VISCOSITY
PROFESSIONAL SERVICES

ORAPUB
POWERED BY VISCOSITY



Oracle Adaptive SQL Optimization
Why do things go so bad?

NLOUG
Nov 13, 2026

NL.OUG
NL ORACLE USER GROUP

Gary Gordhamer

Oracle ACE Director
Managing Principal Consultant

@ggordham.bsky.social
linkedin.com/in/ggordhamer/
gary.gordhamer@viscosityna.com



Gary Gordhamer

Oracle ACE Director
Managing Principal Consultant

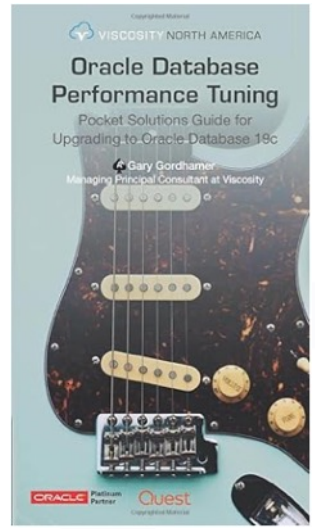
@ggordham.bsky.social
linkedin.com/in/ggordhamer/
gary.gordhamer@viscosityna.com

30+ years with Oracle technology

- Database 6.x – 23c
- Oracle Financials 7, E-Business Suite R11, R12.1, R12.2
- IDM integrations and Security hardening

Worked in many industries including

- manufacturing
- financial
- marketing
- healthcare
- utilities
- Government



Wrote the book:
“Oracle Database Performance Tuning: Pocket Solution Guide Series For Upgrading Oracle Databases”

Active member of the Oracle user community presenting frequently!

Oracle ACE Director

Viscosity's Oracle ACEs

The Oracle ACE Program

The Oracle ACE Program recognizes and rewards individuals for their contributions to the Oracle community.



Charles Kim
CEO | Co-Founder

@racdba
ACE Director



Rich Niemiec
Chief Innovation Officer

@richniemiec
ACE Director



Craig Shallahamer
Applied AI Scientist

@orapub
ACE Director



Sean Scott
Principal Consultant

oraclesean.com
ACE Director



Gary Gordhamer
Principal Consultant

ggordham.bsky.social
ACE Director



Marco Pereira
Software Architect

@markuspg
ACE Associate

viscosityna.com

@ViscosityNa

Want to try this out?

DEMO

You can download all the scripts from GitHub
https://github.com/ggordham/ora-presentations/tree/main/adaptive-sql-NLOUG

All scripts are ”as is” and meant for teaching purposes only.

DO NOT run these in any production system or system relied upon by your development teams.

All steps have been set to run as the created PERFLAB user or SYS

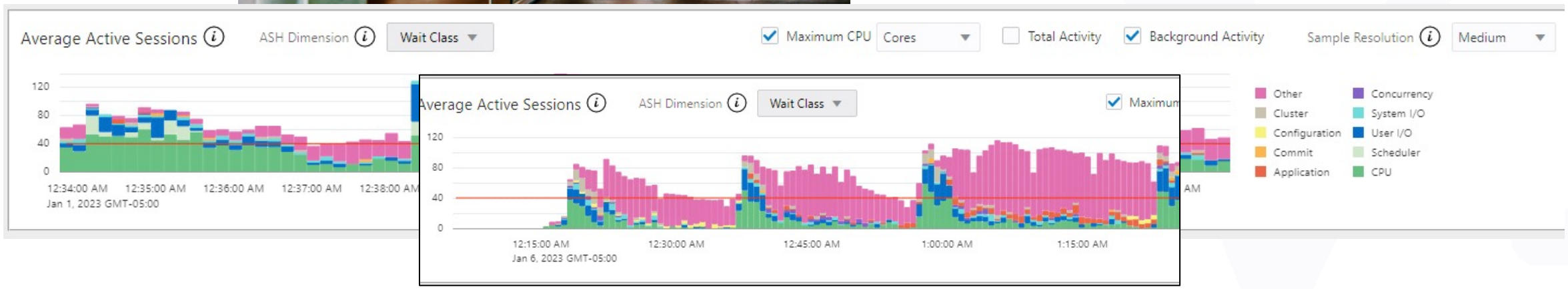


Getting alerts about production CPU usage!

Users are complaining about application not responding

Batch jobs from last night have not completed yet

Can you join a Tcon?



viscosityna.com

@ViscosityNa

What went wrong?

After digging a bit you find a single SQL with multiple plans:

