“Tune that SQL”, now what do I do?

Gary Gordhamer

New York Oracle Users Group

September 2021

Gary Gordhamer

Managing Principal Consultant
Viscosity North America

30 years of IT experience
29 years with Oracle (6.x up to 19c)
Worked in many different industries including healthcare, manufacturing, utilities, banking, and marketing

@ggordham
linkedin.com/in/ggordhamer/
oraontap.blogspot.com

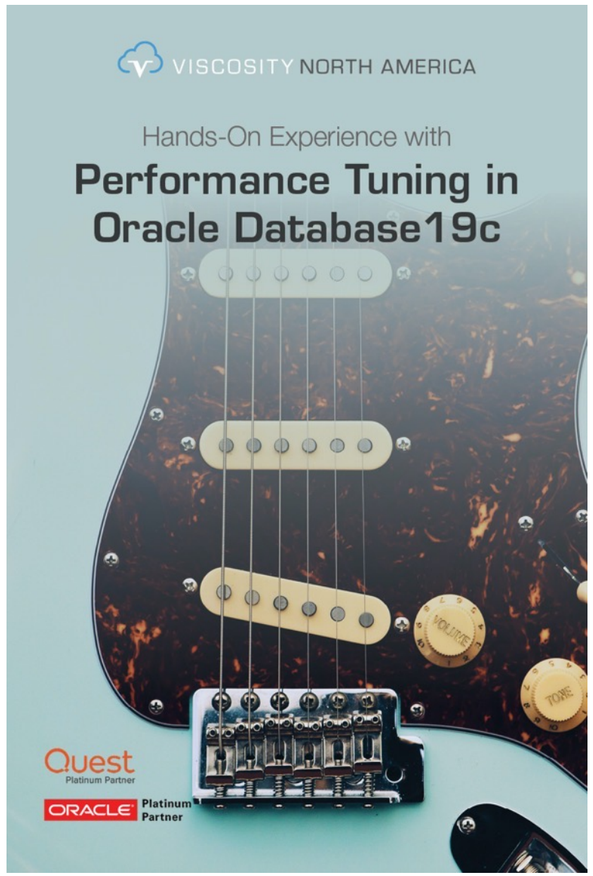
Quest
IOUG Database & Technology Community

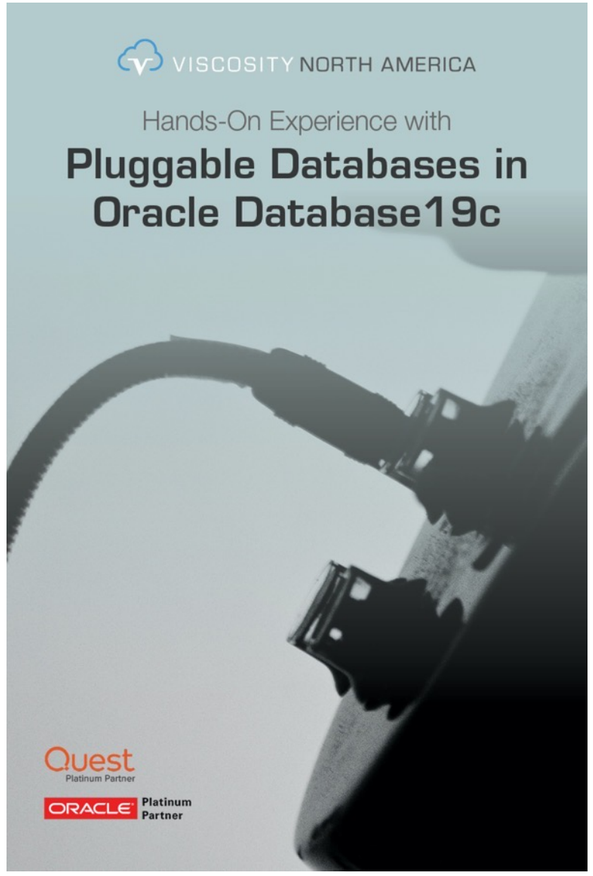
ORACLE
ACE Associate

Note: The views in this presentation are my own and do represent the views of the company I work for.

Oracle 19c Hands-On Experience

Publications coming soon





Want to try this out / follow along?

You can download all the scripts from GitHub
https://github.com/ggordham/ora-presentations/tree/main/tune-that-SQL-NYOUG

Or: https://bit.ly/3tzdbez `bit.ly / 3tzdbez`

All information 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.

So, you have a bad SQL statement?

Options to change the performance of a bad SQL:

- Look at environment setup (like statistics / missing index)
- Change the SQL / add a HINT
- Apply a SQL PROFILE
- Use a SQL PLAN BASELINE
- Create and apply a SQL PATCH

Step 1 identify how a SQL statement is performing

Step 2 what is a better performance option for that SQL statement

Step 3 apply a fix from the list above

Agenda

- Finding the SQL Statement
- The Oracle Optimizer
- Details in an Explain Plan
- What to Tune?
- Tools to help

Finding the SQL Statements

Where to Look

- Cursor Cache
  - What is in memory (G)V$SQL view
- AWR Report
  - Captured data from cursor cache
  - Only contains top N statements (you can adjust this)
- ASH
  - Historical data of SQL statements saved to SYSAUX tablesapce

Cursor Cache – (G)V$SQL

BASIC SQL INFORMATION

- SQL_ID – unique ID based on SQL text (will change if the text changes)
- SQL_TEXT – first 1,000 characters of SQL, consider looking at SQL_FULLTEXT (CLOB)
- PLAN_HASH_VALUE – representation of Execution Plan
- EXECUTIONS – number of executions since loaded into cache
- OPTIMIZER_COST – Execution Plan cost for cached Plan

SQL	PARSING_SCHEMA	SQL_ID	PLAN_HASH_VALUE	EXECS	LAST_ACTIVE_TIME_FORM.	OPTIMIZER_COST
select count(*) from t2 w	PERFLAB	19nmwtxrh5rf1	3321871023	115	13-Sep-2021 16:57:54	590
SELECT /*+ gather_plan_st	PERFLAB	fua0hb5hfst77	906334482	144	13-Sep-2021 16:57:54	1892
select count(t1.a) from t	PERFLAB	38b5rzb4bfmdw	4224043856	39	13-Sep-2021 16:57:07	555
select sum(t1.a + t2.c) f	PERFLAB	979hgkk210sqw	4205221947	50	13-Sep-2021 16:57:20	365
select sum(t1.a + t2.c) f	PERFLAB	979hgkk210sqw	1965891112	1	13-Sep-2021 16:56:26	1537

