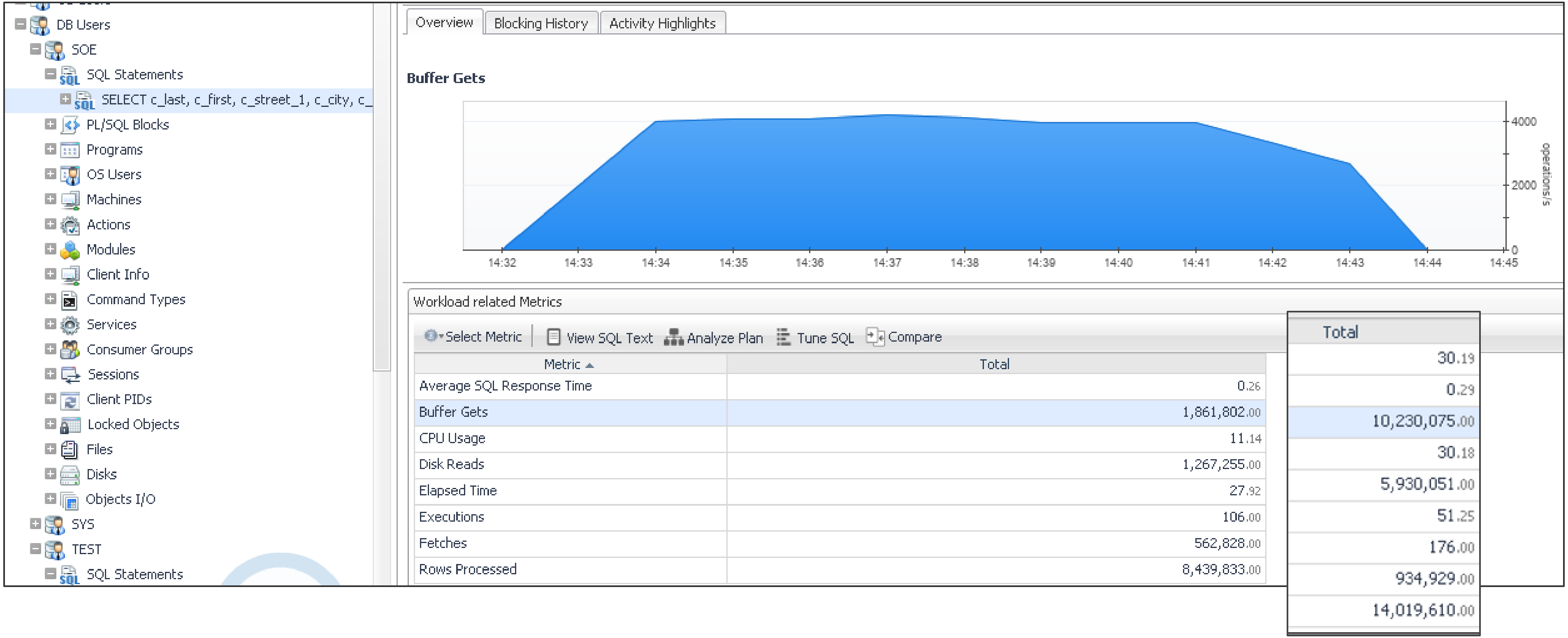


Plan Analysis

Total cost: 79,718 | **Total I/O cost:** 79,357 | **Total CPU cost:** 10,765,972,870