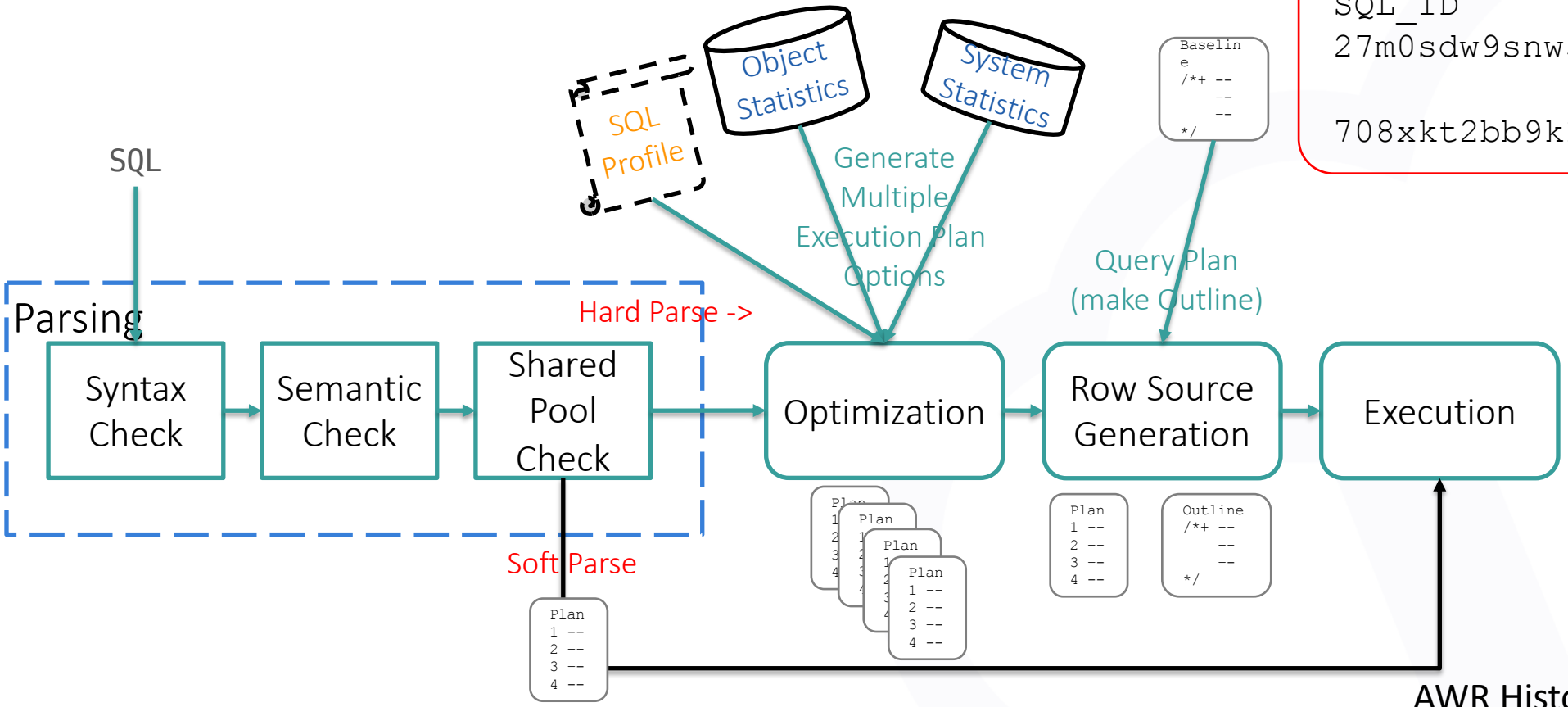
Plan Hash Value	Elap Secs Per Exec	CPU Secs Per Exec	EXECUTIONS	BUFFERS_PER_EXEC	Rows Per Exec
1113757002	0.136838	0.135079	2726	26640	54.314
2580656749	0.165411	0.162940	15306	32674	90.336
3307026250	0.210900	0.208088	14899	42973	109.221
3217703047	0.219462	0.216613	3539	43202	106.234
2498874158	0.352315	0.350418	97468	17067	212.221
916664136	0.417382	0.416215	5154	86204	297.749
792194355	0.562100	0.559277	13789	91089	386.776
3866860724	0.586831	0.585566	5348	106758	400.036
3463742556	0.599296	0.597686	1214	108038	629.894
3043904905	0.616232	0.614801	5250	108697	697.009
351032388	0.629082	0.627445	3892	109800	858.396
2208551890	0.710137	0.707462	21528	117356	1,769.225
3081761355	0.994707	0.992761	18550	145157	929.941
242056001	1.033102	1.030417	18808	147599	1,337.955
2193761257	1.059029	1.057118	1545	147304	1,303.657
3562434294	1.120287	1.116986	18477	152373	2,182.843

Why?

- Why so many execution plans?
- Why did the Optimizer change?
- Were these good plans or bad plans?
- What should I look at / for?

Optimizer Basics

Optimizer SQL Steps



Cursor Cache (few hours?)

SQL_ID	Text	Plan
27m0sdw9snw59	SELECT…	Plan 1 -- 2 --
708xkt2bb9k1f	SELECT…	Plan 1 -- 2 --

AWR History (Top 30?)

SQL_ID	Text	Plan
5nc2ws664t1gx	SELECT…	Plan 1 -- 2 --
708xkt2bb9k1f	SELECT…	Plan 1 -- 2 --

Executions plans exist in cursor cache until aged out and in AWR history if they are in top 30 resource consumers.

Note: AWR could be more than 30 if you have modified the setting.

Cardinality

- Estimate of the number of rows that will be returned
- Influences the Optimizer decision
  - Cost of lookup
  - How to do a lookup

$$\frac{\text{Total Number of Rows}}{\text{Number of Distinct Values}} = \text{Cardinality}$$

Example (Equals =)

- 500,000 row table, a column has 500,000 distinct values
  - Cardinality = 1
- 500,000 row table, a column has 253,072 distinct values
  - Cardinality = ~ 2 (1.975)

Stats and cardinality

```
SELECT COUNT('x')
   FROM sh.sales s, sh.customers c
   WHERE s.cust_id =c.cust_id AND c.social_id = 'XXXXXX';
```

```
TABLE_NAME    NUM_ROWS    BLOCKS AVG_ROW_LEN LAST_ANALYZED
---------- ---------- ---------- ----------- -------------------
CUSTOMERS      55500       1551         195 2024-11-07 17:17:27
SALES         918843       1874          29 2024-10-15 05:00:18
```

```
COLUMN_NAME DATA_TYPE LENGTH NUM_DISTINCT LOW_VALUE HIGH_VALUE    DENSITY NULLS HISTOGRAM LAST_ANALYZED
----------- --------- ------ ------------ --------- ---------- ---------- ----- --------- -------------------
SOCIAL_ID   VARCHAR2      10        51844 1         XXXXXX     .000019289     0 NONE      2024-11-07 17:17:27
```

- Equals predicate (so we are looking for one value)
- Table has 55,500 rows, column has 51,844 distinct values
- Density –
  - Close to 1 – unselective – full table scan
  - Close to 0 – highly selective – use an index (if available)
  - If a histogram exists – only use density for non-popular values

Cardinality computation - selectivity

```
SELECT COUNT('x')
   FROM sh.sales s, sh.customers c
   WHERE s.cust_id =c.cust_id AND c.social_id = 'XXXXXX';
```

```
-----------------------------------------------------------------------------------------
|Id | Operation                    | Name           | Rows  | Bytes | Cost (%CPU)| Time     |
-----------------------------------------------------------------------------------------
| 0 | SELECT STATEMENT             |                |       |       |   452 (100)|          |
| 1 |  SORT AGGREGATE              |                |     1 |    16 |            |          |
| 2 |   NESTED LOOPS               |                |   139 |  2224 |   452   (1)| 00:00:01 |
|*3 |    TABLE ACCESS FULL         | CUSTOMERS      |     1 |    11 |   424   (1)| 00:00:01 |
| 4 |    PARTITION RANGE ALL       |                |   130 |   650 |   452   (1)| 00:00:01 |
| 5 |     BITMAP CONVERSION COUNT  |                |   130 |   650 |   452   (1)| 00:00:01 |
|*6 |      BITMAP INDEX SINGLE VALUE| SALES_CUST_BIX |       |       |            |          |
-----------------------------------------------------------------------------------------
```

```
 Table: CUSTOMERS  Alias:  C  #Rows: 55500  #Blks:  1551  AvgRowLen:  195.00
 Column (#24): SOCIAL_ID(VARCHAR2)
    AvgLen: 6 NDV: 51844 Nulls: 0 Density: 0.000019
Estimated selectivity: 1.9289e-05 , col: #24
Card: Original: 55500.000000  Rounded: 1  Computed: 1.070519
```

