

# Challenges and Chances of the 11g Query Optimizer

Christian Antognini  
Senior Principal Consultant  
December 12, 2011

BASEL    BERN    LAUSANNE    ZÜRICH    DÜSSELDORF    FRANKFURT A.M.    FREIBURG I.BR.    HAMBURG    MÜNCHEN    STUTTGART    WIEN

1

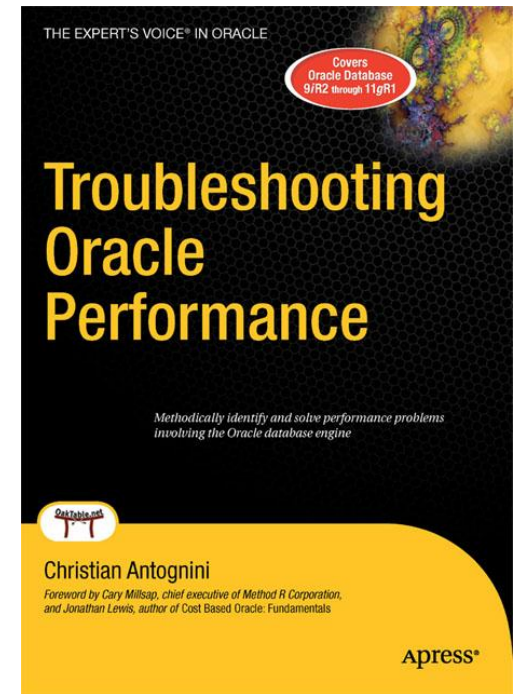
2011 © Trivadis

Challenges and Chances of the 11g Query Optimizer  
December 12, 2011

**trivadis**  
makes IT easier. ■ ■ ■

# Who Am I

- Senior principal consultant, trainer and partner at Trivadis in Zurich (CH)
  - christian.antognini@trivadis.com
  - Blogging at <http://antognini.ch>
- Focus: get the most out of Oracle Database
  - Logical and physical database design
  - Query optimizer
  - Application performance management and tuning
- Author of Troubleshooting Oracle Performance (Apress, 2008)
- Proud member of OakTable Network



# Introduction

- With every new release the query optimizer is enhanced
- This presentation provides an overview of central changes
  - Covered versions: 11.1.0.6 - 11.2.0.3
  - Features related to parallel processing and Exadata are not covered
- Behavioral changes and challenges are marked with the following images