Cursor Cache – (G)V$SQL

V$SQL statistics are updated every 5 seconds for in-process SQL

- DISK_READS – data blocks reads from disk
- DIRECT_READS – reads that "bypass" the buffer cache (put on end of cache)
- BUFFER_GETS – reads from buffer cache (not from disk)
- PHYSICAL_READ_REQUESTS / PHYSICAL_READ_BYTES – any request from disk
- CPU_TIME – CPU used in microseconds
- ELAPSED_TIME – in microseconds – wall clock time used
- CLUSTER_WAIT_TIME – in microseconds

And Many More

These metrics are cumulative, for all executions in the cache

Divide by the number of executions to get per run data

What is Your Pain Point

- Users waiting a long time – Elapsed Time
- CPU on DB server is too busy – CPU Time / Buffer Gets
- Disk subsystem is too busy – Disk Reads / Direct Reads
- Cluster is “busy” – Cluster Wait Time (Interconnect Wait)

Narrow down what you are trying to tune, then identify the SQL’s that are using the most of that resource.

AWR – A Workload Repository

Stored in DBA_HIST* tables

- DBA_HIST_SNAPSHOT – begin and end times for each capture
- DBA_HIST_SQLSTAT – Top N statements captured during a specific snapshot
- DBA_HIST_SQLPLAN – the plans for the captured statements
- DBA_HIST_SQLTEXT – SQL text for captured statements

Stats data is total since cursor was in memory, delta is amount of resource spent from begin to end of snap time

AWR – Automatic Workload Repository

You can adjust the amount of AWR data that is stored.

You can also set the number of TOP N SQL to capture

- **Retention** – number of minutes to keep (57600 = 40 days)
- **Interval** – how frequent to take snaps (30 = 30 minutes)
- **Topnsql** – How many SQL to capture (quantity)

```
begin
dbms_workload_repository.modify_snapshot_settings(
    retention => 57600,
    interval => 30,
    topnsql => 30);
end;
/
```

