TOP 7 PLAN STABILITY PITFALLS AND HOW TO AVOID THEM

Neil Chandler,
Chandler Systems

22. hroug
godišnja konferencija
17.-20.10.2017.

TOP 7 PLAN STABILITY PITFALLS AND HOW TO AVOID THEM

Neil Chandler, Chandler Systems

Oracle & SQL Server DBA

In IT since 1988

Working with Oracle since about 1991

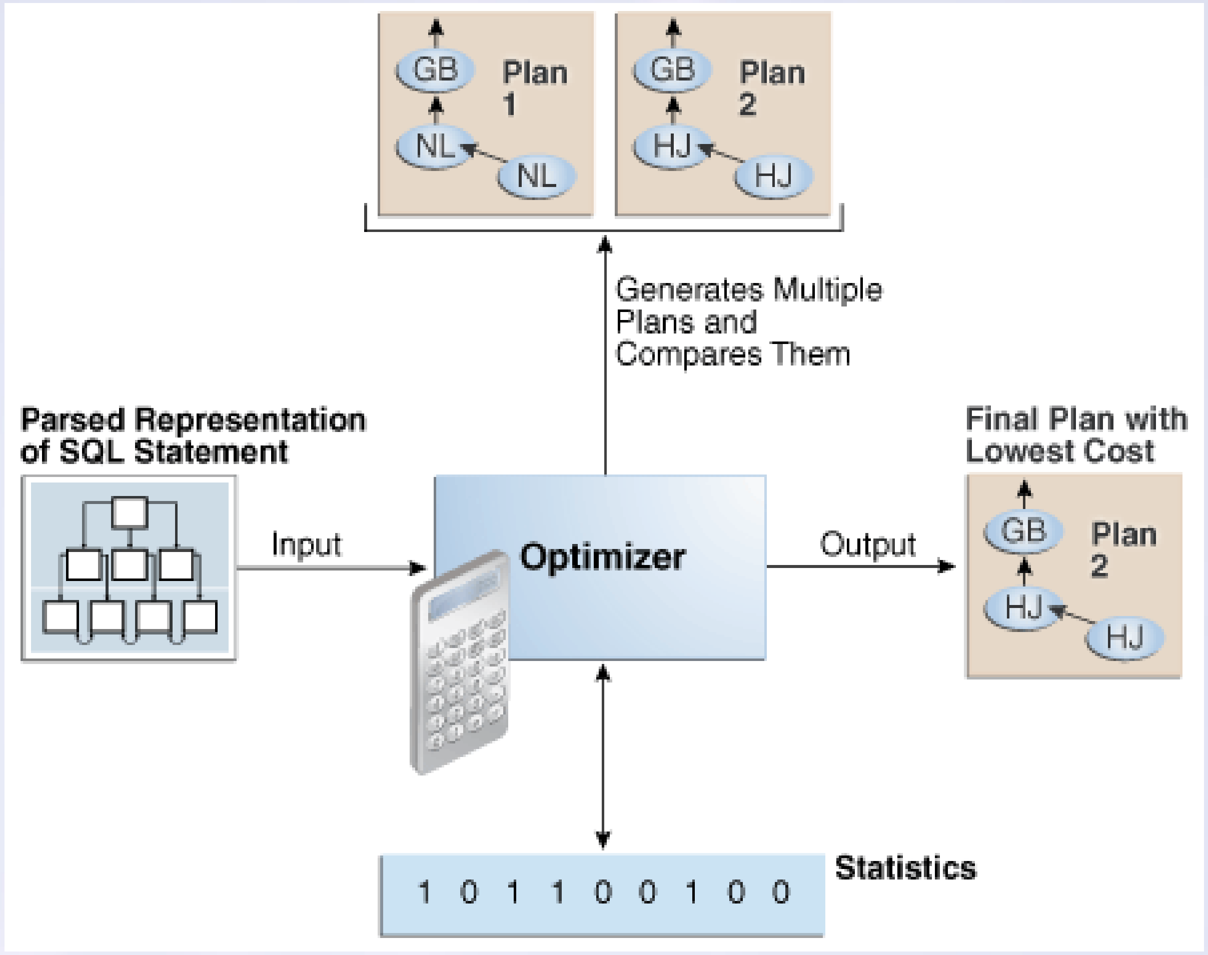
Chairman of the UKOUG RAC, Cloud, Infrastructure and Availability SIG

BLOG: http://chandlerdba.wordpress.com/

Tweets: @chandlerDBA

INTRODUCTION

When you write some SQL, Oracle runs it through the Optimizer to determine the best access path


GB
Plan 1
NL
NL
GB
Plan 2
HJ
HJ
Generates Multiple Plans and Compares Them
Parsed Representation of SQL Statement
Input
Optimizer
Output
Final Plan with Lowest Cost
GB
Plan 2
HJ
HJ
1 0 1 1 0 0 1 0 0
Statistics

INTRODUCTION

When you write some SQL, Oracle runs it through the Optimizer to determine the best access path

Oracle weighs up a lot of information about your query and has a limited number of guesses (2000) to determine the best way to get to the data

A 5-table join could have over 30m possible access paths, so with only 2000 guesses available, the Optimizer is going to get things wrong in the name of expediency.

Join Order = 1*2*3*4*5 = **120** possible permutations

Join Method = NL/HJ/MJ = 3*3*3*3 = 81 (**81*120 = 10,000**)

Data Access = FTS/IUqS/I(Rg)S/ISkipS/IFFS/*BITMconv/STAR/BloomF/bInSglVal/bInRgS*
= 5*5*5*5*5 = **3,000 (*10,000)**
= **30,000,000** possible access path

6-Table join is 720 * 243 * 15625 = **2,733,750,000** possible access paths...