Behavioral  
Change

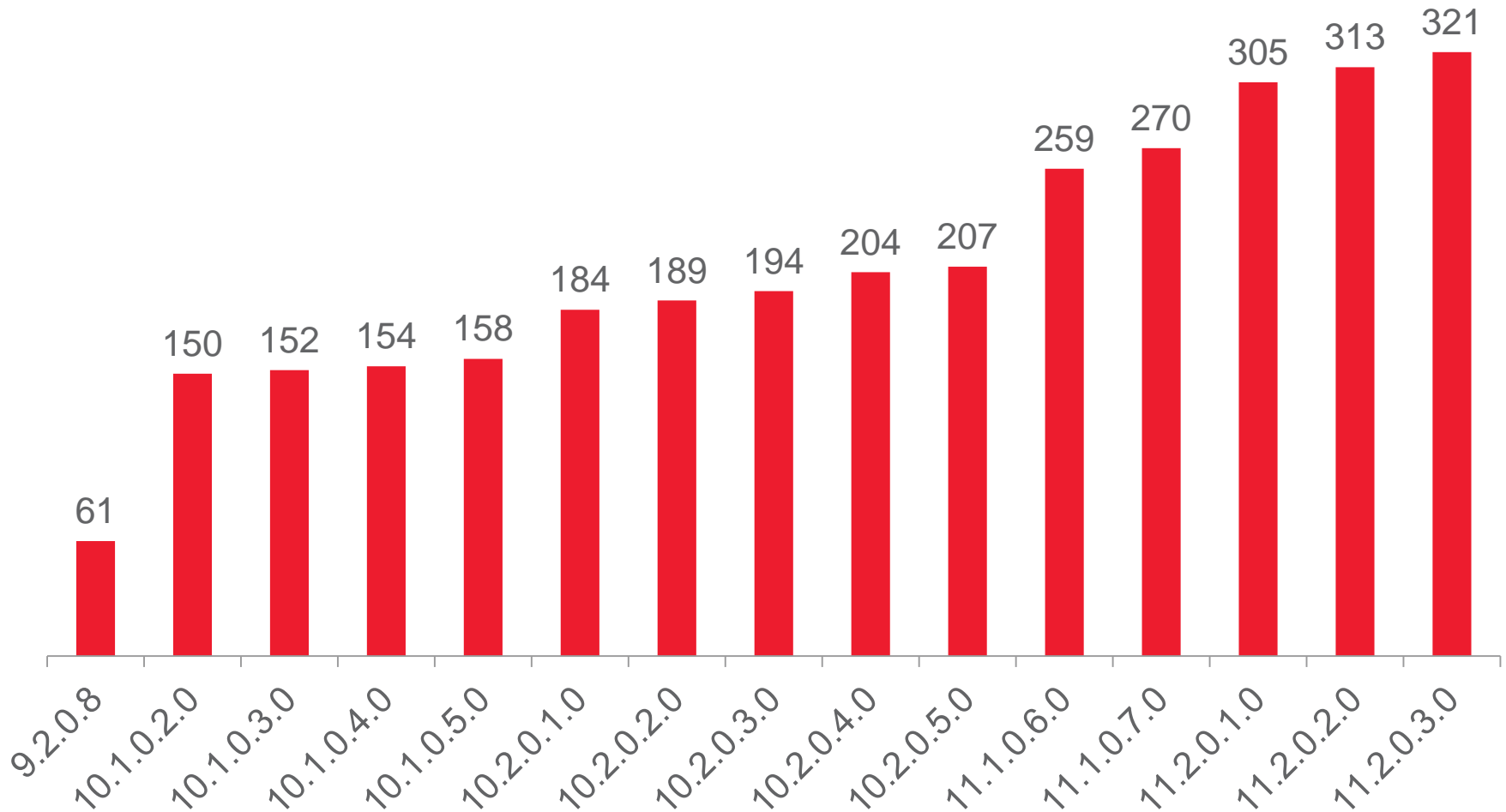
Challenge



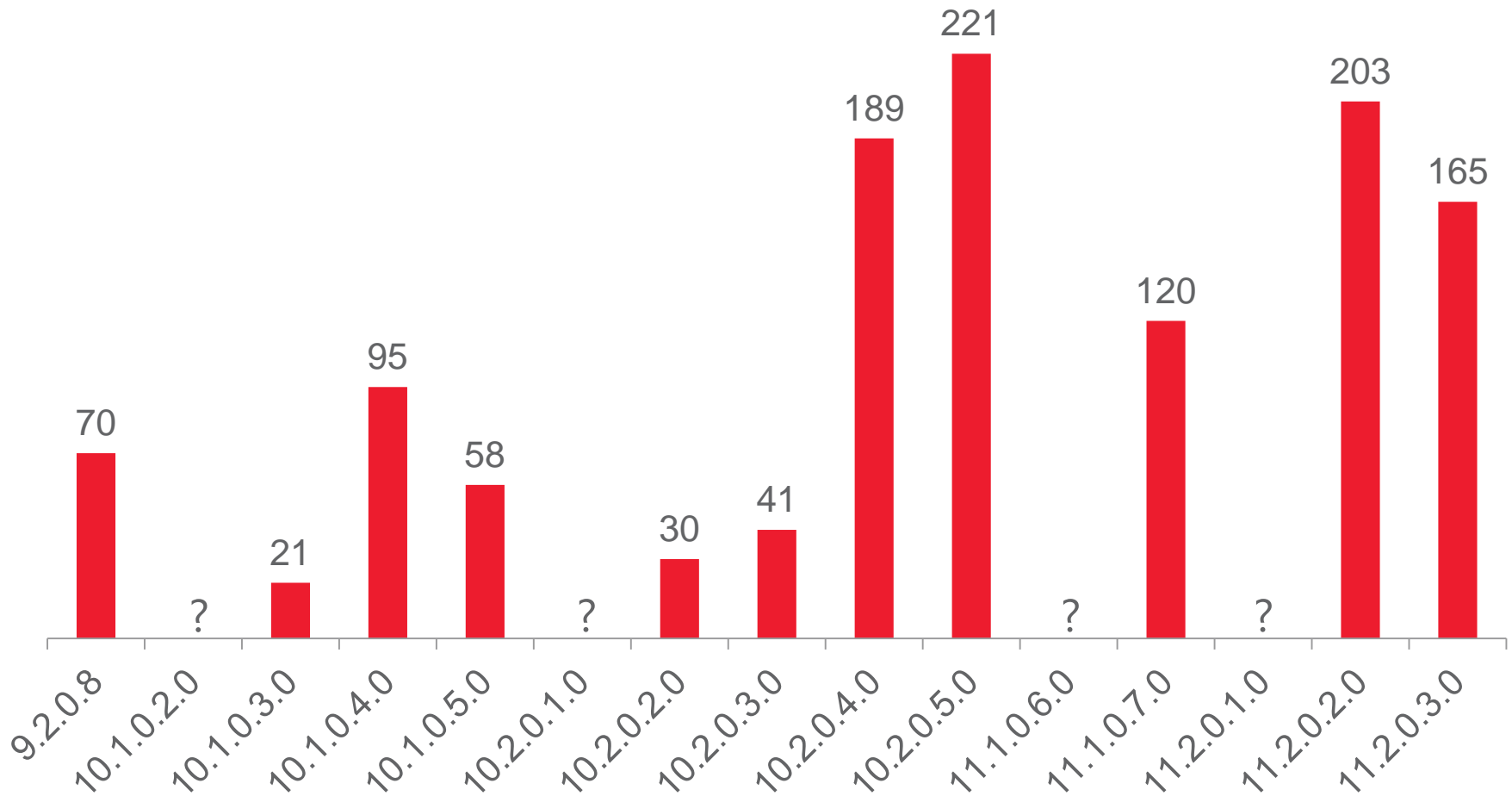
# AGENDA

1. Observations
2. Indexing
3. Optimization Techniques
4. System and Object Statistics (DBMS\_STATS)
5. Plan Stability

# Number of Query Optimizer Parameters by Release



# Number of Query Optimizer Bugs Fixed by Patchset



# AGENDA

1. Observations
2. Indexing
3. Optimization Techniques
4. System and Object Statistics (DBMS\_STATS)
5. Plan Stability

# Invisible Indexes

- As of 11.1 indexes can be hidden from the query optimizer
- The visibility of an index can be changed

```
ALTER INDEX ... [ INVISIBLE | VISIBLE ]
```

- The visibility can be specified for new indexes

```
CREATE INDEX ... ON ... ( ... ) [ INVISIBLE | VISIBLE ]
```

- The visibility can be controlled at the statement and session level
  - Hints: (NO\_)USE\_INVISIBLE\_INDEXES
  - Parameter: OPTIMIZER\_USE\_INVISIBLE\_INDEXES (default FALSE)
- Index hints are not honored
- Invisible indexes are regularly maintained and used for setting locks



# Index Support for Linguistic LIKE

- Up to 10.2 linguistic indexes are not used for evaluating predicates based on LIKE
- As of 11.1 it works as expected

```
CREATE INDEX fbi ON emp (nlssort(ename, 'nls_sort=binary_ci'));
ALTER SESSION SET nls_sort=binary_ci;
ALTER SESSION SET nls_comp=linguistic;
SELECT * FROM emp WHERE ename LIKE 'sco%';
```

Operation	Name
SELECT STATEMENT	
TABLE ACCESS BY INDEX ROWID	EMP
<b>INDEX RANGE SCAN</b>	<b>FBI</b>

# INDEX REBUILD and Statistics History

- Up to 11.1 a rebuild does not save old statistics in the history
  - A restore leads to missing statistics
  - The only workaround is to manually save statistics
- As of 11.2 it works as expected