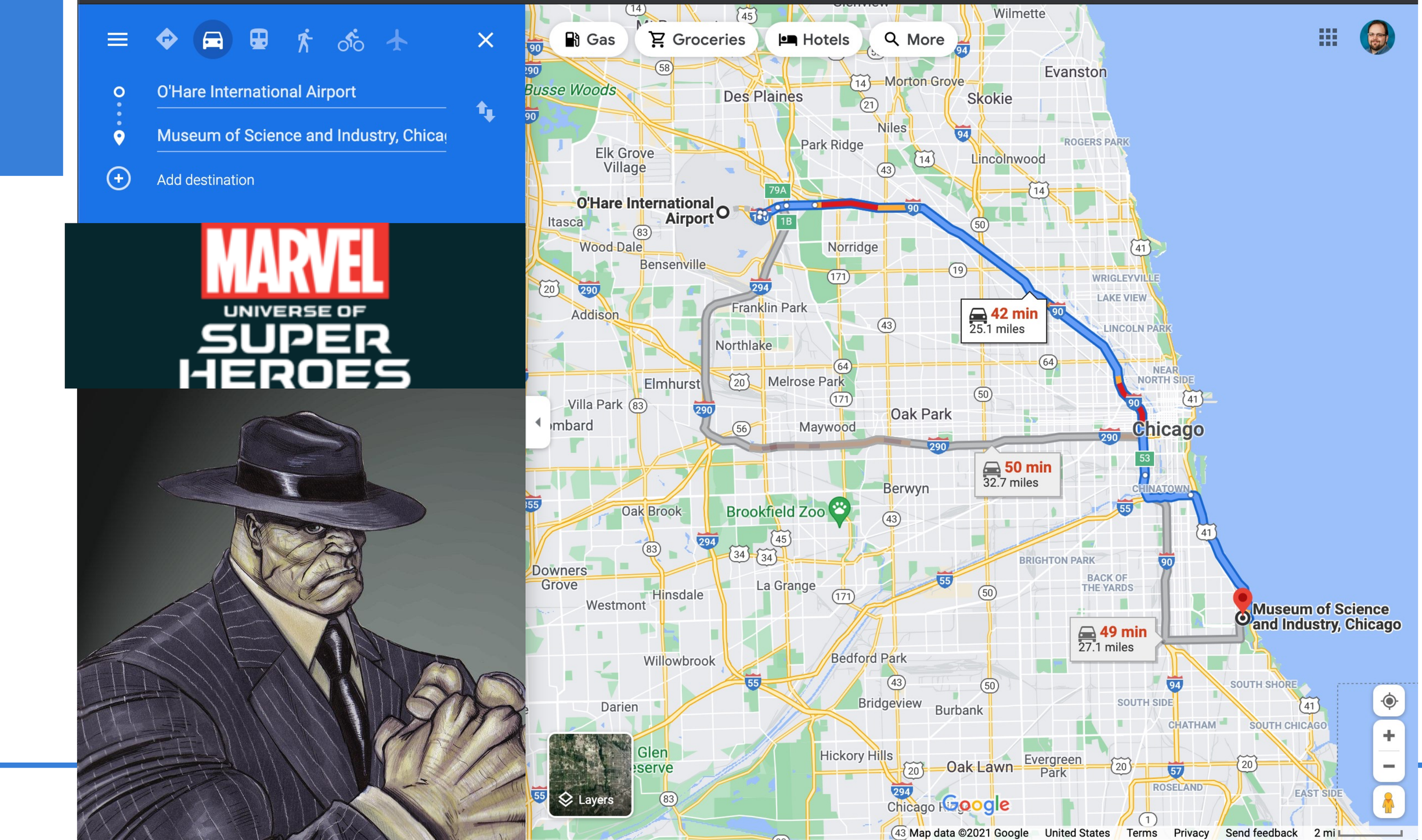
ASH – Active Session History

Stored in DBA_HIST_ACTIVE_SESS_HISTORY

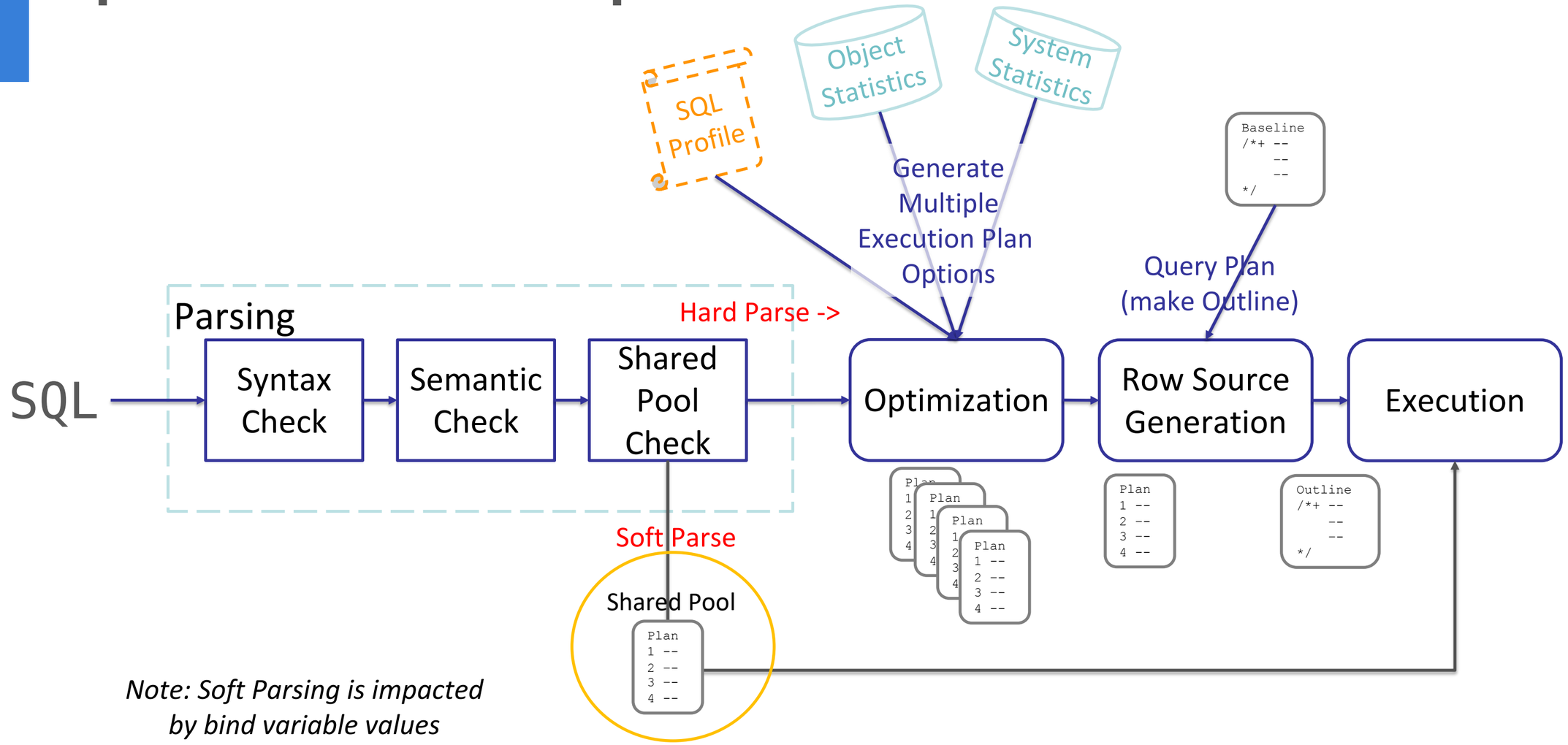
- Collects information on SQL that
  - >1 second of execution
  - Not Idle

- Maps to DBA_HIST_ASH_SNAPSHOT for capture timings

The Oracle Optimizer


O'Hare International Airport
Museum of Science and Industry, Chica
Add destination
Gas
Groceries
Hotels
More
MARVEL
UNIVERSE OF
SUPER
HEROES
42 min
25.1 miles
50 min
32.7 miles
49 min
27.1 miles
O'Hare International Airport
Museum of Science and Industry, Chicago
Chicago
Oak Park
Evanston
Skokie
Des Plaines
Brookfield Zoo
Layers
Google
Map data ©2021 Google
United States
Terms
Privacy
Send feedback
2 mi

Optimizer SQL Steps



Note: Soft Parsing is impacted by bind variable values

Example Tables

Tables t1 and t2

Each has 500,000 rows of the same data

t1			
A	B	C	D
NUMBER	VARCHAR2(1000)	NUMBER	NUMBER

t2			
A	B	C	D
NUMBER	VARCHAR2(1000)	NUMBER	NUMBER

```
OWNER    SEGMENT_NAME  SEGMENT_TYPE TABLESPACE_NAME NO_EXTENTS    SEG_SIZE
-------- ------------- ------------ --------------- ---------- -----------
PERFLAB  T1            TABLE        USERS                  31  16,777,216

ROWNUM OWNER    SEGMENT_NAME       BYTES     BLOCKS
------ -------- ------------- ---------- ----------
     1 PERFLAB  T1                 65536          8
    16 PERFLAB  T1                 65536          8
    17 PERFLAB  T1               1048576        128
    31 PERFLAB  T1               1048576        128
```

```
SQL> select * from t1 where rownum < 5;

         A B                    C          D
---------- ---------- ---------- ----------
         1 EXWVLHKQDF          1          1
   1000001 EXWVLHKQDF          1         10
         2 EXWVLHKQDF          1          2
   1000002 EXWVLHKQDF          1         10
```

First Example – No Stats

```
SELECT sum(t1.c), sum(t2.c)
  FROM t1, t2
  WHERE t1.a = t2.a AND t1.d = :idnum
```

```
-------------------------------------+-----------------------------------+
| Id  | Operation            | Name  | Rows  | Bytes | Cost  | Time      |
-------------------------------------+-----------------------------------+
| 0   | SELECT STATEMENT     |       |       |       |  1096 |           |
| 1   |  SORT AGGREGATE      |       |     1 |    65 |       |           |
| 2   |   HASH JOIN          |       |  1643 |  104K |  1096 |  00:00:14 |
| 3   |    TABLE ACCESS FULL | T1    |  1643 |   63K |   548 |  00:00:07 |
| 4   |    TABLE ACCESS FULL | T2    |  160K | 4171K |   547 |  00:00:07 |
-------------------------------------+-----------------------------------+
```

Table has 500K rows, why 160K estimate?