INTRODUCTION

So the Optimizer ALWAYS gets it wrong

(for complex sql)

Sometimes it changes its mind and usually that makes only a minor difference

Sometimes it is not a minor difference

Sometimes it is not good at all

Sometimes it is very very bad

MY PLAN CHANGED

Why did Oracle change the plan?

What caused it to made a different decision?

How can I make the Optimizer make the “right” decision?

And most importantly, what am I really looking for from the Optimizer?

STATS

A common cause of a new SQL Execution plan is...

You added some data to your database (I hope you are all doing this)

You gathered statistics on at least one of the objects that you are accessing

Your plan aged out of the shared pool or was invalidated

Oracle re-optimized and came to a different conclusion about how to access the data based upon your new statistics

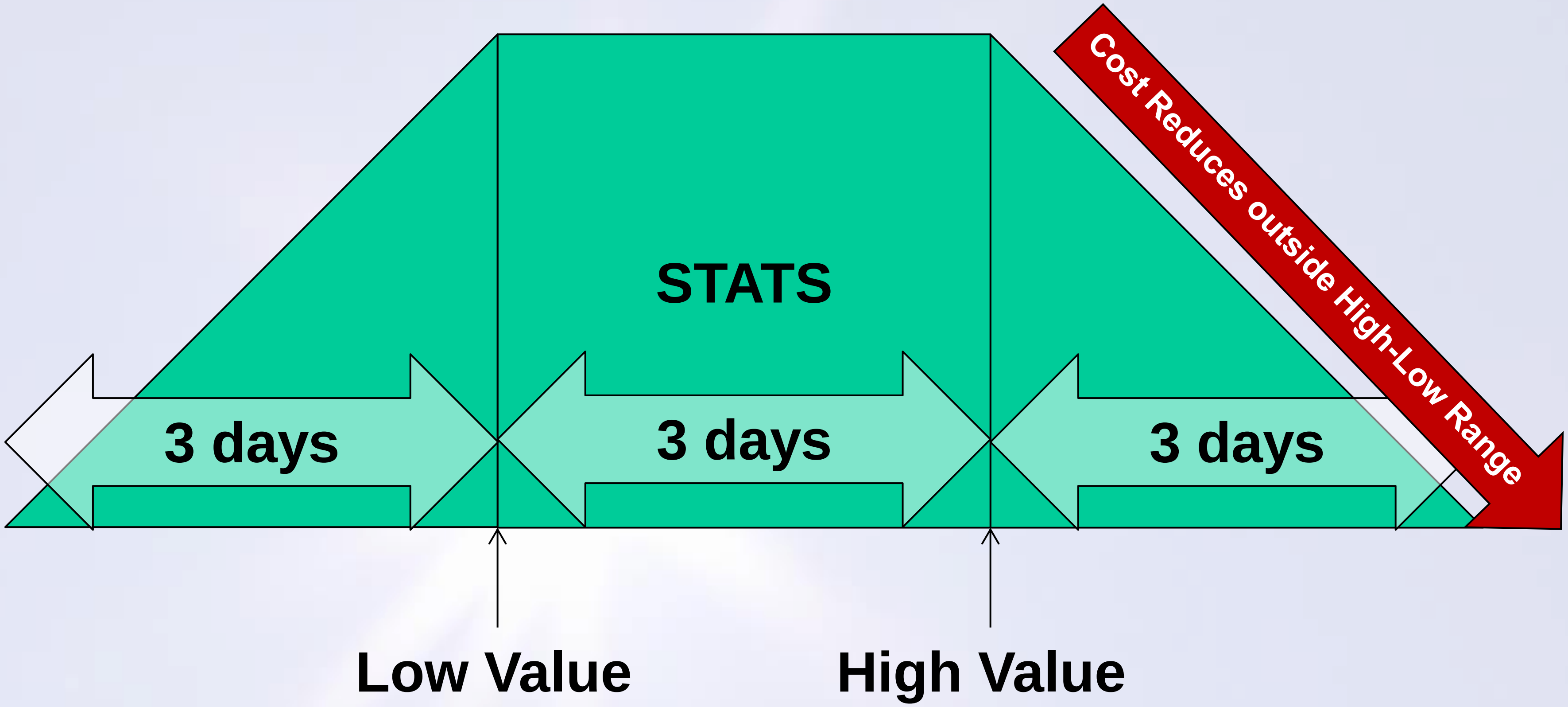
...alternatively...

You did nothing, maybe added data, never changed your stats, kept running the same query day after day, and suddenly the plan changed....

>> Demo 1 & 2

STATS

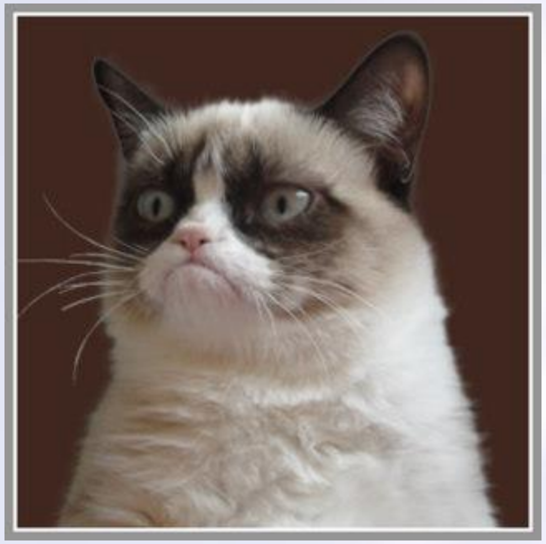
Statistical Decay



PLANS CHANGE

We can conclude from the 2 previous demonstrations that

- Your plans can change when you add data and gather stats
- Your plans can change when you don't add data and don't gather stats