0.000019289% = (1.070519 / 55500)

Number of expected rows (end points)

55500/51844*1

Join cardinality

TABLE_NAME	NUM_ROWS	BLOCKS	AVG_ROW_LEN	LAST_ANALYZED
CUSTOMERS	55500	1551	195	2024-11-07 17:17:27
SALES	918843	1874	29	2024-10-15 05:00:18

COLUMN_NAME	DATA_TYPE	LENGTH	NUM_DISTINCT	LOW_VALUE	HIGH_VALUE	DENSITY	NULLS	HISTOGRAM	LAST_ANALYZED
CUST_ID	NUMBER	22	7059	2	101000	.000141663	0	NONE	2024-10-15 05:00:18

- Cardinality for SALES join to CUSTOMERS
- Customer number of expected rows = 1
- SALES calculation
  - 918,843 rows / 7,059 distinct = 130 (rows per customer)
  - Customer rows 1 * 130 (rows per customer
- SALES number of expected rows = 130, use index

Optimizer Estimate vs Reality

Id	Operation	Name	Starts	E-Rows	E-Bytes	Cost (%CPU)	A-Rows	A-Time	Buffers	Reads
0	SELECT STATEMENT		1			553 (100)	1	00:00:00.04	1978	1976
1	SORT AGGREGATE		1	1	22		1	00:00:00.04	1978	1976
2	NESTED LOOPS		1	1	22	553 (1)	1	00:00:00.04	1978	1976
3	NESTED LOOPS		1	1	22	553 (1)	1	00:00:00.04	1977	1975
* 4	TABLE ACCESS FULL	T1	1	1	13	550 (1)	1	00:00:00.04	1974	1972
* 5	INDEX RANGE SCAN	T2I	1	1		2 (0)	1	00:00:00.01	3	3
6	TABLE ACCESS BY INDEX ROWID	T2	1	1	9	3 (0)	1	00:00:00.01	1	1

Id	Operation	Name	Starts	E-Rows	E-Bytes	Cost (%CPU)	A-Rows	A-Time	Buffers	Reads
0	SELECT STATEMENT		1			553 (100)	1	00:00:01.00	11089	4536
1	SORT AGGREGATE		1	1	22		1	00:00:01.00	11089	4536
2	NESTED LOOPS		1	1	22	553 (1)	250K	00:00:00.93	11089	4536
3	NESTED LOOPS		1	1	22	553 (1)	250K	00:00:00.36	9118	2566
* 4	TABLE ACCESS FULL	T1	1	1	13	550 (1)	250K	00:00:00.05	1976	1971
* 5	INDEX RANGE SCAN	T2I	250K	1		2 (0)	250K	00:00:00.34	7142	595
6	TABLE ACCESS BY INDEX ROWID	T2	250K	1	9	3 (0)	250K	00:00:00.45	1971	1970

Same Query, Different Bind Value

```
-----------------------------------------------------------------------------------------------------------------------
| Id  | Operation                      | Name | Starts | E-Rows |E-Bytes| Cost (%CPU)| A-Rows |   A-Time    | Buffers |
-----------------------------------------------------------------------------------------------------------------------
|   0 | SELECT STATEMENT               |      |      1 |        |       |   553 (100)|      1 |00:00:00.01 |    1980 |
|   1 |  SORT AGGREGATE                |      |      1 |      1 |    22 |            |      1 |00:00:00.01 |    1980 |
|   2 |   NESTED LOOPS                 |      |      1 |      1 |    22 |   553   (1)|      1 |00:00:00.01 |    1980 |
|   3 |    NESTED LOOPS                |      |      1 |      1 |    22 |   553   (1)|      1 |00:00:00.01 |    1979 |
|*  4 |     TABLE ACCESS FULL          | T1   |      1 |      1 |    13 |   550   (1)|      1 |00:00:00.01 |    1976 |
|*  5 |     INDEX RANGE SCAN           | T2I  |      1 |      1 |       |     2   (0)|      1 |00:00:00.01 |       3 |
|   6 |    TABLE ACCESS BY INDEX ROWID| T2   |      1 |      1 |     9 |     3   (0)|      1 |00:00:00.01 |       1 |
-----------------------------------------------------------------------------------------------------------------------
```

```
-----------------------------------------------------------------------------------------------------------------------
| Id  | Operation                      | Name | Starts | E-Rows |E-Bytes| Cost (%CPU)| A-Rows |   A-Time    | Buffers |
-----------------------------------------------------------------------------------------------------------------------
|   0 | SELECT STATEMENT               |      |      1 |        |       |   741K(100)|      1 |00:00:00.43 |   11089 |
|   1 |  SORT AGGREGATE                |      |      1 |      1 |    22 |            |      1 |00:00:00.43 |   11089 |
|   2 |   NESTED LOOPS                 |      |      1 |    246K|  5305K|   741K  (1)|    250K|00:00:00.44 |   11089 |
|   3 |    NESTED LOOPS                |      |      1 |    246K|  5305K|   741K  (1)|    250K|00:00:00.30 |    9118 |
|*  4 |     TABLE ACCESS FULL          | T1   |      1 |    246K|  3134K|   550   (1)|    250K|00:00:00.04 |    1976 |
|*  5 |     INDEX RANGE SCAN           | T2I  |    250K|      1 |       |     2   (0)|    250K|00:00:00.18 |    7142 |
|   6 |    TABLE ACCESS BY INDEX ROWID| T2   |    250K|      1 |     9 |     3   (0)|    250K|00:00:00.09 |    1971 |
-----------------------------------------------------------------------------------------------------------------------
```