- Segment size 16MB (from pervious slide) / 100 row length

Table: T2 Alias: T2 (NOT ANALYZED)

#Rows: 164259 SSZ: 0 LGR: 0 #Blks: 2011 AvgRowLen: 100.00

Second Example –Stats

```
SELECT sum(t1.c), sum(t2.c)
  FROM t1, t2
  WHERE t1.a = t2.a AND t1.d = :idnum
```

```
-----------------------------------------------+-----------------------------------+
| Id  | Operation                      | Name    | Rows  | Bytes | Cost  | Time      |
-----------------------------------------------+-----------------------------------+
| 0   | SELECT STATEMENT               |         |       |       |   556 |           |
| 1   |  SORT AGGREGATE                |         |     1 |    22 |       |           |
| 2   |   NESTED LOOPS                 |         |     2 |    44 |   556 |  00:00:07 |
| 3   |    NESTED LOOPS                |         |     2 |    44 |   556 |  00:00:07 |
| 4   |     TABLE ACCESS FULL          | T1      |     2 |    26 |   550 |  00:00:07 |
| 5   |     INDEX RANGE SCAN           | T2I     |     1 |       |     2 |  00:00:01 |
| 6   |    TABLE ACCESS BY INDEX ROWID | T2      |     1 |     9 |     3 |  00:00:01 |
-----------------------------------------------+-----------------------------------+
```

Table: T2 Alias: T2: #Rows: 500000 SSZ: 0 LGR: 0 #Blks: 2011 AvgRowLen: 23.00

Column (#4): D(NUMBER)

AvgLen: 4 NDV: 253072 Nulls: 0 Density: 0.000004 Min: 1.000000 Max: 250000.000000

Estimated selectivity: 0.0000039514, col: #4

Card: Original: 500000.000000 Rounded: 2 Computed: 1.975722 Non Adjusted: 1.975722

Estimates 2 rows or < 1%

Second Example –Stats + Histograms

```
SELECT sum(t1.c), sum(t2.c)
  FROM t1, t2
  WHERE t1.a = t2.a AND t1.d = :idnum
```

```
-------------------------------------+-----------------------------------+
| Id  | Operation            | Name  | Rows  | Bytes | Cost  | Time      |
-------------------------------------+-----------------------------------+
| 0   | SELECT STATEMENT     |       |       |       |  1893 |           |
| 1   |  SORT AGGREGATE      |       |     1 |    22 |       |           |
| 2   |   HASH JOIN          |       |  241K | 5305K |  1893 |  00:00:23 |
| 3   |    TABLE ACCESS FULL | T1    |  241K | 3135K |   550 |  00:00:07 |
| 4   |    TABLE ACCESS FULL | T2    |  488K | 4395K |   549 |  00:00:07 |
-------------------------------------+-----------------------------------+
```

Column (#4): D(NUMBER)

AvgLen: 4 NDV: 253072 Nulls: 0 Density: 0.000002 Min: 1.000000 Max: 250000.000000

Histogram: Hybrid #Bkts: 254 UncompBkts: 5433 EndPtVals: 254 ActualVal: yes

Estimated selectivity: 0.493858 , endpoint value predicate, col: #4

Card: Original: 500000.000000 Computed: 246929.000000 Non Adjusted: 246929.000000

We went from:
2 rows
24,6929 rows
Or 49.3%

What have I not been telling you?

Tables t1 and t2

- Each has 500,000 rows
- Column D
  - 250,001 rows value = 10
  - 249,999 rows "random" unique value

t1			
A	B	C	D
NUMBER	VARCHAR2(1000)	NUMBER	NUMBER

t2			
A	B	C	D
NUMBER	VARCHAR2(1000)	NUMBER	NUMBER

So, if I search for `d=10`

  - Oracle must find ½ the table

If I search for `d=1000`

  - Oracle must find 1 record

Joining 2 tables using "most" of the data – Hash Join

Joining 2 tables looking for just 1 record in each table – Nested Loops

Driving directions

O'Hare International Airport 10000 O'...

Museum of Science and Industry 57...