# AGENDA

1. Observations
2. Indexing
3. Optimization Techniques
4. System and Object Statistics (DBMS\_STATS)
5. Plan Stability

# Full Outer Join

- Up to 10.2 the full outer join syntax is supported, but the query optimizer rewrites it to a UNION ALL query
  - As a result joined tables are accessed twice
- As of 11.1, for equi-joins only, it works as expected

```
SELECT * FROM t1 FULL OUTER JOIN t2 ON t1.id = t2.id
```

Operation	Name
VIEW	VW_FOJ_0
<b>HASH JOIN FULL OUTER</b>	
TABLE ACCESS FULL	T1
TABLE ACCESS FULL	T2

- Hints: (NO\_)NATIVE\_FULL\_OUTER\_JOIN

# Join-Filter Pruning

- Subquery pruning allows the utilization of hash joins instead of nested loops to perform partition pruning on join conditions
  - Part of the processing is executed twice
- As of 11.1 the double execution is avoided thanks to join-filter pruning

```
SELECT * FROM t t1, t t2 WHERE t1.n = t2.n AND t1.id = 1
```

Operation	Name	Pstart	Pstop
HASH JOIN			
<b>PART JOIN FILTER CREATE</b>	<b>:BF0000</b>		
TABLE ACCESS BY GLOBAL INDEX ROWID	T	ROWID	ROWID
INDEX UNIQUE SCAN	T_PK		
<b>PARTITION RANGE JOIN-FILTER</b>		<b>:BF0000</b>	<b>:BF0000</b>
TABLE ACCESS FULL	T	<b>:BF0000</b>	<b>:BF0000</b>

- Hints: (NO\_)PX\_JOIN\_FILTER

# Table Expansion

- As of 11.2, to cope with partially indexed partitioned tables, different access paths can be used for different partitions

```
SELECT count(d)
FROM t
WHERE d BETWEEN to_date('2009-04-30 23:00','yyyy-mm-dd hh24:mi')
           AND to_date('2009-05-02 01:00','yyyy-mm-dd hh24:mi')
```

Operation	Name	Pstart	Pstop
SORT AGGREGATE			
VIEW	VW_TE_2		
<b>UNION-ALL</b>			
PARTITION RANGE SINGLE		5	5
INDEX RANGE SCAN	I	5	5
PARTITION RANGE SINGLE		4	4
TABLE ACCESS FULL	T	4	4

- Hints: (NO)\_EXPAND\_TABLE

# Join Factorization

- Up to 11.1 each branch of a UNION ALL is evaluated independently
- As of 11.2 common processing can be factorized

```
SELECT * FROM t1, t2 WHERE t1.id = t2.id AND t2.id < 10
UNION ALL
SELECT * FROM t1, t2 WHERE t1.id = t2.id AND t2.id > 990
```

```
-----
| Operation                | Name                               |
-----
| HASH JOIN                 |                                     |
| VIEW                      | VW_JF_SET$3DF3FC68               |
| UNION-ALL                 |                                     |
| TABLE ACCESS FULL      | T2                                 |
| TABLE ACCESS FULL      | T2                                 |
| TABLE ACCESS FULL      | T1                                 |
-----
```

- Hints: (NO\_)FACTORIZE\_JOIN

# OR Expansion

- Up to 11.2.0.1 FBI cannot be used with OR expansion
- As of 11.2.0.2 this restriction no longer exists

```
SELECT *
FROM t
WHERE floor(n1) = 8 OR floor(n2) = 4
```

Operation	Name
SELECT STATEMENT	
CONCATENATION	
TABLE ACCESS BY INDEX ROWID	T
INDEX RANGE SCAN	I2_FLOOR
TABLE ACCESS BY INDEX ROWID	T
INDEX RANGE SCAN	I1_FLOOR