Operation	Object Name	Object Type	Cost	CPU Cost	I/O Cost	Cardinality	Bytes
⊟ NESTED LOOPS			5.81 %	37,065,794	4,630	1,542	84,810
⊟ NESTED LOOPS			5.81 %	37,065,794	4,630	1,542	84,810
⊟ STATISTICS COLLECTOR			0.00 %	0	0	0	0
⊟ HASH JOIN			0.12 %	2,663,115	94	1,512	66,528
⊟ NESTED LOOPS			0.12 %	2,663,115	94	1,512	66,528
⊟ STATISTICS COLLECTOR			0.00 %	0	0	0	0
⊟ TABLE ACCESS BY INDEX ROWID BATCHED	SOE.DISTRICT	TABLE	0.01 %	44,979	6	5	60
INDEX SKIP SCAN	SOE.DISTRICT_I1	INDEX (UNIQUE)	0.00 %	8,121	1	5	0
INDEX RANGE SCAN	SOE.IORDL	INDEX (UNIQUE)	0.11 %	1,715,087	88	302	9,664
INDEX RANGE SCAN	SOE.IORDL	INDEX (UNIQUE)	0.11 %	1,715,087	88	3,023	96,736
INDEX RANGE SCAN	SOE.STOCK_IDX	INDEX	0.00 %	15,293	2	1	0
TABLE ACCESS BY INDEX ROWID	SOE.STOCK	TABLE	0.00 %	22,753	3	1	11
TABLE ACCESS FULL	SOE.STOCK	TABLE	0.00 %	22,753	3	1	11
TABLE ACCESS FULL	SOE.ORDERS	TABLE	10.10 %	2,086,361,577	7,985	15,116	256,972
INDEX RANGE SCAN	SOE.SYS_AI_8k0xma30nayxn	INDEX	0.01 %	298,486	4	1,350	0
TABLE ACCESS BY INDEX ROWID	SOE.CUSTOMER	TABLE	1.66 %	10,219,651	1,327	3	282
⊟ TABLE ACCESS BY INDEX ROWID BATCHED	SOE.CUSTOMER	TABLE	1.66 %	10,219,651	1,327	7,500	705,000
INDEX RANGE SCAN	SOE.SYS_AI_0jfsy72532qv3	INDEX	0.01 %	298,486	4	1,350	0

Performance



- DB Users
  - SOE
    - SQL Statements
      - SELECT c_last, c_first, c_street_1, c_city, c_
    - PL/SQL Blocks
    - Programs
    - OS Users
    - Machines
    - Actions
    - Modules
    - Client Info
    - Command Types
    - Services
    - Consumer Groups
    - Sessions
    - Client PIDs
    - Locked Objects
    - Files
    - Disks
    - Objects I/O
  - SYS
  - TEST
    - SQL Statements

Metric ▲	Total
Average SQL Response Time	0.26
Buffer Gets	1,861,802.00
CPU Usage	11.14
Disk Reads	1,267,255.00
Elapsed Time	27.92
Executions	106.00
Fetches	562,828.00
Rows Processed	8,439,833.00

Total
30.19
0.29
10,230,075.00
30.18
5,930,051.00
51.25
176.00
934,929.00
14,019,610.00