Lets try *no* stats

DYNAMIC SAMPLING

No stats? Oracle needs to get some meta-data about your data to plug into the optimizer. Oracle uses **Dynamic Sampling**; reading some blocks randomly and having a look at the data in them:

DS Levels (parameter OPTIMIZER_DYNAMIC_SAMPLING)

0=off
2=use if one or more tables have no stats – samples 64 blocks *[default]*
4=Conjunctive (AND/OR) or Complex Expression (A+B=). 64 blocks read
5 to 9 = as level 4 but reads progressively more blocks, up to 4096.
10=sample ALL blocks for ALL statements (FTS)
or
(12c) turn to up to **11**... Uses **DS**, (even) if you have stats if the optimizer thinks it would be a good idea, and persists the stats for other queries

Consider setting Level 4+ in Data Warehouse environments

NO DEMO

CARDINALITY FEEDBACK : 11G

Used when your have:

- no stats
- a complex predicate (COLA+COLB=n) where Oracle can't calculate the cardinality
- multiple conjunctive/disjunctive filters (AND and OR)

Compares Estimated Row Counts (from the stats) to Actual Row Counts

11G – Marked in V$SQL_SHARED_CURSOR.USE_FEEDBACK_STATS to show use

First Parse gets Plan A, but it gets marked as having poor cardinality (i.e. bad stats)
2nd run, hard parse again and we get Plan B and we stick with Plan B
Plan ages out of shared pool, and back around the loop we go... Back to Plan A

When the plan is aged out of the shared pool, all of this information is lost

STATISTICS FEEDBACK: 12C

12C Cardinality Feedback renamed to **Statistics feedback**, and improved.

Use-case for Statistics feedback same as Cardinality Feedback.

V$SQL.IS_REOPTIMIZABLE="Y" causes a hard parse and a child cursor

By default (in 12.1 only) a SQL Plan Directive may also be created, to semi-permanently store the fact you have stats problems. SQL Plan directives are associated with COLUMNS not SQL's and so can affect every SQL.

You may also get some Extended Statistics created for column correlations

select * from DBA_SQL_PLAN_DIRECTIVES

There's only 2 type of directives so far...
DYNAMIC_SAMPLING
DYNAMIC_SAMPLING_RESULT (from 12.2)

SQL PLAN DIRECTIVES

Adaptive Statistics are not good for stability

If you have an OLTP system, you probably do not want Adaptive Stats enabled!

but you can't just DROP SQL Plan Directives to get rid of them.
They come back!

in 12.1:

DISABLE THEM:

```
dbms_spd.alter_sql_plan_directive(id,'ENABLED','NO');
```

But 53 weeks later, they will be auto-dropped, and re-appear

DISABLE the AUTO_DROP too:

```
dbms_spd.alter_sql_plan_directive(id,‘AUTO_DROP','NO');
```

SQL PLAN DIRECTIVES

12.2 – new init.ora parameters

optimizer_adaptive_plans=TRUE

This will disable **adaptive**

- Joins
- PX Distribution
- Star Transformation bitmap pruning

optimizer_adaptive_statistics=FALSE *<= switched OFF by default*

This will disable

- _optimizer_dsdir_usage_control
- _optimizer_use_feedback_for_join
- _optimizer_ads_for_pq

Check Bug: 22652059 ***if*** you want this back-ported to 12.1

BIND VARIABLE PEEKING

Bind Variable Peeking highlights a contradictory element within the Optimizer.

Plans are stored in the shared pool so we don't need to reparse the code.

We use bind variables so we can re-use more code.

With no need to reparse, we can re-execute the same SQL many times and save the parsing overhead.

But, if we have skewed datasets – non uniform distribution of data – we may want different plans for different data inputs.

So, this is what Oracle does to your plans...

>> Demo 3

ADAPTIVE CURSOR SHARING

From the Demo, you have seen Oracle use Adaptive Cursor Sharing to combat the problem with Bind Variable Peeking

Look at V$SQL view entry for the query

- **IS_BIND_SENSITIVE** - is Oracle even aware it should be adapting plans?
- **IS_BIND_AWARE** – is Oracle actively hard parsing previously parsed plan for new bind variable?
- **IS_SHAREABLE** – can the cursor be shared with other bind values

- You must run a bad plan before Oracle realises it must reparse the SQL
- SQL may never be deemed sharable and you get *many* child cursors (this gets better from 11.1.0.7 onwards)
- You may be better-off using literals in this circumstance
- Use BINDs *when appropriate*

HISTOGRAMS

Oracle loves Histograms
They appear columns which appear in predicates - *Skew has nothing to do with it!*

Histograms** use Adaptive Sampling to randomly select data to build the histogram
(**All Histograms pre 12C. Height-Balanced and Hybrid from 12C onwards)

Oracle 11.2:

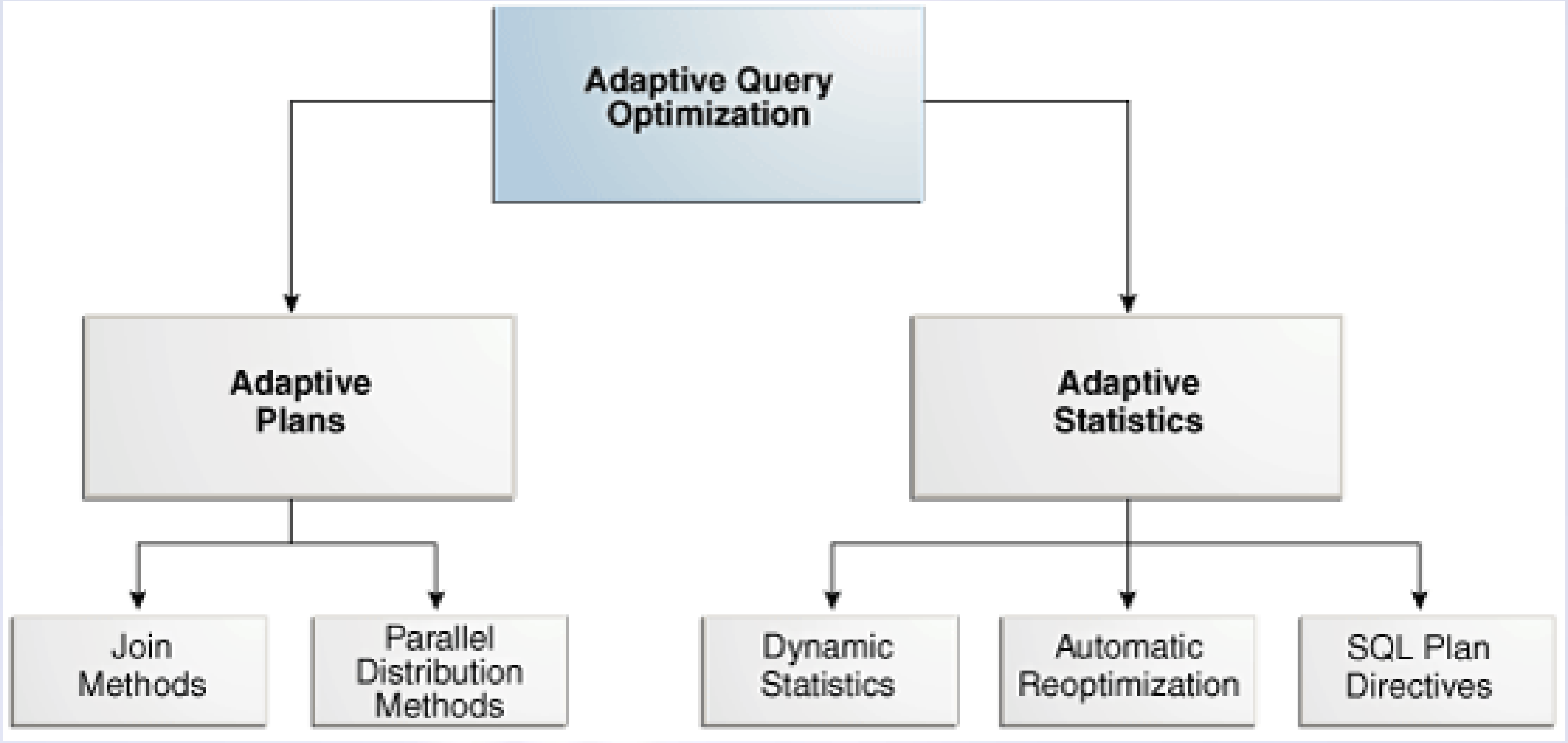
```
exec dbms_stats.set_global_prefs('method_opt','for all columns size repeat');
```

Oracle have changed the algorithm in 12C
They use the CURRENT number of buckets as an input *for stability*.
You **MUST NOT USE** for **all columns size repeat** after Oracle 11.2

Oracle 12C:

```
exec dbms_stats.set_global_prefs('method_opt','for all columns size 1');
exec dbms_stats.set_table_prefs(
     ownname=>'NEIL',
     tabname=>'TEST_REPEAT',
     pname  =>'method_opt',
     pvalue =>'for all columns size 1
               for columns size auto col1,col2,col3');
```

12C ADAPTIVE QUERY OPTIMIZATION



ADAPTIVE EXECUTION PLANS

In 12C, Oracle can change the plan WHILE the code is executing.

During execution, "Statistics Collector" is analyzing the blocks that are being read in, comparing the Estimated #Rows from Stats to what's actually happening

It can switch between a NESTED LOOP and HASH JOIN <u>during</u> execution

DBMS_XPLAN.DISPLAY_CURSOR shows the Adapted plan & tells you which bits of the plan have been switched-off.

```
[ select * from table(dbms_xplan.display_cursor(format=>'+adaptive')); ]
```

Check V$SQL.IS_RESOLVED_ADAPTIVE_PLAN
null = not adaptive / N = Adaptive aware but did not use / Y = used adaptive plan

>> Demo 4

OTHER REASONS FOR CHANGED PLANS

You might have changed some initialisation parameter, like memory pool sizes

Or Oracle might have helpfully changed them for you with Automatic Memory Management (**AMM**)

Or you might have modified optimizer-specific parameters

Or you might have dropped/added/changed an Index that you weren't using
but where Oracle was using the stats on that index
to make optimization decisions

Or you might have upgraded or even just patched the database
(NOTE: PSU's do not change the Optimizer)

WHAT'S A DBA TO DO?

I (usually) want consistency

Since the Cost-Based Optimizer arrived to replace the RBO in the 90's, we have been vulnerable to the vagaries of Optimizer guesses with too little time for it to do a thorough job.

What can you do about your plan changing?

Oracle have introduced 4 primary mechanisms over the last 20 years to cope with the CBO getting it wrong:

- Stored Outlines
- SQL Patches
- SQL Profiles
- SQL Plan Management, incorporating SQL Baselines

STORED OUTLINES

Stored Outlines were introduced long long ago (Oracle8), and they *try* to convince the optimizer to use a particular path.

You associate them with SQL Statements and they are basically a whole lot of SQL HINTS used to try to describe the Optimizer Access Path.

Unlike when a Developer/DBA uses one or two hints, trying to persuade a plan out of the optimizer, Outlines use lots. A dozen. Twenty. More.

I have found they generally worked pretty well, but are inflexible.

Once you put them in, they stick. Management (and DBA's) are scared to change or remove them, which you really should at upgrade time to look to use the new Optimizer features.