Arrive at

Fri, Sep 17

6:00 PM

Save to app

Routes

1 **1h 9m** Leave by 4:51 PM OPTIMAL

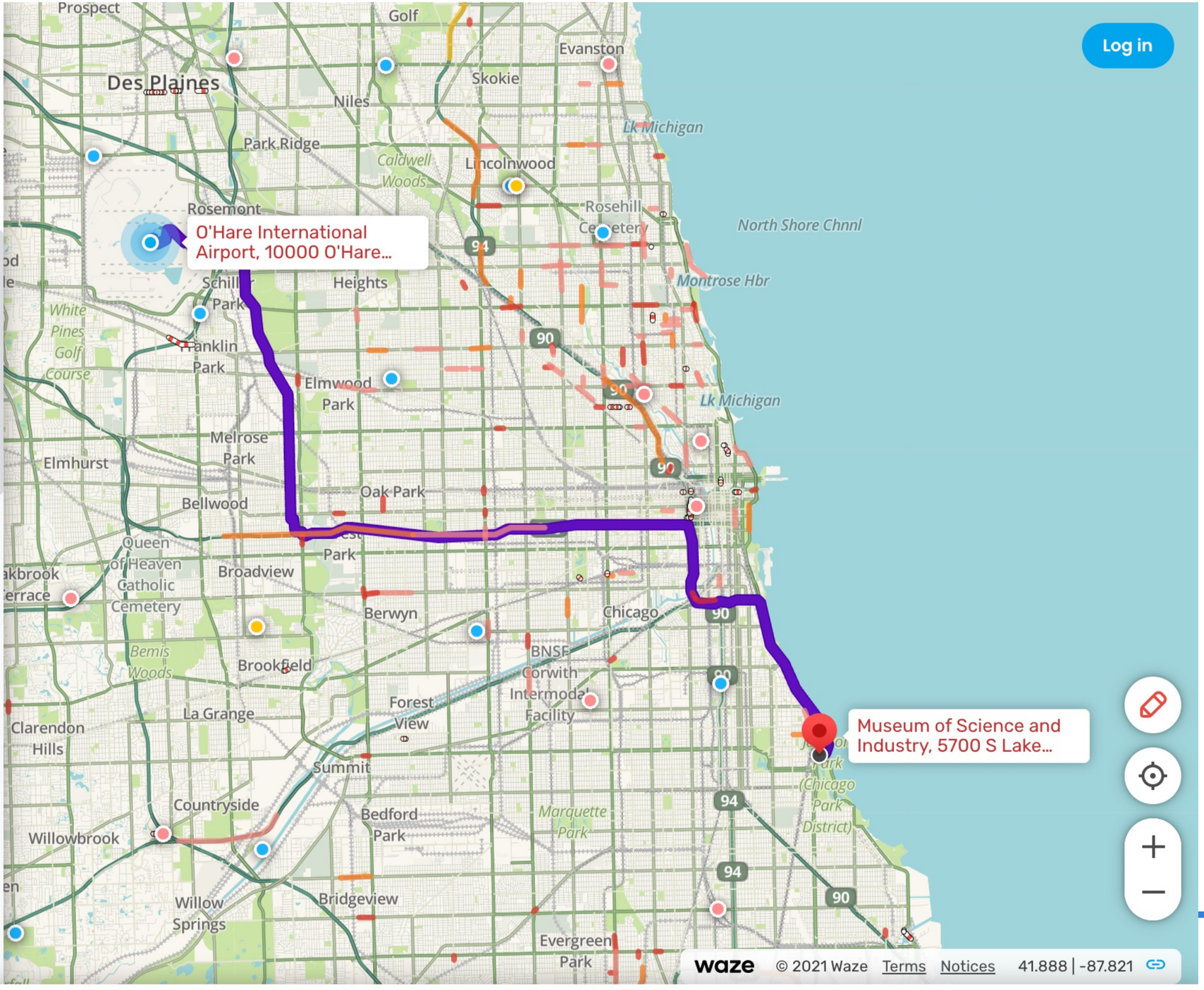
I-290 E Chicago

28.9 MILES

Starting point | Destination

Museum of Science and Industry

5700 S Lake Shore Dr, Chicago, Illinois, United States



Details in the Explain Plan

What is an Explain Plan?

These are the steps the SQL engine in Oracle will take to retrieve the data

What methods are generally slow:

- Full table scan ❌
- Nested Loops ❌ - this also can depend on quantity of data
- Index full scan ❌

What methods are generally faster:

- Index range scan ✓
- Index unique scan ✓
- Index skip scan ✓
- Table access by index rowid ✓

Performance depends:

- Hash Join
- Index Fast Full Scan

Parts of an explain plan

- SQL_ID - Unique ID for the SQL statement (should be consistent across instances unless text of SQL changes)
- Plan Hash Value – Unique for each plan generated
- Plan - details of steps taken to execute the SQL
- Predicate Information – how is the data being filtered / accessed
- Column Projection – what columns are involved in the execution steps (are in the work area)
- Outline Data – could be used to generate an Outline also a somewhat internal view of the execution plan

Getting an Explain Plan

Run a query or do

```
EXPLAIN PLAN FOR <query>
```

Get explain plan:

```
SELECT *
FROM
      table(DBMS_XPLAN.DISPLAY_CURSOR(FORMAT=>'ALL'));
```

Note: we cast the output of the function to be of type “TABLE”

Only the “last” query you explain or run will be there

Basic plan

```
SQL_ID  08htbt4xvf76d
```

SQL ID:

Unique ID based on SQL text

Should match across databases

If you hint or change the SQL text, the ID will change!

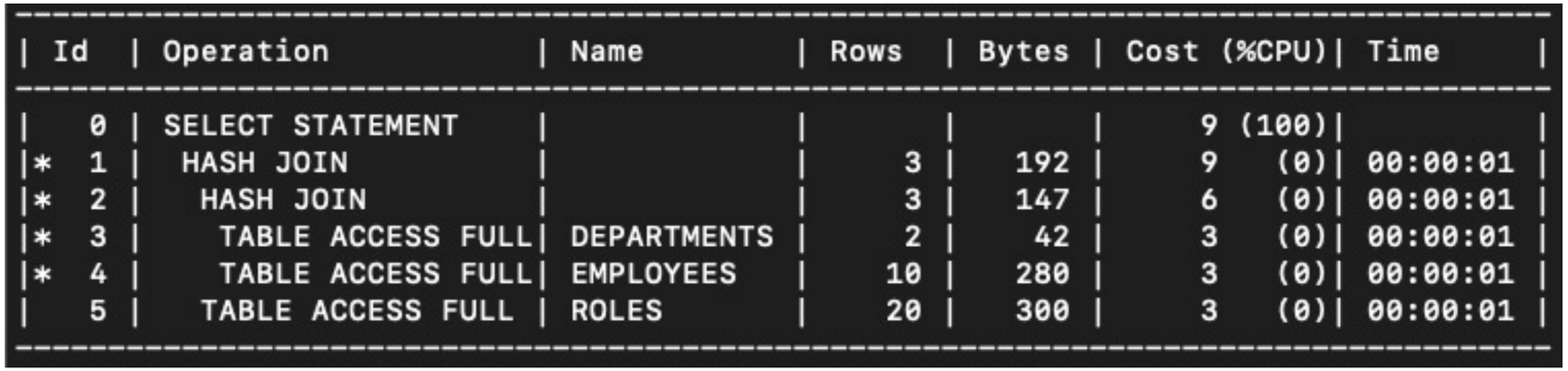
```
-----------------------------------------
| Id  | Operation           | Name      |
-----------------------------------------
|   0 | SELECT STATEMENT    |           |
|   1 |  HASH JOIN          |           |
|   2 |   HASH JOIN         |           |
|   3 |    TABLE ACCESS FULL| DEPARTMENTS |
|   4 |    TABLE ACCESS FULL| EMPLOYEES |
|   5 |   TABLE ACCESS FULL | ROLES     |
-----------------------------------------
```

Use:

```
format=> 'BASIC'
```

(or no format option passed)