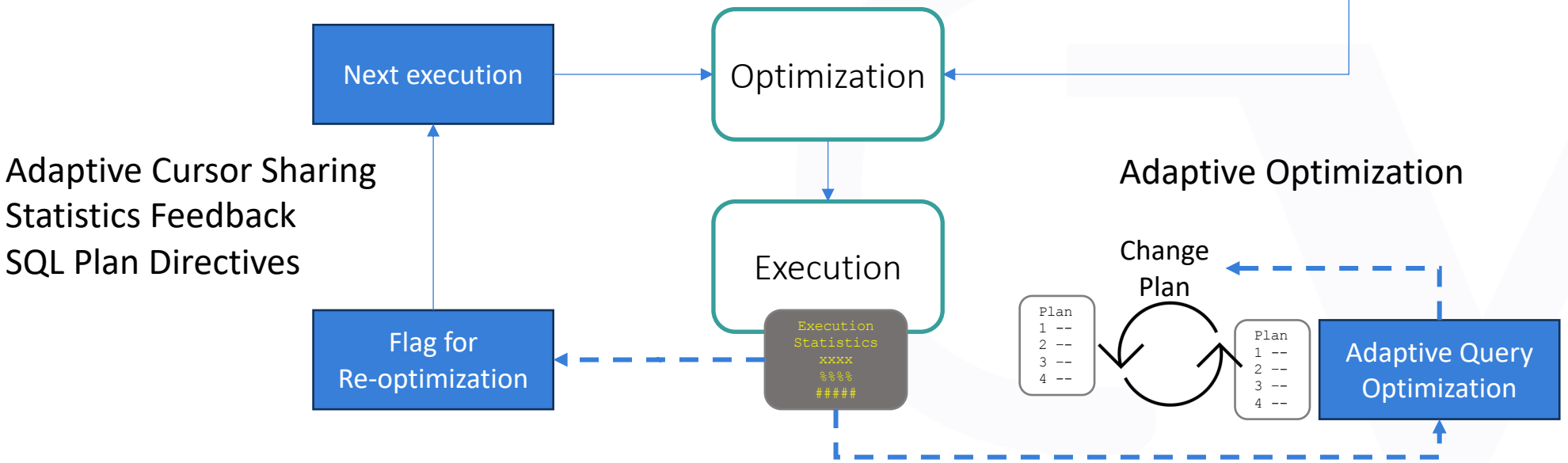
Adaptive Query Optimization

Optimizer Features

Two main categories

- Real time (Adaptive Optimization)
- Next run or learn from previous run

```
SELECT product_name
  FROM oe.order_items o,
       oe.product_information p
 WHERE o.unit_price = :v_price
   AND o.quantity > 1
   AND p.product_id = o.product_id;
```

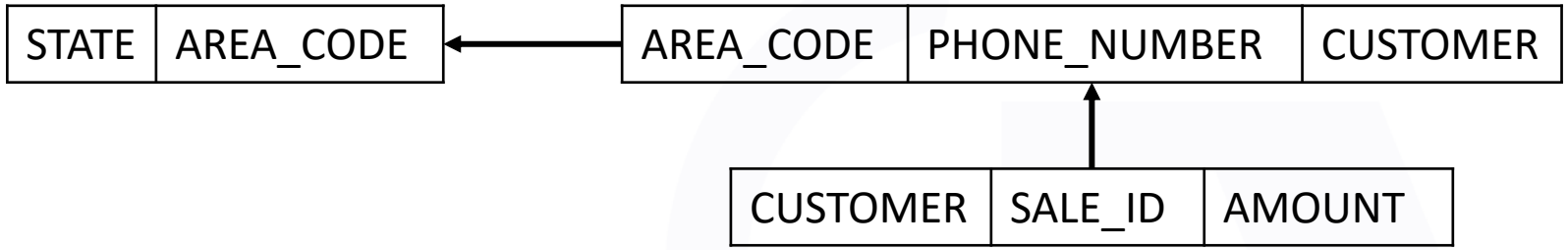


In the old days

The Optimizer would create an execution plan

A statement could have only one plan in memory

What if the query had different data cardinality?



Query to pull total amount sales by state, developed with:

```
V_STATE := 'WY'
```
(Wyoming)

Then someone runs the same query with:

```
V_STATE := 'CA'
```
(California)

Cardinality for these two are dramatically different (13% vs. 0.3%)

Adaptive cursor sharing

Allows the Optimizer to generate a set of plans for the same SQL based on bind values

Why multiple execution plans will show up in `v$sql`

Note: `v$sqlarea` will show only a single statement.

Helps deal with data skew and complex predicates

Can see why each plan was generated by looking at the view:

`v$sql_shared_cursor` (has 64 flags!)

New plans generated based on feedback from previous execution

Previously called “Automatic Reoptimization”

Bind peeking

This statement uses equals with a bind variable:

```
select count(1), max(cust_id)
     from sh.customers
     where cust_state_province = :cust_prov;
```

Statistics:

Rows	55,500
CUST_STATE_PROVINCE (NDV)	145
Estimated Cardinality (=)	383

There is a histogram, but unless we ”look” at the bind value it won’t do us any good.

The Optimizer will “peek” at the value

CUST_PROV = ‘CA’ versus CUST_PROV = ‘Tokyo’

Histogram

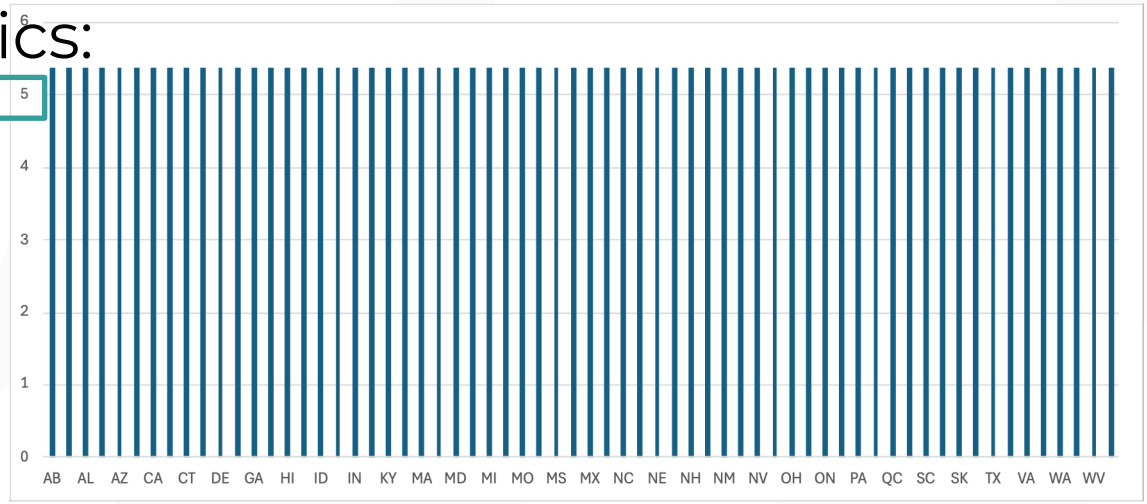
END_POINT	COUNT
CA	3341
FL	2438
Baden-Wuerttemberg	2319
Nordrhein-Westfalen	2274
Noord-Brabant	2170
...	
UT	40
Warsaw	37
Delhi	34
VT	33
Lublin	18
Tokyo	16

What is a Histogram?