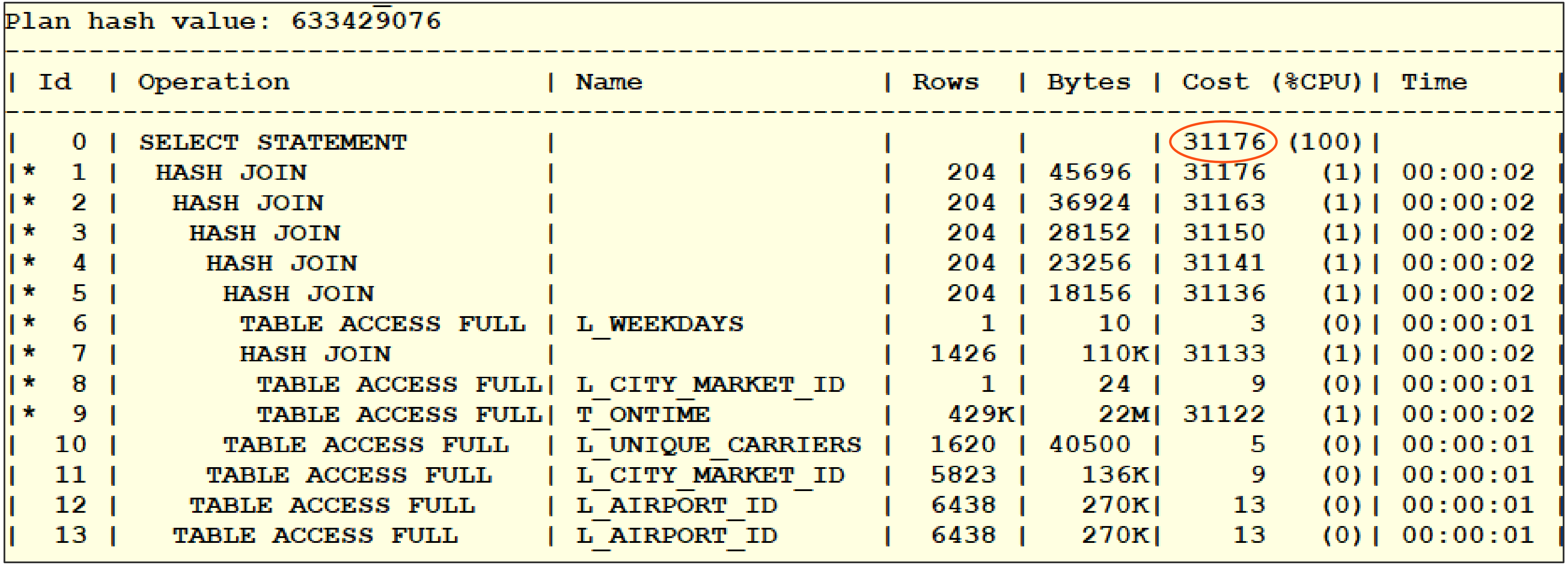
Popular Airline Flights in USA

```
SELECT
o.carrier, uc.description AS carrier_name
,ao.description AS origin_airport,co.Description AS origin_city
,o.fl_date,o.fl_num,o.tail_num
,ad.description AS destination_airport
,cd.Description AS destination_city ,w.Description Day_of_Week
FROM t_ontime o
     INNER JOIN L_UNIQUE_CARRIERS  uc ON uc.Code = o.UNIQUE_CARRIER
     INNER JOIN L_AIRPORT_ID  ao ON ao.Code = o.ORIGIN_AIRPORT_ID
     INNER JOIN L_AIRPORT_ID  ad ON ad.Code = o.DEST_AIRPORT_ID
     INNER JOIN L_CITY_MARKET_ID  co ON co.Code = o.ORIGIN_CITY_MARKET_ID
     INNER JOIN L_CITY_MARKET_ID  cd ON cd.Code = o.DEST_CITY_MARKET_ID
     INNER JOIN L_WEEKDAYS w ON w.Code = o.DAY_OF_WEEK
WHERE to_date(fl_date,'YYYY-MM-DD')  BETWEEN &beg_date AND &end_date
AND co.Description = &city
AND w.Description = &day_of_week;
```

L_UNIQUE_CARRIERS:	1620
L_AIRPORT_ID:	6438
L_CITY_MARKET_ID:	5823
L_WEEKDAYS:	8
T_ONTIME:	6784044

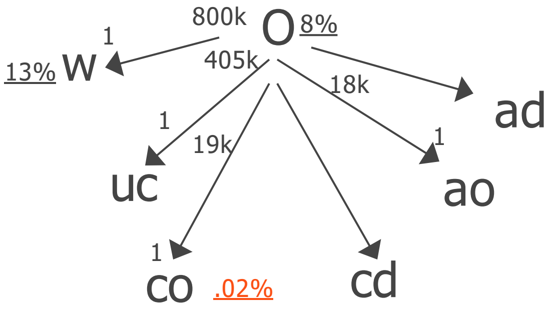
US DOT - On-time Performance

No Other Option but Full Table Scans



```
Plan hash value: 633429076

----------------------------------------------------------------------------------------------
| Id  | Operation                | Name                | Rows  | Bytes | Cost (%CPU)| Time     |
----------------------------------------------------------------------------------------------
|   0 | SELECT STATEMENT         |                     |       |       | 31176 (100)|          |
|*  1 |  HASH JOIN               |                     |   204 | 45696 | 31176   (1)| 00:00:02 |
|*  2 |   HASH JOIN              |                     |   204 | 36924 | 31163   (1)| 00:00:02 |
|*  3 |    HASH JOIN             |                     |   204 | 28152 | 31150   (1)| 00:00:02 |
|*  4 |     HASH JOIN            |                     |   204 | 23256 | 31141   (1)| 00:00:02 |
|*  5 |      HASH JOIN           |                     |   204 | 18156 | 31136   (1)| 00:00:02 |
|*  6 |       TABLE ACCESS FULL  | L_WEEKDAYS          |     1 |    10 |     3   (0)| 00:00:01 |
|*  7 |       HASH JOIN          |                     |  1426 |   110K| 31133   (1)| 00:00:02 |
|*  8 |        TABLE ACCESS FULL | L_CITY_MARKET_ID    |     1 |    24 |     9   (0)| 00:00:01 |
|*  9 |        TABLE ACCESS FULL | T_ONTIME            |   429K|    22M| 31122   (1)| 00:00:02 |
|  10 |      TABLE ACCESS FULL   | L_UNIQUE_CARRIERS   |  1620 | 40500 |     5   (0)| 00:00:01 |
|  11 |     TABLE ACCESS FULL    | L_CITY_MARKET_ID    |  5823 |   136K|     9   (0)| 00:00:01 |
|  12 |    TABLE ACCESS FULL     | L_AIRPORT_ID        |  6438 |   270K|    13   (0)| 00:00:01 |
|  13 |   TABLE ACCESS FULL      | L_AIRPORT_ID        |  6438 |   270K|    13   (0)| 00:00:01 |
```