All plan (plans/eg1)



```
-------------------------------------------------------------------------------
| Id  | Operation           | Name        | Rows  | Bytes | Cost (%CPU)| Time     |
-------------------------------------------------------------------------------
|   0 | SELECT STATEMENT    |             |       |       |     9 (100)|          |
|*  1 |  HASH JOIN          |             |     3 |   192 |     9   (0)| 00:00:01 |
|*  2 |   HASH JOIN         |             |     3 |   147 |     6   (0)| 00:00:01 |
|*  3 |    TABLE ACCESS FULL| DEPARTMENTS |     2 |    42 |     3   (0)| 00:00:01 |
|*  4 |    TABLE ACCESS FULL| EMPLOYEES   |    10 |   280 |     3   (0)| 00:00:01 |
|   5 |   TABLE ACCESS FULL | ROLES       |    20 |   300 |     3   (0)| 00:00:01 |
-------------------------------------------------------------------------------
```

Use:

```
format=> 'ALL'
```

Note: these are numbers based on statistics on the objects or system wide statistics

Plan with statistics (plans/eg2)

Id	Operation	Name	Starts	E-Rows	E-Bytes	Cost (%CPU)	E-Time	A-Rows	A-Time	Buffers	OMem	1Mem	Used-Mem
0	SELECT STATEMENT		1			9 (100)		1	00:00:00.01	21			
* 1	HASH JOIN		1	3	192	9 (0)	00:00:01	1	00:00:00.01	21	1449K	1449K	740K (0)
* 2	HASH JOIN		1	3	147	6 (0)	00:00:01	2	00:00:00.01	13	2546K	2546K	902K (0)
* 3	TABLE ACCESS FULL	DEPARTMENTS	1	2	42	3 (0)	00:00:01	2	00:00:00.01	6			
* 4	TABLE ACCESS FULL	EMPLOYEES	1	10	280	3 (0)	00:00:01	10	00:00:00.01	6			
5	TABLE ACCESS FULL	ROLES	1	20	300	3 (0)	00:00:01	20	00:00:00.01	7			

Add hint to SQL:

```
/*+ gather_plan_statistics */
```

Use:

```
format=>'ALLSTATS LAST +cost +bytes'
format=>'ALLSTATS LAST ALL'
```

A- is Actuals

E- is Estimates

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

Execution Plans Are Not Absolutes!

An execution plan can have different costs

A SQL may run better with different execution plans at different times

Plan stability tools can help and can hurt performance

- SQL Profiles are statistical information on generating plans
- SQL Baselines allow for multiple plans
- SQL Patches are the "hammer" of the toolset

Your system will change over time

- Data growth / change
- Hardware changes
- Software changes

Your tuning methodology needs to adapt to change!

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

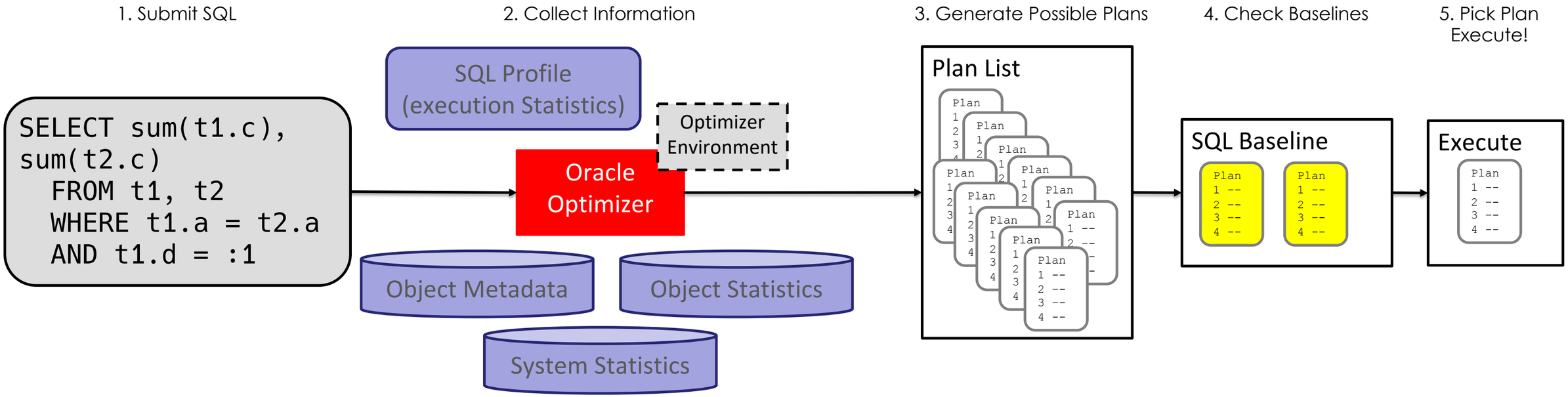
Id	Operation	Name	Starts	E-Rows	E-Bytes	Cost (%CPU)	A-Rows	A-Time	Buffers
0	SELECT STATEMENT		1			741K(100)	1	00:00:00.43	11089
1	SORT AGGREGATE		1	1	22		1	00:00:00.43	11089
2	NESTED LOOPS		1	246K	5305K	741K (1)	250K	00:00:00.44	11089
3	NESTED LOOPS		1	246K	5305K	741K (1)	250K	00:00:00.30	9118
* 4	TABLE 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

What to tune?

SQL Tuning

SQL Statement – what data you want Oracle to retrieve

Execution Plan – method Oracle will use to retrieve the data



SQL Tuning – the art of affecting one or more parts of this process to decrees execution time or resource consumption

SQL Tuning – Basic Steps

1. Document the issue (capture information)
   a) SQL Statement ID
   b) Current Execution plan
   c) Current performance / Resource usage
2. Identify the goal (target performance)
3. Try the standard tools
   a) Explain plan – look for bad decisions (ignoring indexes, etc)
   b) Statistics – look for missing or stale statistics
   c) SQL Profile – provide the optimizer more information
   d) SQL Baseline – force a specific plan that is already available
4. Deep Dive

Note: take lots of notes, the deeper you go the more lost you will get!