DEMO

Oracle normally has basic table statistics:

- Number of rows in the table
- Average size of each row
- Number of distinct values per column



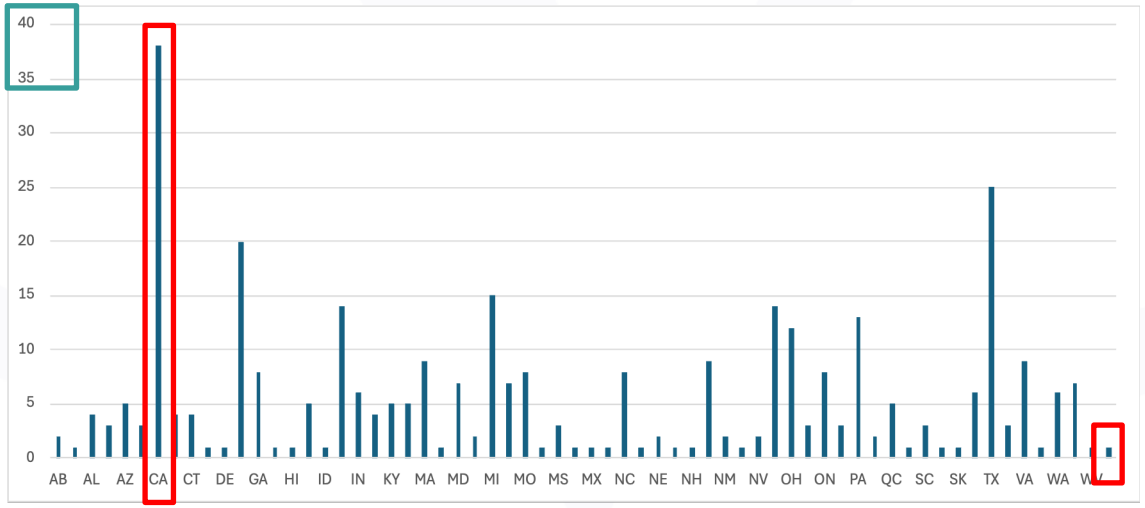
Example without histogram

- 64 states, territories, districts with area codes
- 344 total area codes, so ~ 5 each

Oracle can keep up to 255 ”buckets”

Here row value = CA has 38 rows

Each has their own value,
many very low



Bind Peeking + Adaptive Cursor Sharing

DEMO

When a SQL is re-executed, the Optimizer will review cardinality estimates and see if re-optimization is needed:

- Query must be using bind variables
- Can see if query is bind aware by querying `v$sql` view
- If additional plan is crated due to bind peeking, a secondary flag will be set

When the original plan is used again, a new row in `v$sql` will be generated with the same `PLAN_HASH_VALUE` but this time it will be bind aware. Note the difference in `BUFFER_GETS`

CHILD_NUMBER	PLAN_HASH_VALUE	EXECUTIONS	BUFFER_GETS	IS_BIND_SENSITIVE	IS_BIND_AWARE
0	558307901	2	1486	Y	N
1	270892308	1	156	Y	Y
2	558307901	1	18	Y	Y

Adaptive Cursor Sharing (ACS)

The Optimizer detects that the number of buffer gets has changed

- Amount of buffer changes must be substantial (10x)
- Order of number of buffers changed can make a difference (growing vs shrinking)

Once detected the cursor is marked as “Bind Aware”

- Generates a new cursor (could be the same execution plan)
- Explain plan should show cardinality based on the bind variable value
- Optimizer is more likely to detect another change

Optimizer may not change plans immediately

- Often multiple executions are required before the Optimizer decides the plan should change

Optimizer may not change plans at all!

Huge Thank You to fellow ACE Monika Lewandowska for more ACS details! https://ace.oracle.com/ords/ace/profile/mlewan1115

Additional Notes on ACS

Thresholds for changes are based on there buckets

1 Row (few)	between few & 1,000,000 Rows	> 1,000,000 Rows

- *Note: 1 row bucket is more like very few rows bucket compared to middle bucket (small number of buffers)*

The column of the join or predicate filter must be indexed

- There must be a histogram on the column, this allows the Optimizer to estimate the row counts correctly

Use of an index is still predicated on the percentage of rows being returned from the table

- At hard parse, initial calculations is based on number of IO's to do lookup
- If the data can be pulled from only the index, the Optimizer may use index only lookups (FULL SCAN, FAST FULL SCAN, JOIN SCAN)

Huge Thank You to fellow ACE Monika Lewandowska for more ACS details! https://ace.oracle.com/ords/ace/profile/mlewan1115

Adaptive Optimiztion

Adaptive Query Optimization

Allows the Optimizer to make run time decisions

- Can start executing a query plan
- Based on execution statistics, adjust execution plan
- Adaptive Plans
  - Change Join Method (NL vs HJ)
  - Parallel Distribution Method (Broadcast vs Hybrid hash)
  - Bitmap Index Pruning (for star transformations)
- Adaptive Statistics
  - Dynamic statistics
  - Automatic Re-optimization
  - SQL Plan Directive

Adaptive – Cursor vs plan

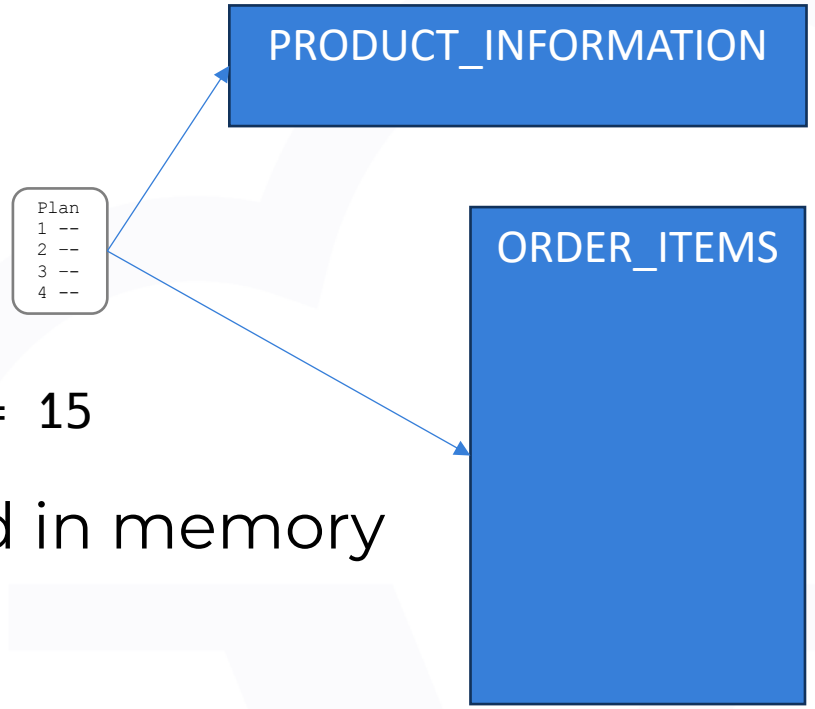
Adaptive cursor sharing is just about having multiple different plans

Adaptive plan is about an execution plan that has multiple options and during run time it could change

Adaptive Plan – join

```
SELECT product_name
  FROM oe.order_items o,
       oe.product_information p
 WHERE o.unit_price = :v_price
   AND o.quantity > 1
   AND p.product_id = o.product_id;
```

`:v_price = 15`



A query is optimized, and a plan is cached in memory

- This plan is ONE way to retrieve the data
- Optimized when first executed based on:
  - o.unit_price, o.quanitity – NDV => cardinality of each table access
  - p.product_id, o.product_id => cardinality of join
- Generates a plan with
  - Type of join between each table -> Nested Loop or Hash Join
  - Type of access for each table –> index lookup or full table scan

What happens when this query re-runs with `:v_price = 211`
Would the cardinality be the same?