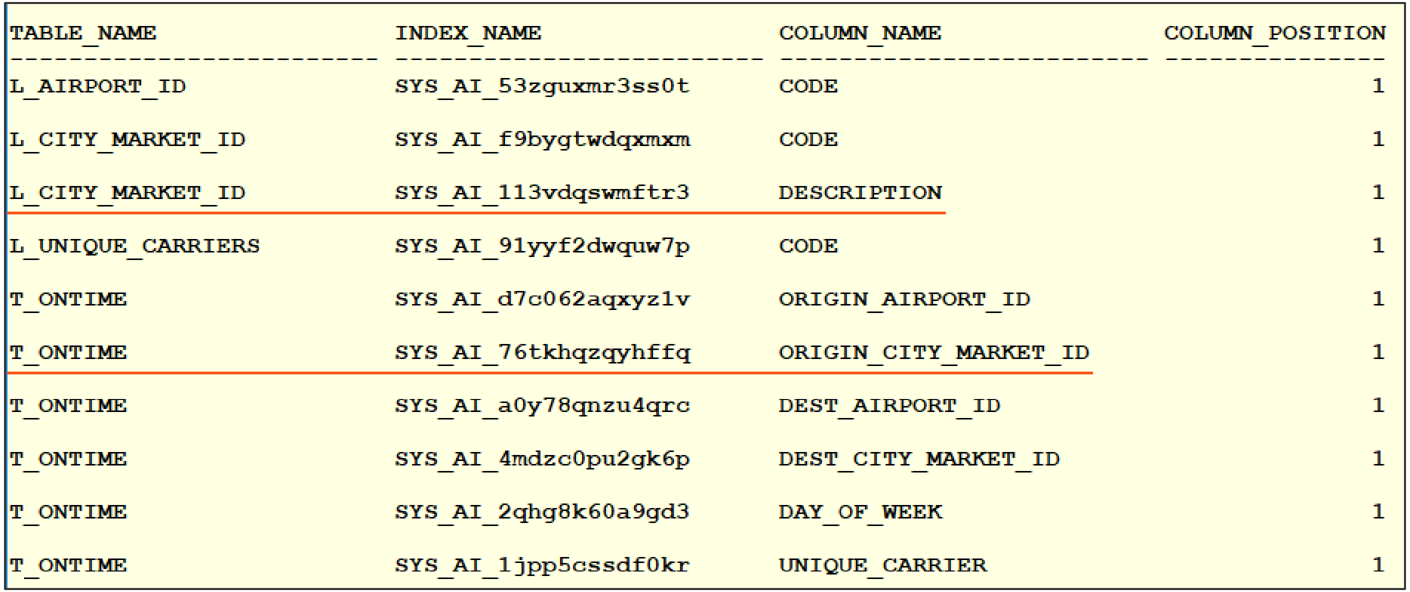
Find the Driving Table



Filtering Selectivity

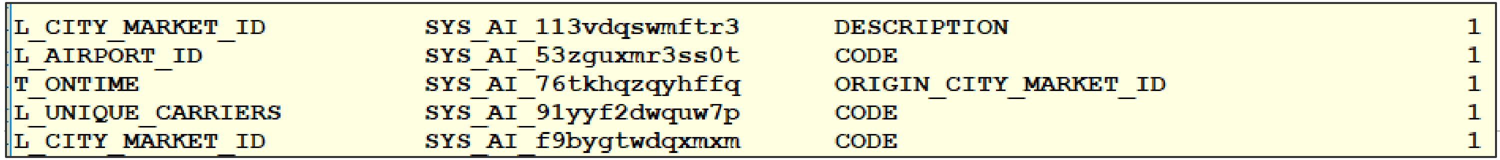
```
select count(1) from t_ontime where fl_date
          between '2015-12-01 00:00:00.000' and'2015-12-31 00:00:00.000‘;
select 479230.00 / 5819067.00 * 100 = 8.23

select count(1) from L_CITY_MARKET_ID where description = 'Chicago, IL'
select 1.00 / 5760.00 * 100 = 0.017

select count(*) from L_WEEKDAYS where description = 'Friday'
select 1.00 / 8 * 100 = 12.50
```