TKPROF Output – Sample Execution 2

Explain plan vs:

```
Plan hash value: 3534348942
-------------------------------------------------------------------------------------
| Id  | Operation                     | Name | Rows  | Bytes | Cost (%CPU)| Time     |
-------------------------------------------------------------------------------------
|   0 | SELECT STATEMENT              |      |       |       |   741K(100)|          |
|   1 |  SORT AGGREGATE               |      |     1 |    22 |            |          |
|   2 |   NESTED LOOPS                |      |   246K|  5305K|   741K  (1)| 00:00:29 |
|   3 |    NESTED LOOPS               |      |   246K|  5305K|   741K  (1)| 00:00:29 |
|*  4 |     TABLE ACCESS FULL         | T1   |   246K|  3134K|   550   (1)| 00:00:01 |
|*  5 |     INDEX RANGE SCAN          | T2I  |     1 |       |     2   (0)| 00:00:01 |
|   6 |    TABLE ACCESS BY INDEX ROWID| T2   |     1 |     9 |     3   (0)| 00:00:01 |
-------------------------------------------------------------------------------------
```

Trace

```
Rows (1st) Rows (avg) Rows (max)  Row Source Operation
---------- ---------- ----------  ---------------------------------------------------
         1          1          1  SORT AGGREGATE (cr=11089 pr=0 pw=0 time=426057 us starts=1)
    250001     250001     250001   NESTED LOOPS  (cr=11089 pr=0 pw=0 time=444330 us starts=1 cost=741506 size=5432438 card=246929)
    250001     250001     250001    NESTED LOOPS  (cr=9118 pr=0 pw=0 time=290903 us starts=1 cost=741506 size=5432438 card=246929)
    250001     250001     250001     TABLE ACCESS FULL T1 (cr=1976 pr=0 pw=0 time=36524 us starts=1 cost=550 size=3210077 card=246929)
    250001     250001     250001     INDEX RANGE SCAN T2I (cr=7142 pr=0 pw=0 time=185318 us starts=250001 cost=2 size=0 card=1)(object id 139465)
    250001     250001     250001    TABLE ACCESS BY INDEX ROWID T2 (cr=1971 pr=0 pw=0 time=93985 us starts=250001 cost=3 size=9 card=1)
```

Tools to help

SQL Tuning – Tools

	Runs Statement	SQL Execution Steps	Object Information	Database Environment	HTML Reports	Easy to read	Requires Install
Explain Plan	Maybe	Yes	No	No	Maybe	Yes	No
SQL Monitor	Maybe	Yes	No	No	Yes	Yes	No
SQL Health Check (HC)	No	Yes	Extensivie	Yes	Yes	Yes	No
SQL Trace	Yes	Yes	Little	Maybe	No	No	No
CBO Trace	Yes	Yes	Extensivie	Some	No	No	No
SQLT	Maybe	Yes	Extensivie	Extensivie	Yes	Yes	Yes
ADDM*	No	Yes	Some	Some	Yes	Yes	No*
SQL Tuning Advisor*	Yes	Yes	Some	Some	Yes	Yes	No*

Number of tools to choose from, using the right tool for the situation

* *Additional License Required*

SQL Tuning – These Tools Show Me What?

Explain Plan – What steps will be taken to execute the SQL

SQL Health Check – What is the status / details of all the objects touched by the SQL

SQL Trace – What happened when the SQL was executed

CBO Trace – How did the Optimizer decide to execute the SQL

SQLT – Everything + some "Observations"

What are SQL Profiles?

A SQL profile is a database object that contains auxiliary statistics specific to a SQL statement. Conceptually, a SQL profile is to a SQL statement what a set of object-level statistics is to a table or index. A SQL profile can correct suboptimal optimizer estimates discovered during SQL tuning.

SQL profiles fix incorrect estimates while giving the optimizer the flexibility to pick the best plan in different situations.

A SQL profile contains, among other statistics, a set of cardinality adjustments. A profile uses parts of the query to determine whether the estimated cardinalities are close to the actual cardinalities and, if a mismatch exists, uses the corrected cardinalities.

SQL Plan Management / SPM

Plan Management is a new feature in Oracle 11g and up

The optimizer stores and uses multiple different execution plans for the same SQL statement

- Each plan is called a SQL Plan Baseline or Baseline
- Once enabled, Oracle will collect new Baselines for statements
- Automated jobs will “evolve” or accept new baselines nightly
- You can also manually accept, delete, and FIX baselines

SQL Plan Management

Plan Capture

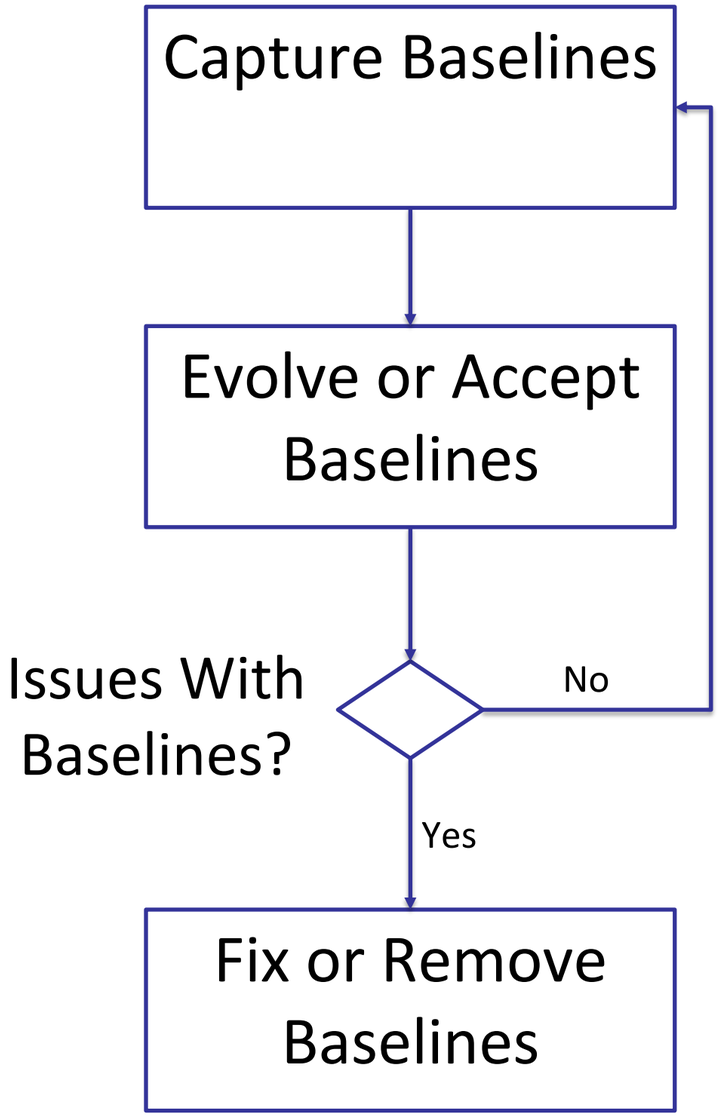
- Automatic
  `optimizer_capture_sql_plan_baselines`
- Manually
  `DBMS_SPM.LOAD_PLANS_FROM_CURSOR_CACHE()`

Plan Selection

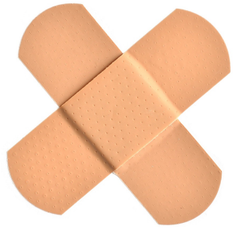
- The Optimizer uses Accepted Baselines

Plan Evolution

- Accept or Reject new Baselines



What is a SQL Patch?