Adaptive Plan

DEMO

```
-------------------------------------------------------------------------------------------------------------------
| Id  | Operation                     | Name                     | Starts | E-Rows | Cost (%CPU)| A-Rows | Buffers |
-------------------------------------------------------------------------------------------------------------------
|   0 | SELECT STATEMENT              |                          |      1 |        |     7 (100)|     13 |      23 |
|-  1 |  HASH JOIN                    |                          |      1 |      4 |     7   (0)|     13 |      23 |
|   2 |   NESTED LOOPS                |                          |      1 |      4 |     7   (0)|     13 |      23 |
|   3 |    NESTED LOOPS               |                          |      1 |      4 |     7   (0)|     13 |      10 |
|-  4 |     STATISTICS COLLECTOR      |                          |      1 |        |            |     13 |       6 |
|   5 |      TABLE ACCESS FULL        | ORDER_ITEMS              |      1 |      4 |     3   (0)|     13 |       6 |
|   6 |     INDEX UNIQUE SCAN         | PRODUCT_INFORMATION_PK   |     13 |      1 |     0   (0)|     13 |       4 |
|   7 |    TABLE ACCESS BY INDEX ROWID| PRODUCT_INFORMATION      |     13 |      1 |     1   (0)|     13 |      13 |
|-  8 |   TABLE ACCESS FULL           | PRODUCT_INFORMATION      |      0 |      1 |     1   (0)|      0 |       0 |
-------------------------------------------------------------------------------------------------------------------
   - this is an adaptive plan (rows marked '-' are inactive)
```

Using FORMAT of DBMS_XPLAN

```
FORMAT=>'ALLSTATS LAST +ADAPTIVE'
```

Shows that the Optimizer had two paths in mind

- HASH JOIN + index lookup on PRODUCT_INFORMATION – not used
- NESTED LOOPS + full table scan of PRODUCT_INFORMATION

```
SQL_ID        CHILD_NUMBER PLAN_HASH_VALUE IS_REOPTIMIZABLE IS_RESOLVED_ADAPTIVE_PLAN IS_BIND_AWARE IS_BIND_SENSITIVE
------------- ------------ --------------- ---------------- ------------------------- ------------- -----------------
9swuztstqj7nn            0      1255158658 N                Y                         N             Y
```

Why did the Optimizer choose?

Execution of the plan started

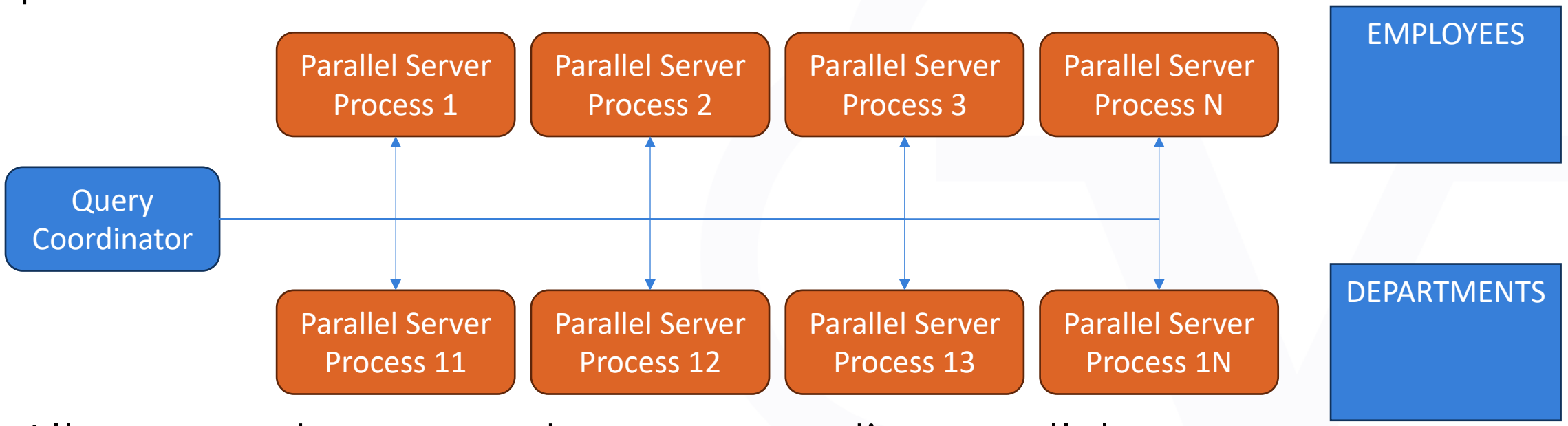
- Collect rows for ORDER_ITEMS
- STATISTICS COLLECTOR watches:
  - Number of rows buffered
- Check internal threshold
  - Greater than threshold -> use HASH JOIN
  - Less than threshold -> use NESTED LOOPS

Adaptive plans are enabled if:

`OPTIMIZER_ADAPTIVE_PLANS = TRUE` (default)
`OPTIMIZER_FEATURES_ENABLE > 12.1.0.1`
`OPTIMIZER_ADAPTIVE_REPORTING_ONLY = FALSE` (default)

Adaptive Plan - Parallel

Parallel queries need a way to communicate between child processes



All rows can be sent to the corresponding parallel process

OR a hash can be sent to the corresponding parallel process

Statistics feedback

Statistics (Cardinality) Feedback

If the execution of the SQL statement is monitored, post execution the Optimizer compares cardinality estimates to actuals.