Automatic Indexes



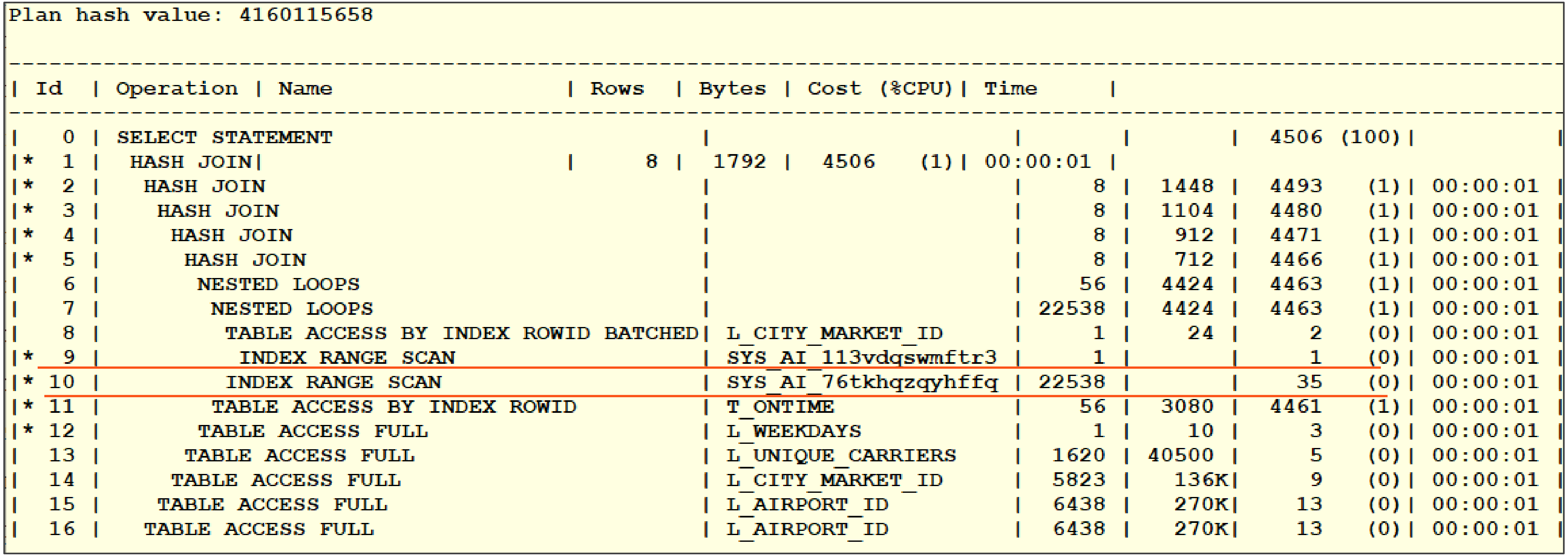
```
TABLE_NAME                 INDEX_NAME                 COLUMN_NAME                COLUMN_POSITION
-------------------------- -------------------------- -------------------------- ---------------
L_AIRPORT_ID               SYS_AI_53zguxmr3ss0t       CODE                                     1

L_CITY_MARKET_ID           SYS_AI_f9bygtwdqxmxm       CODE                                     1

L_CITY_MARKET_ID           SYS_AI_113vdqswmftr3       DESCRIPTION                              1

L_UNIQUE_CARRIERS          SYS_AI_91yyf2dwquw7p       CODE                                     1

T_ONTIME                   SYS_AI_d7c062aqxyz1v       ORIGIN_AIRPORT_ID                        1

T_ONTIME                   SYS_AI_76tkhqzqyhffq       ORIGIN_CITY_MARKET_ID                    1

T_ONTIME                   SYS_AI_a0y78qnzu4qrc       DEST_AIRPORT_ID                          1

T_ONTIME                   SYS_AI_4mdzc0pu2gk6p       DEST_CITY_MARKET_ID                      1

T_ONTIME                   SYS_AI_2qhg8k60a9gd3       DAY_OF_WEEK                              1

T_ONTIME                   SYS_AI_1jpp5cssdf0kr       UNIQUE_CARRIER                           1
```