STORED OUTLINES

```
alter session set create_stored_outlines=true;

create outline EMP_DEPT for CATEGORY outtest ON
SELECT e.empno, e.ename, d.dname FROM emp e, dept d WHERE e.deptno = d.deptno;

select * from user_outline_hints;

NAME                NODE       STAGE   JOIN_POS HINT
EMP_DEPT               1           1          0 USE_MERGE(@"SEL$1" "E"@"SEL$1")
EMP_DEPT               1           1          0 LEADING(@"SEL$1" "D"@"SEL$1" "E"@"SEL$1")
EMP_DEPT               1           1          2 FULL(@"SEL$1" "E"@"SEL$1")
EMP_DEPT               1           1          1 INDEX(@"SEL$1" "D"@"SEL$1" ("DEPT"."DEPTNO"))
EMP_DEPT               1           1          0 OUTLINE_LEAF(@"SEL$1")
EMP_DEPT               1           1          0 ALL_ROWS
EMP_DEPT               1           1          0 DB_VERSION('11.2.0.2')
EMP_DEPT               1           1          0 OPTIMIZER_FEATURES_ENABLE('11.2.0.2')
EMP_DEPT               1           1          0 IGNORE_OPTIM_EMBEDDED_HINTS

                          select hash_value,sql_text from v$sql where sql_text like '%dept%'
    3931763680 SELECT e.empno, e.ename, d.dname FROM emp e, dept d WHERE e.deptno = d.deptno

DBMS_OUTLN.create_outline(hash_value=>3931763680, child_number=>0, category=>'OUTTEST');
```

STORED OUTLINES

```
ALTER SESSION SET query_rewrite_enabled=TRUE;
ALTER SESSION SET use_stored_outlines=OUTTEST;
SELECT e.empno, e.ename, d.dname FROM emp e, dept d WHERE e.deptno = d.deptno;

 1* select * from table(dbms_xplan.display_cursor(null,null,'ALL'))
PLAN_TABLE_OUTPUT
SQL_ID  3yrvgdrp5mwz0, child number 1
-------------------------------------
SELECT e.empno, e.ename, d.dname FROM emp e, dept d WHERE e.deptno = d.deptno
Plan hash value: 844388907
-----------------------------------------------------------------------------------------
| Id  | Operation                    | Name    | Rows  | Bytes | Cost (%CPU)| Time     |
-----------------------------------------------------------------------------------------
|   0 | SELECT STATEMENT             |         |       |       |     6 (100)|          |
|   1 |  MERGE JOIN                  |         |    14 |   364 |     6  (17)| 00:00:01 |
|   2 |   TABLE ACCESS BY INDEX ROWID| DEPT    |     4 |    52 |     2   (0)| 00:00:01 |
|   3 |    INDEX FULL SCAN           | PK_DEPT |     4 |       |     1   (0)| 00:00:01 |
|*  4 |   SORT JOIN                  |         |    14 |   182 |     4  (25)| 00:00:01 |
|   5 |    TABLE ACCESS FULL         | EMP     |    14 |   182 |     3   (0)| 00:00:01 |
-----------------------------------------------------------------------------------------
Note
-----
   - outline "EMP_DEPT" used for this statement
```

SQL PATCHES

Used to inject a “single hint” into a SQL Statement

```
sys.dbms_sql_diag_internal.i_create_patch (
 sql_text => 'select * from emp where dept_no = :deptno',
 hint_text => 'FULL',
 name => 'Full_Scan_Emp');
```

```
select * from emp where dept_no = :deptno
```

becomes

```
select /*+ FULL */ * from emp where dept_no = :deptno
```

- Does not require the SQL Tuning License & can use SQL Repair Advisor
- Works on Standard Edition
- From 12.2, it's "Official" not an *internal* function: **DBMS_SQLDIAG.CREATE_SQL_PATCH**

SQL PROFILES

SQL Profiles are, well, a bunch of hints attached to a SQL Plan. Sound familiar?

Their intention is to provide the Optimizer with additional statistics about how the data is related, generally using the **OPT_ESTIMATE** hint to change the cardinality of the join predicate or column correlation, to make the COST more accurate

Basically, SQL Profiles are statistics, and need to be re-tuned regularly to ensure the scaling factor remains correct.

They go <u>stale</u>, and they are not as easy to update as statistics

And they require the SQL Tuning License

And I don't like them but there are still use cases for them

SQL PLAN MANAGEMENT BASELINES

From 11G, we have been able to create Baselines within Oracle

The basic premise is that we restrict the plans a SQL statement may use
to one or more plans that we say is allowed

If a BETTER plan comes along, it can be captured
but will NOT be used unless we say that's OK

We have the ability to Evolve the plans, as our data evolves.
To allow the optimizer to select which plan is best from a defined list

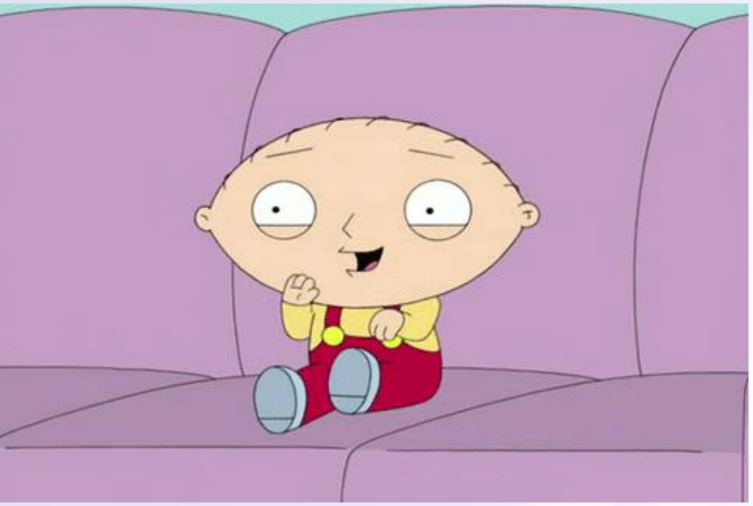
>>just before demo 5

SQL PLAN MANAGEMENT BASELINES

BREAKING NEWS!