- Statement is flagged for re-optimization
- Feedback is stored in the SGA as an `OPT_ESTIMATE` hint

If estimates are are significantly different than actuals:

- Actual cardinality information is stored
- SQL Plan directive is created (more in next demo)
- Monitoring is disabled for next run

```
Note
-----
 - statistics feedback used for this statement
```

Statistics feedback

Note that estimates (E-Rows) are way off from actuals (A-Rows)

Id	Operation	Name	E-Rows	E-Bytes	A-Rows
0	SELECT STATEMENT				1
1	SORT AGGREGATE		1	24	1
2	NESTED LOOPS		1	24	269
3	MERGE JOIN CARTESIAN		4	64	9135
* 4	TABLE ACCESS FULL	PRODUCT_INFORMATION	1	12	87
5	BUFFER SORT		105	420	9135
6	INDEX FULL SCAN	ORDER_PK	105	420	105
* 7	INDEX UNIQUE SCAN	ORDER_ITEMS_UK	1	8	269

Statistics feedback

DEMO

On the next run estimates are more accurate

```
-------------------------------------------------------------------------------------
| Id  | Operation                    | Name                | E-Rows |E-Bytes| A-Rows |
-------------------------------------------------------------------------------------
|   0 | SELECT STATEMENT             |                     |        |       |      1 |
|   1 |  SORT AGGREGATE              |                     |      1 |    24 |      1 |
|   2 |   NESTED LOOPS               |                     |    313 |  7512 |    269 |
|*  3 |    HASH JOIN                 |                     |    313 |  6260 |    269 |
|*  4 |     TABLE ACCESS FULL        | PRODUCT_INFORMATION |     87 |  1044 |     87 |
|   5 |     INDEX FAST FULL SCAN     | ORDER_ITEMS_UK      |    665 |  5320 |    665 |
|*  6 |    INDEX UNIQUE SCAN         | ORDER_PK            |      1 |     4 |    269 |
-------------------------------------------------------------------------------------
```

```
 CHILD_NUMBER PLAN_HASH_VALUE EXECUTIONS BUFFER_GETS IS_REOPTIMIZABLE
------------ --------------- ---------- ----------- ------------------
           0      1751333422          1        1289                  Y
           1      1332721014          1          29                  N
```

Where are dynamic stats stored?

Sometimes they are stored in the SGA, other times they are stored as a SQL HINT:

```
SELECT hint_text
  FROM V$SQL_REOPTIMIZATION_HINTS
  WHERE sql_id ='c7db4mrf05vtx';
```

```
OPT_ESTIMATE (@"SEL$3" TABLE "P"@"SEL$3" ROWS=87.000000 )
```

This is a cardinality adjustment for a specific operation in the query

Auxiliary Statistics

Dynamic statistics uses the same hint at SQL profiles generated by tuning advisor

Example hints:

```
OPT_ESTIMATE (@"SEL$1" INDEX_SCAN "SALES"@"SEL$1" "SALESI" SCALE_ROWS=0.001760 )
OPT_ESTIMATE (@"SEL$1" TABLE "SALES"@"SEL$1" SCALE_ROWS=0.000003 )
```

```
OPT_ESTIMATE( [Query Block] operation_type identifier adjustment)
```

Note: this hint is undocumented and used by Oracle internally, you generally should not be using it directly!

Thanks to blog post by Christo Kutrovsky for syntax details.

OPT_EST Hint – some more details

- QUERY BLOCK – optional, names can be found in the outline or provide them in your query
- OPERATION_TYPE – what type of operation are we adjusting
  - TABLE
  - INDEX_FILTER
  - INDEX_SCAN / INDEX_SKIP_SCAN
  - JOIN
- IDENTIFIER – the object you are adjusting
  - TABLES table_alias@query_block
  - INDEXS table_alias@query_block index_name
  - JOINS JOIN(table1_alias@query_block table2_alias@query_block)
- ADJUSTMENT
  - ROWS – number of rows that will be returned
  - SCALE_ROWS – percentage of rows that will be returned of amount estimated
  - MIN – minimum number of rows from step
  - MAX – maximum number of rows from step

SQL Plan Directives

Additional information stored by the Optimizer to generate better plans in the future.

Based on query expressions not SQL statements

Can apply to multiple statements

Viewed in: `DBA_SQL_PLAN_DIRECTIVES & DBA_SQL_PLAN_DIR_OBJECTS`

Stored in SYSAUX tablespace,
maintenance job removes directives not used in 53 weeks.

If a query is flagged to be reoptimized, then set in `v$sql` `IS_REOPTIMIZABLE = Y`

New plan will show this:

```
Note
-----
 - statistics feedback used for this statement
```

DEMO

SQL Plan Directives – dynamic stats

SPD can also be used with dynamic statistics, when turned on:

```
OPTIMIZER_ADAPTIVE_STATISTICS = TRUE
```

Dynamic sampling can also be used when the Optimizer detects that object statistics are not sufficient.

```
Note
-----
 - dynamic statistics used: dynamic sampling (level=AUTO)
    - 1 Sql Plan Directive used for this statement
```

That will also mark the SPD to be used for future statistics gathering to make enhanced statistics:

```
STATE      LAST_USE AUT ENA SPD_TEXT                                                INTERNAL_STATE
---------- -------- --- --- ------------------------------------------------------- ------------
USABLE     20250520 YES YES {EC(SH.CUSTOMERS)[CUST_CITY, CUST_STATE_PROVINCE]} MISSING_STATS
```

SQL Plan Directives

These are stored information at the OBJECT level.

They can be used for more than one SQL statement.

Can also be used by automatic extended statistics

- Available starting in 12.1 (patch 21171382)
- Needs to be enabled

```
EXEC DBMS_STATS.SET_DATABASE_PREFS(
   pname => 'AUTO_STAT_EXTENSION',
   value => 'ON');
```

You can also purge directives you no longer want,

Extended statistics

Custom gathered statistics on a set of columns in one or more tables. Helps with complex joins.

Example relationship:

- SH.CUSTOMERS table
- CUST_STATE_PROVINCE + COUNTRY_ID

- When querying these two columns together, one is always a subset of the other
- EG each CUST_STATE_PROVINCE only exist in a specific COUNTRY_ID

The Optimizer will not know this and may make incorrect cardinality estimates.

Extended statistics map that relationship.

What to do?

Use a plan stability tool

Oracle provides multiple tools for “plan stability”

Each tool has their pluses and minuses

Tool	Method	Pluses	Minuses
SQL Patch	Embedded hint	Easy No License	Hint (can be ignored) Only one per statement
SQL Profile	Embedded hint (one of) • Execution plan • Cardinality adjustment	Tuning Pack License	Hint (can be ignored) Only one per statement
SQL Baseline	Saved execution plan	No License Multiple per statement (EE) Evolve	None

These tools are “temporary fixes”
the SQL will probably change again

Manual capture example – Cursor Cache

Capturing a single plan from cursor cache

- Specify a SQL_ID and a PLAN_HASH_VALUE (PHV)
- If you exclude PHV, then all plans for the given SQL are loaded

```
VARIABLE v_plan_cnt NUMBER

EXECUTE :v_plan_cnt :=
   DBMS_SPM.LOAD_PLANS_FROM_CURSOR_CACHE(
     sql_id => '27m0sdw9snw59',
     plan_hash_value => 3534348942);
```

Returns the number of plans loaded

SPM “One-Shot” – quick fix

One call to fix performance issue on a single statement:

- Looks for SQL Plans (ASTS, AWR, SPM, cursor cache)
- Creates baselines and tests them.
- Automatically accepts good performing baselines.