- Visible Indexes



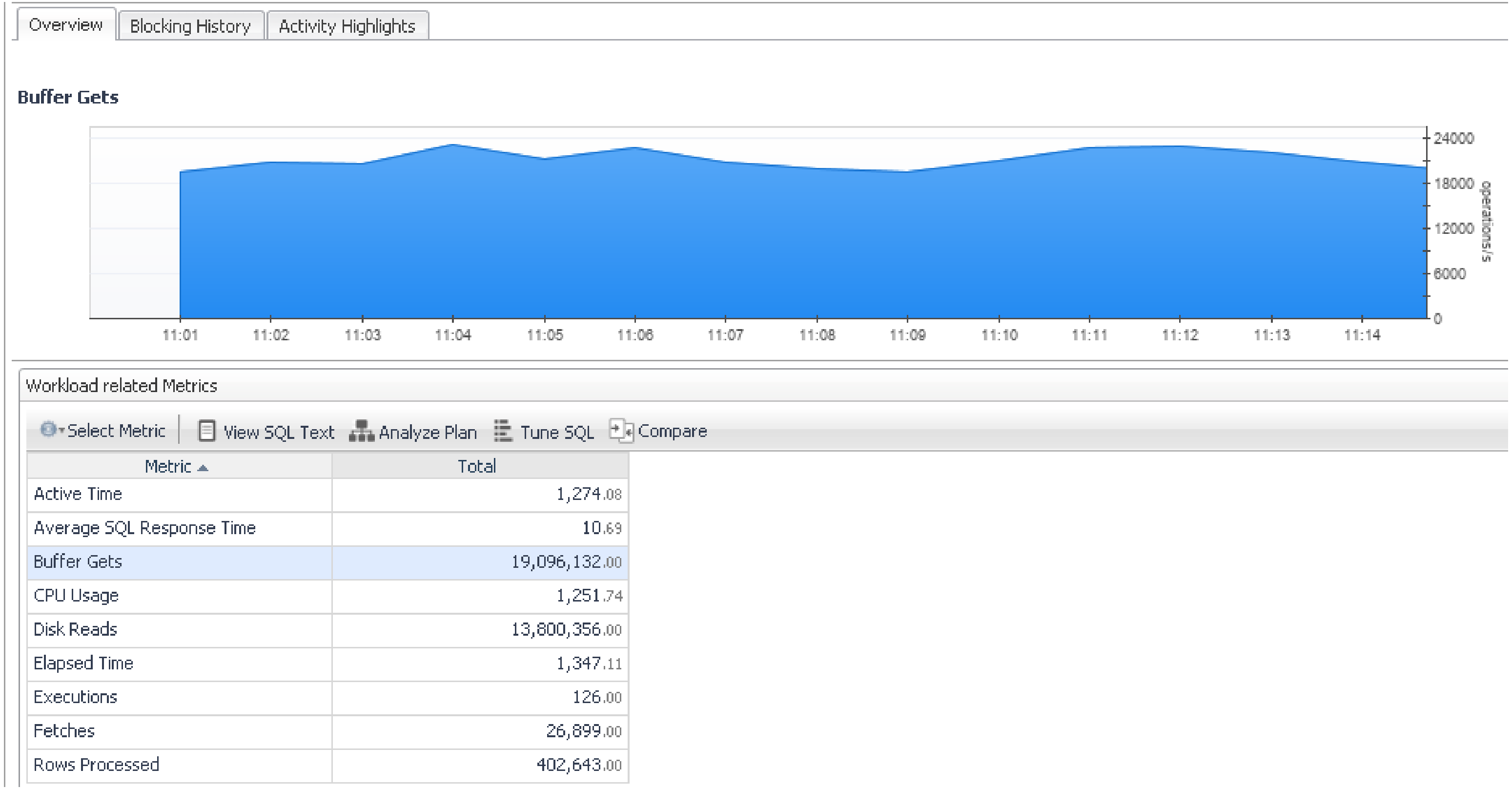
```
L_CITY_MARKET_ID           SYS_AI_113vdqswmftr3       DESCRIPTION                              1
L_AIRPORT_ID               SYS_AI_53zguxmr3ss0t       CODE                                     1
T_ONTIME                   SYS_AI_76tkhqzqyhffq       ORIGIN_CITY_MARKET_ID                    1
L_UNIQUE_CARRIERS          SYS_AI_91yyf2dwquw7p       CODE                                     1
L_CITY_MARKET_ID           SYS_AI_f9bygtwdqxmxm       CODE                                     1
```