SPM will be in Oracle 18 *Standard Edition* !

And it will be back-ported!



>>demo5

SPM

So, we have seen auto-capture of baselines using:

alter session set optimizer_capture_sql_plan_baselines=TRUE

Which captures any statement in a session executed more than once.

We have seen how that locks-down your plan, and only allows accepted plans to be used, but still captures other *potentially* good plans.

We tell Oracle that the plan is good to use, by Evolving the plan.

There are other ways to capture Baselines, too

BASELINE CAPTURE

We can capture any cursor currently in the cursor cache:

```
dbms_spm.load_plans_from_cursor_cache
```

We can capture baselines from a SQL Tuning Set:

```
dbms_spm.load_plans_from_sqlset
```

We can take Stored Outlines and migrate them to be Baselines

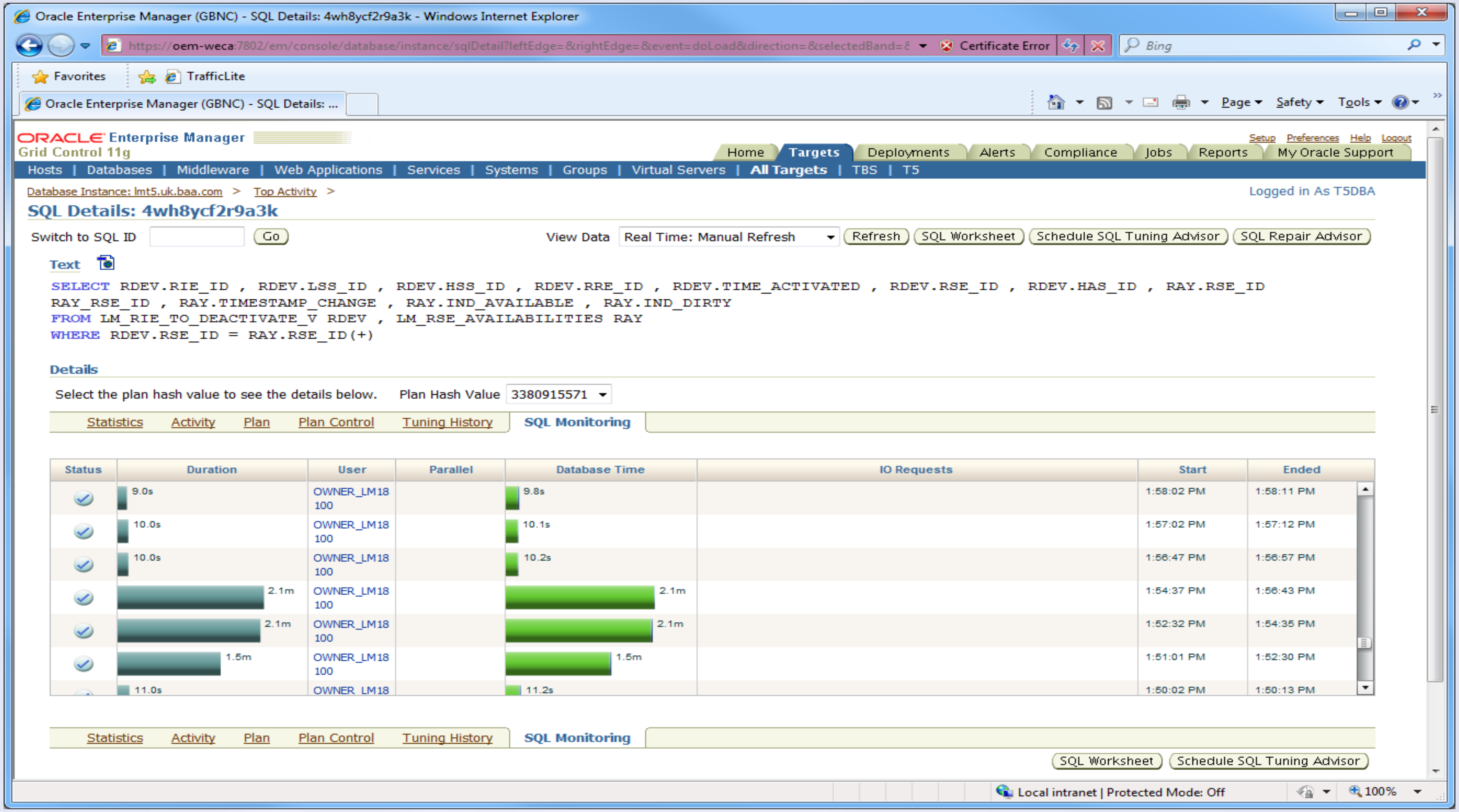
```
dbms_spm.migrate_stored_outline
```

And we can copy Baselines from system to system:

```
dbms_spm.create_stgtab_baseline
dbms_spm.pack_stgtab_baseline
expdp → impdp
dbms_spm.unpack_stgtab_baseline
```

BASELINE EXAMPLE

You can use it to react to situations too: SQL Monitor showing inconsistent plan



Oracle Enterprise Manager (GBNC) - SQL Details: 4wh8ycf2r9a3k - Windows Internet Explorer

https://oem-weca:7802/em/console/database/instance/sqlDetail?leftEdge=&rightEdge=&event=doLoad&direction=&selectedBand=&