Included in 23ai

Back ported to 19c – patch 34534504 (19.11 – 19.21)

Included in RU 19.22 (Jan 2024)

```
exec :report := dbms_spm.add_verified_sql_plan_baseline('9y5vtn2knayhs');

select :report report from dual;
```

Extended Statistics

Oracle 12.1 and above has automatic detection.

```
set long 100000
set lines 120
set pages 50

select dbms_stats.report_col_usage('OE', 'ORDERS') from   dual;
```

Then create new extended statistics:

```
select dbms_stats.create_extended_stats('OE', 'ORDERS') from dual;
exec dbms_stats.gather_table_stats('OE', 'ORDERS');
```

Manually create extended statistics on specific column groupings using:

```
DBMS_STATS.create_extended_stats
```

Also, you can enable automatic creation of extended statistics:

```
EXEC DBMS_STATS.SET_GLOBAL_PREFS('AUTO_STAT_EXTENSIONS','ON')
```

Tracing Adaptive features

Some adaptive features take place at execution time not parsing

- SQL Trace (10046) – captures execution of SQL
  - Useful to see what happened, but a SQL monitor or gather plan statistics will give you basically the same information in easier to read format.
- CBO Trace (10053) – capture parse of SQL
  - Must be done during a hard parse of a statement
  - Provides all the details and calculations performed by the Optimizer
  - Does not show Adaptive Cursor Sharing information (or other execution time decisions)
- Bind Equivalence Logic (10507) – captures bind value changes
  - Must be done during additional runs of statement, shows when the Optimizer detected the change

Huge Thank You to fellow ACE Monika Lewandowska for more ACS details! https://ace.oracle.com/ords/ace/profile/mlewan1115

Trace example

```
ALTER SESSION SET EVENTS '10507 trace name context forever, level 4294967295’;
<YOUR SQL>
ALTER SESSION SET EVENTS '10507 trace name context off';
```

```
Adaptive Sharing Context [sql_id=g89xvwavgqpha]
  Peeked Binds:
    Bind Set:
      Hash: 3640740885
  Peeked Bind Bucket: none
...
KKSMEC: Invalidating old cursor 0 with hash val = 2366003765
KKSMEC: Produced New cursor 1 with hash val = 1723173041
BINDS #139986510955536:
```

Huge thanks to ACE Alumni Franck Pachot for post on tracing!

Questions / Comments

Dynamic Statistics (Sampling)

The Optimizer normally uses static object statistics

By default, Dynamic Statistics is enabled, you can adjust the setting

```
OPTIMIZER_DYNAMIC_SAMPLING = [ 0 - 11]
```

0	Disabled
1	Table with no stats & (no indexes or more than 32 blocks)
2	Any table without statistics
3	One or more expressions in the WHERE clause predicates
4	SQL has multiple predicates on the same table
5 -11	Same as 4 with different sample size

Sample Size

2,3,4	64 blocks
5	128 blocks
6	256 blocks
7	512 blocks
8	1024 blocks
9	4086 blocks
10	All blocks
11	Automatic

References

- SQL Plan Management – SQL Tuning Guide: https://docs.oracle.com/en/database/oracle/oracle-database/23/tgsql/overview-of-sql-plan-management.html#GUID-BC65F4D4-1CEE-4014-8188-C849A0D9251A
- Monitoring SQL Statements with Real-Time SQL Monitoring (KB135307 / Doc ID 1380492.1)
- Adaptive Query Optimization (KB116528 / Doc ID 2031605.1)
- How to Use SQL Plan Management (SPM) - Plan Stability Worked Example (KB129222 / Doc ID 456518.1)
- SQL Plan Baseline Not always created (KB140278 / Doc ID 788853.1)
- How To Use DBMS_XPLAN.COMPARE_PLANS & Hint Usage Report To Troubleshoot Plan Reproducibility Issues (SPM Baselines & SQL Profiles) From 19c (KB118771 / Doc ID 2736319.1)
- FAQ: SQL Plan Management (SPM) Frequently Asked Questions (FAQ2280 / Doc ID 1524658.1)
- Loading Hinted Execution Plans into SQL Plan Baseline. (KB83477 / Doc ID 787692.1)
- Master Note: Plan Stability Features (Including SQL Plan Management (SPM)) (KB149103 / Doc ID 1359841.1)
- How to Drop Plans from the SQL Plan Management (SPM) Repository (KB129427 / Doc ID 790039.1)
- How to Load SQL Plans into SQL Plan Management (SPM) from the Automatic Workload Repository (AWR) (KB82689 / Doc ID 789888.1)

References – Cont.

- SRDC - How to Collect Standard Information for an Issue Where a SQL Plan Management (SPM) Baseline is Not Used (FAQ2280 / Doc ID 1644732.1)
- Using the Automatic SQL Tuning Set in Oracle Database 19c – Nigel Bayliss https://blogs.oracle.com/optimizer/post/using-asts-in-19c
- Blog “Should You Gather System Statistics?” by Nigel Bayliss – Oracle product manager Optimizer https://blogs.oracle.com/optimizer/post/should-you-gather-system-statistics
- Bitmap Index vs. B-tree Index: Which and When? https://www.oracle.com/technical-resources/articles/sharma-indexes.html
- Query Optimizer: What is Density? (KB79135 / Doc ID 43041.1)
- Best Practices for Gathering Optimizer Statistics with Oracle Database 19c https://www.oracle.com/docs/tech/database/technical-brief-bp-for-stats-gather-19c.pdf
- Optimizer Cost Based Query Transformation (KB143282 / Doc ID 1082127.1)
- Are Extended Statistics Collected Automatically on Oracle 12c? (KB127675 / Doc ID 1964223.1)
- How to Identify Specific SQL Plan Directives In Use For A SQL Statement (KB130950 / Doc ID 2390551.1)

References

- Diagnostics for Query Tuning Problems (Doc ID 68735.1)
- SQL Tuning Health-Check Script (SQLHC) (Doc ID 1366133.1)
- How to Obtain Tracing of Optimizer Computations (EVENT 10053) (Doc ID 225598.1)
- Script to Bulk Drop SQL Plan Directives (Doc ID 2323144.1)



VISCOSITY

PROFESSIONAL SERVICES