New Plan



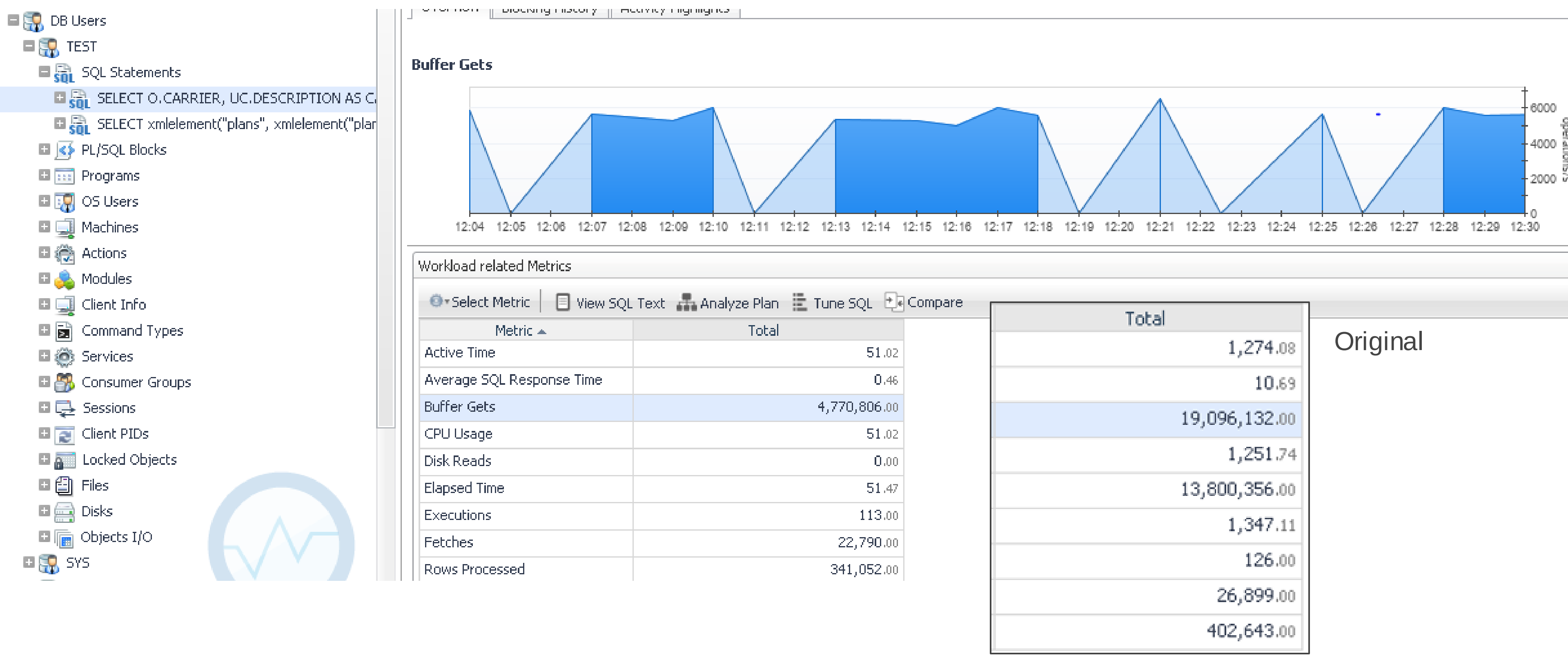
```
Plan hash value: 4160115658

-------------------------------------------------------------------------------------------------------------------
| Id  | Operation | Name                 | Rows  | Bytes | Cost (%CPU)| Time     |
-------------------------------------------------------------------------------------------------------------------
|   0 | SELECT STATEMENT                           |                      |       |       |  4506 (100)|          |
|*  1 |  HASH JOIN|                      |     8 |  1792 |  4506   (1)| 00:00:01 |
|*  2 |   HASH JOIN                                |                      |     8 |  1448 |  4493   (1)| 00:00:01 |
|*  3 |    HASH JOIN                               |                      |     8 |  1104 |  4480   (1)| 00:00:01 |
|*  4 |     HASH JOIN                              |                      |     8 |   912 |  4471   (1)| 00:00:01 |
|*  5 |      HASH JOIN                             |                      |     8 |   712 |  4466   (1)| 00:00:01 |
|   6 |       NESTED LOOPS                         |                      |    56 |  4424 |  4463   (1)| 00:00:01 |
|   7 |        NESTED LOOPS                        |                      | 22538 |  4424 |  4463   (1)| 00:00:01 |
|   8 |         TABLE ACCESS BY INDEX ROWID BATCHED| L_CITY_MARKET_ID     |     1 |    24 |     2   (0)| 00:00:01 |
|*  9 |          INDEX RANGE SCAN                  | SYS_AI_113vdqswmftr3 |     1 |       |     1   (0)| 00:00:01 |
|* 10 |         INDEX RANGE SCAN                   | SYS_AI_76tkhqzqyhffq | 22538 |       |    35   (0)| 00:00:01 |
|* 11 |        TABLE ACCESS BY INDEX ROWID         | T_ONTIME             |    56 |  3080 |  4461   (1)| 00:00:01 |
|* 12 |       TABLE ACCESS FULL                    | L_WEEKDAYS           |     1 |    10 |     3   (0)| 00:00:01 |
|  13 |      TABLE ACCESS FULL                     | L_UNIQUE_CARRIERS    |  1620 | 40500 |     5   (0)| 00:00:01 |
|  14 |     TABLE ACCESS FULL                      | L_CITY_MARKET_ID     |  5823 |   136K|     9   (0)| 00:00:01 |
|  15 |    TABLE ACCESS FULL                       | L_AIRPORT_ID         |  6438 |   270K|    13   (0)| 00:00:01 |
|  16 |   TABLE ACCESS FULL                        | L_AIRPORT_ID         |  6438 |   270K|    13   (0)| 00:00:01 |
```