Database Instance: lmt5.uk.baa.com > Top Activity >

Logged in As T5DBA

SQL Details: 4wh8ycf2r9a3k

Switch to SQL ID

View Data: Real Time: Manual Refresh

Text

```
SELECT RDEV.RIE_ID , RDEV.LSS_ID , RDEV.HSS_ID , RDEV.RRE_ID , RDEV.TIME_ACTIVATED , RDEV.RSE_ID , RDEV.HAS_ID , RAY.RSE_ID
RAY_RSE_ID , RAY.TIMESTAMP_CHANGE , RAY.IND_AVAILABLE , RAY.IND_DIRTY
FROM LM_RIE_TO_DEACTIVATE_V RDEV , LM_RSE_AVAILABILITIES RAY
WHERE RDEV.RSE_ID = RAY.RSE_ID(+)
```

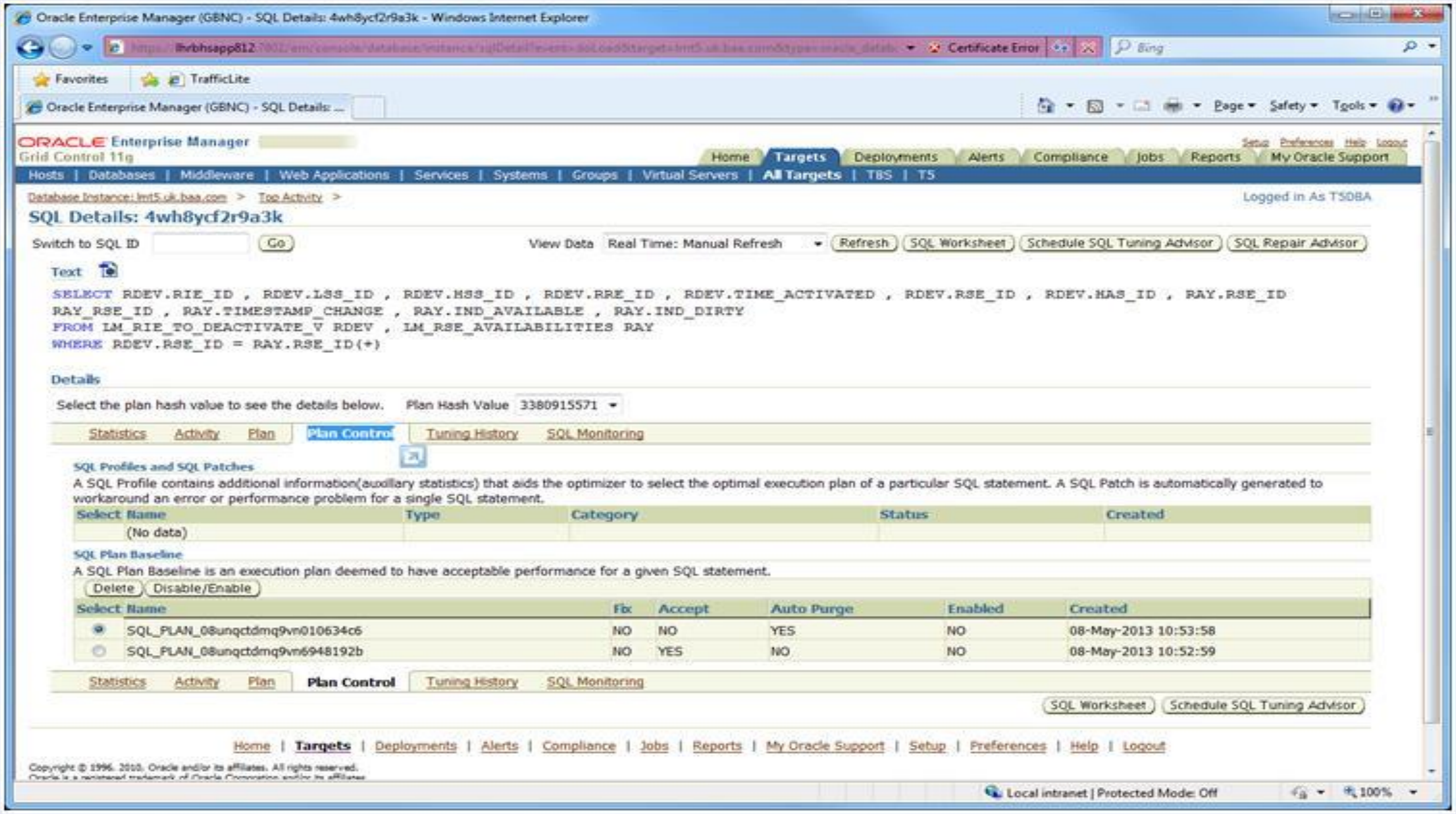
Details

Select the plan hash value to see the details below. Plan Hash Value 3380915571

Status	Duration	User	Parallel	Database Time	IO Requests	Start	Ended
✓	9.0s	OWNER_LM18 100		9.8s		1:58:02 PM	1:58:11 PM
✓	10.0s	OWNER_LM18 100		10.1s		1:57:02 PM	1:57:12 PM
✓	10.0s	OWNER_LM18 100		10.2s		1:56:47 PM	1:56:57 PM
✓	2.1m	OWNER_LM18 100		2.1m		1:54:37 PM	1:56:43 PM
✓	2.1m	OWNER_LM18 100		2.1m		1:52:32 PM	1:54:35 PM
✓	1.5m	OWNER_LM18 100		1.5m		1:51:01 PM	1:52:30 PM
	11.0s	OWNER_LM18		11.2s		1:50:02 PM	1:50:13 PM