A patch forces the optimizer to use a different execution plan

You can manually create a patch

A patch can be moved from one database to another

A patch forces a specific hint or hints into an execution plan

Patches are created using the function

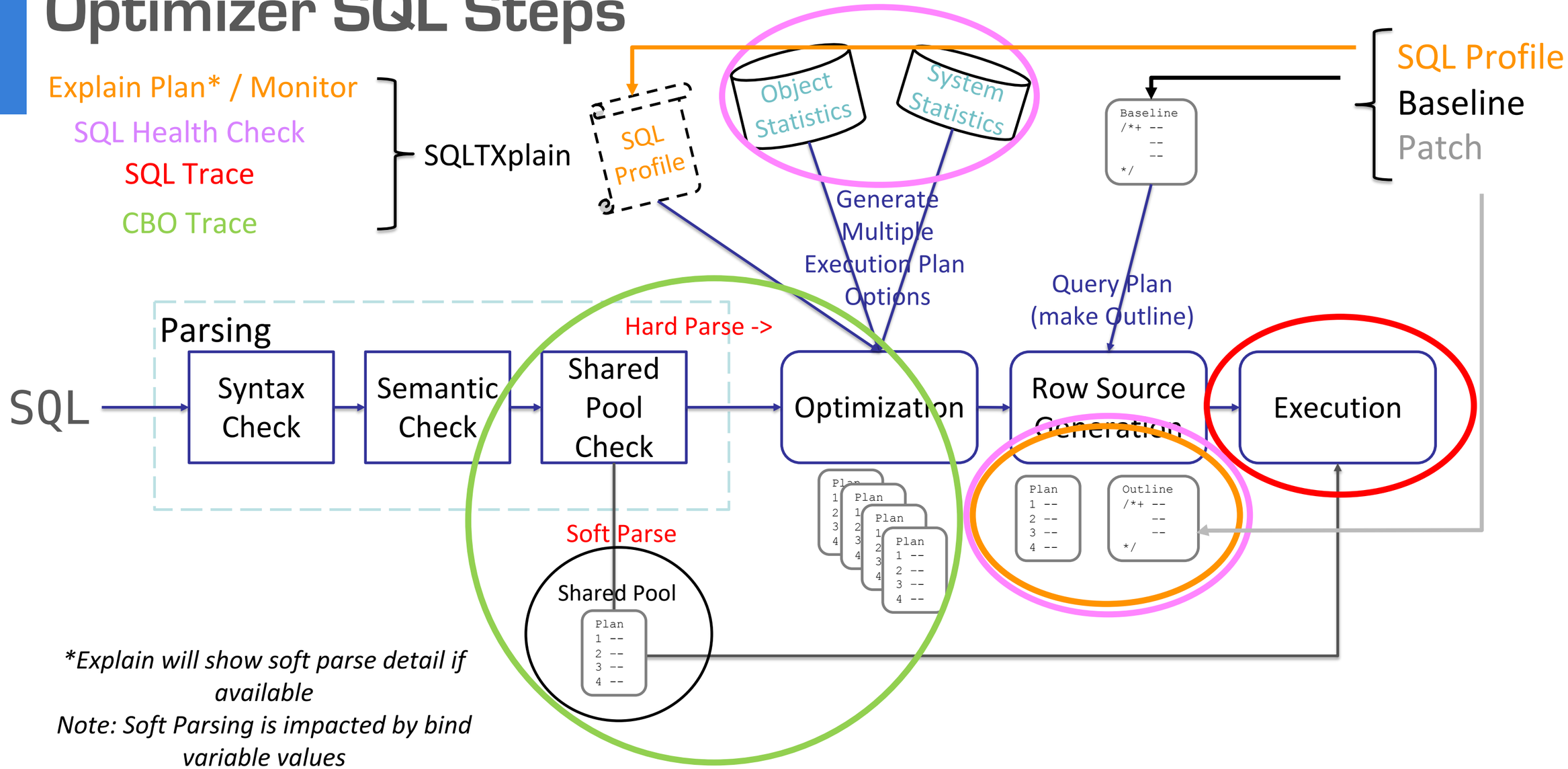
```
SYS.DBMS_SQLDIAG.CREATE_SQL_PATCH
```

Patch Example

```
sys.dbms_sqldiag.create_sql_patch(
                sql_id => '37090abayamah',
                hint_text => 'FULL(@"SEL$1" "TAB1"@"SEL$1")',
                name      =>'q0_patch');
```

Note: `hint_text` here is from the OUTLINE section of an explain plan

Optimizer SQL Steps



**Explain will show soft parse detail if available*

Note: Soft Parsing is impacted by bind variable values

Wrapping Up

Have a plan to find poor performing SQL
Have your tuning tools ready to use!
Know when each tool might be useful
Oracle will tune a lot of things for you, but sometimes it wont

Don’t trace everything! Be surgical or specific!

Practice in non-production!

References

- Diagnostics for Query Tuning Problems (Doc ID 68735.1)
- SQL Tuning Health-Check Script (SQLHC) (Doc ID 1366133.1)
- SQLT Usage Instructions (Doc ID 1614107.1)
- How To Collect 10046 Trace (SQL_TRACE) Diagnostics for Performance Issues (Doc ID 376442.1)
- EVENT: 10046 "enable SQL statement tracing (including binds/waits)" (Doc ID 21154.1)
- Interpreting Raw SQL_TRACE output (Doc ID 39817.1)
- How to Obtain Tracing of Optimizer Computations (EVENT 10053) (Doc ID 225598.1)

Follow Us Online!

Facebook.com/ViscosityNA

Linkedin.com/company/Viscosity-North-America

@ViscosityNA

Viscosity North America

@Viscosity_NA

VISCOSITY

NORTH AMERICA