Original Performance



Metric ▲	Total
Active Time	1,274.08
Average SQL Response Time	10.69
Buffer Gets	19,096,132.00
CPU Usage	1,251.74
Disk Reads	13,800,356.00
Elapsed Time	1,347.11
Executions	126.00
Fetches	26,899.00
Rows Processed	402,643.00

Auto Index Performance



- DB Users
  - TEST
    - SQL Statements
      - SELECT O.CARRIER, UC.DESCRIPTION AS C
      - SELECT xmlelement("plans", xmlelement("plar
    - PL/SQL Blocks
    - Programs
    - OS Users
    - Machines
    - Actions
    - Modules
    - Client Info
    - Command Types
    - Services
    - Consumer Groups
    - Sessions
    - Client PIDs
    - Locked Objects
    - Files
    - Disks
    - Objects I/O
  - SYS

Buffer Gets

Workload related Metrics

Metric	Total	Total
Active Time	51.02	1,274.08
Average SQL Response Time	0.46	10.69
Buffer Gets	4,770,806.00	19,096,132.00
CPU Usage	51.02	1,251.74
Disk Reads	0.00	13,800,356.00
Elapsed Time	51.47	1,347.11
Executions	113.00	126.00
Fetches	22,790.00	26,899.00
Rows Processed	341,052.00	402,643.00

Original

Summary

- There are a lot of challenges in Query Tuning
- If you remember the Top 5 Tips, they should take you a long way
  - 1. Monitor Wait time
    - Look at wait events, record baseline metrics
  - 2. Review the Execution Plan
    - Look for expensive steps, know what's optimizer features are supporting the plan
  - 3. Gather Object Information
    - For expensive objects – know what the optimizer knows
  - 4. Find the Driving Table
    - Consider SQL Diagramming techniques
    - If you have the Tuning & Diagnostic Packs, check out the Tuning Advisor
    - Use 19c Automatic Indexing in dev/test
  - 5. Engineer out the Stupid

Questions?

Quest®

Thank you

Quest®