PLAN INCONSISTENT

Plan was still in the cursor cache, so we captured it and Enabled it


Oracle Enterprise Manager (GBNC) - SQL Details: 4wh8ycf2r9a3k - Windows Internet Explorer
Certificate Error
Favorites
TrafficLite
Page
Safety
Tools
ORACLE Enterprise Manager
Grid Control 11g
Home | Targets | Deployments | Alerts | Compliance | Jobs | Reports | My Oracle Support
Hosts | Databases | Middleware | Web Applications | Services | Systems | Groups | Virtual Servers | All Targets | TBS | T5
Database Instance: lmt5.uk.baa.com > Top Activity >
Logged in As TSDBA
Switch to SQL ID
Go
Refresh
SQL Worksheet
Schedule SQL Tuning Advisor
SQL Repair Advisor
Statistics | Activity | Plan | Plan Control | Tuning History | SQL Monitoring
Delete
Disable/Enable
Local intranet | Protected Mode: Off
100%

SQL Details: 4wh8ycf2r9a3k

View Data: Real Time: Manual Refresh

Text

```
SELECT RDEV.RIE_ID , RDEV.LSS_ID , RDEV.MSS_ID , RDEV.RRE_ID , RDEV.TIME_ACTIVATED , RDEV.RSE_ID , RDEV.HAS_ID , RAY.RSE_ID
RAY_RSE_ID , RAY.TIMESTAMP_CHANGE , RAY.IND_AVAILABLE , RAY.IND_DIRTY
FROM LM_RIE_TO_DEACTIVATE_V RDEV , LM_RSE_AVAILABILITIES RAY
WHERE RDEV.RSE_ID = RAY.RSE_ID(+)
```

Details

Select the plan hash value to see the details below. Plan Hash Value: 3380915571

SQL Profiles and SQL Patches

A SQL Profile contains additional information(auxillary statistics) that aids the optimizer to select the optimal execution plan of a particular SQL statement. A SQL Patch is automatically generated to workaround an error or performance problem for a single SQL statement.

Select	Name	Type	Category	Status	Created
	(No data)				

SQL Plan Baseline

A SQL Plan Baseline is an execution plan deemed to have acceptable performance for a given SQL statement.

Select	Name	Fix	Accept	Auto Purge	Enabled	Created
●	SQL_PLAN_08unqctdmq9vn010634c6	NO	NO	YES	NO	08-May-2013 10:53:58
○	SQL_PLAN_08unqctdmq9vn6948192b	NO	YES	NO	NO	08-May-2013 10:52:59

Home | Targets | Deployments | Alerts | Compliance | Jobs | Reports | My Oracle Support | Setup | Preferences | Help | Logout

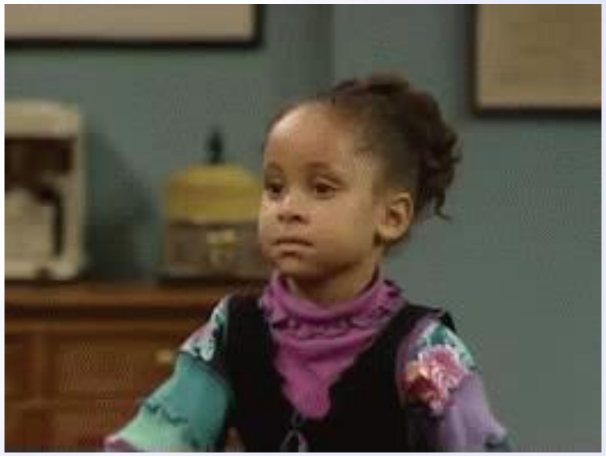
Copyright © 1996, 2010, Oracle and/or its affiliates. All rights reserved.

EVOLVE!

So, we've gone to a lot of trouble to restrict our plans, to <u>ensure</u> we have a neat and tidy set of baselines in 12C.

So, overnight, Oracle runs the "sql tuning advisor" autotask, and runs…

`SYS_AUTO_SPM_EVOLVE_TASK`
which run and evolves all of our plans



AUTO EVOLVE!

```
SELECT PARAMETER_NAME, PARAMETER_VALUE AS "VALUE" FROM DBA_ADVISOR_PARAMETERS
WHERE TASK_NAME = 'SYS_AUTO_SPM_EVOLVE_TASK'
AND PARAMETER_NAME in ('ACCEPT_PLANS','TIME_LIMIT') ORDER BY 1;
```

```
PARAMETER_NAME               VALUE
---------------------------- ----------
ACCEPT_PLANS                 TRUE
TIME_LIMIT                   3600
```

You might want to consider disabling it

```
exec DBMS_SPM.SET_EVOLVE_TASK_PARAMETER
('SYS_AUTO_SPM_EVOLVE_TASK', 'ACCEPT_PLANS', 'false');
```

CONCLUSION

Execution plans are important

Consistency is good, especially for OLTP

Baselining plans is no useful, but you need to keep on top of it. Monitor for new plans, copy them to DEV/UAT, test them and evolve the good ones, yourself! Be aware of the parse overhead!

Keep evolving the plans to ensure you take advantage of your changing data and the Optimizers ability to access it successfully

ANY QUESTIONS

?

BLOG: http://chandlerdba.com
EMAIL: neil@chandler.uk.com
TWITTER: @chandlerDBA

EXADATA STORAGE CELLS

The plan remains the same, but the execution time changes dramatically

There are Dynamic Storage Cell "Indexes" with High/Low values for every 1M of data

Only 8 Indexes per table are allowed, built dynamically on the 1st 8 predicates used.

If a 9th predicate is used against a table, and it is "better" (more selective) than a former index, it replaces the "least useful" index. Dynamically.
Your super-fast super-selective dynamic index just vanished.

First run is always slow as indexes are dynamic.
Indexes do not persist across reboots
Indexes do not persist if a more selective column is referenced in a WHERE clause
Only for FTS and Index FFS

Wide tables bad