- Hints: OR\_EXPAND, NO\_EXPAND

# Join Elimination

- Join elimination purpose is to avoid executing a join even if a SQL statement explicitly calls for it
- In 10.2 and 11.1 only PK-FK relations are used for the elimination
- As of 11.2 also self-joins based on the PK are used

```
SELECT t1.*, t2.* FROM t t1, t t2 WHERE t1.id = t2.id
```

```
-----
| Operation          | Name |
-----
| TABLE ACCESS FULL| T    |
-----
```

- Hints: (NO\_)ELIMINATE\_JOIN

# Subquery Unnesting

- Subquery unnesting purpose is to rewrite [NOT] IN and [NOT] EXISTS subqueries into regular joins
  - The initial implementation dates back several major releases
- As of 11.2 also subqueries containing set operators can be unnested

```
SELECT * FROM t2 WHERE EXISTS (SELECT 1 FROM t1 WHERE t1.id =
t2.t1_id UNION ALL SELECT 1 FROM t1 WHERE t1.id = t2.t1_id)
```

Operation	Name
NESTED LOOPS	
NESTED LOOPS	
VIEW	VW_SQ_1
HASH UNIQUE	
UNION-ALL	
TABLE ACCESS FULL	T1
TABLE ACCESS FULL	T1
INDEX RANGE SCAN	T2_T1_ID
TABLE ACCESS BY INDEX ROWID	T2

- Hints: (NO\_)UNNEST

# AGENDA

1. Observations
2. Indexing
3. Optimization Techniques
4. System and Object Statistics (DBMS\_STATS)
5. Plan Stability

# Workload System Statistics

- In 11.2 the workload system statistics gathered by DBMS\_STATS are broken
  - SREADTIM and MREADTIM are very high
- Workaround: they have to be manually tweaked
- Solutions:
  - Install patch set 11.2.0.3
  - Install patch 9842771 (available for 11.2.0.1 and 11.2.0.2)



# Object Statistics – Default Preferences

- Up to 10.2 default values for parameters can only be set database wide
  - DBMS\_STATS.SET\_PARAM is available for this purpose
- As of 11.1 default values for parameters, called preferences, can be set at table level as well

```
dbms_stats.set_table_prefs (ownname => user,  
                             tabname => 'T',  
                             pname   => 'cascade',  
                             pvalue  => 'true')
```

- At least it is possible to rely on a generic call at the schema or database level to gather all statistics
- SET\_PARAM is deprecated in favor of SET\_GLOBAL\_PREFS

# Object Statistics – Auto Sample Size

- Up to 10.2 the auto sample size is of limited usage
  - More often than not a fix value is specified
- 11.1 introduces a key enhancement
  - The underlying algorithm is new
  - Representative statistics can be gathered in a much shorter timeframe
- Bug 9056912 - Not able to gather statistics on external tables
  - Gathering statistics at the table level raises an error
  - External tables are not considered when gathering statistics at the schema or database level
  - Fixed in 11.2.0.2



# Object Statistics – Pending Statistics

- Up to 10.2 gathered statistics are immediately made available to the query optimizer
- As of 11.1 it is possible to separate the gathering from the publishing
  - This is done by disabling the publishing

```
dbms_stats.set_table_prefs (ownname => user,  
                             tabname => 'T',  
                             pname   => 'publish',  
                             pvalue  => 'false')
```

- Statistics that are gathered but not published are called *pending statistics*
  - The query optimizer uses pending statistics only when OPTIMIZER\_USE\_PENDING\_STATISTICS=TRUE
  - Pending statistics can be published, compared, deleted and exported

# Object Statistics – Incremental Statistics

- Up to 10.2 DBMS\_STATS gathers global statistics by accessing all underlying partitions
- As of 11.1 incremental statistics can be enabled

```
dbms_stats.set_table_prefs (ownname => user,  
                           tabname => 'T',  
                           pname    => 'incremental',  
                           pvalue   => 'true')
```

- When enabled, a synopsis summarizes what is stored in each partition and, therefore, not all partitions have to be accessed
- Synopses have to be created to take advantage from them!
  - Enabling incremental statistics is not enough
- Auto sample size is also required

# Object Statistics – Extended Statistics on Expressions

- As of 11.1 it is possible to gather statistics on expressions
  - Called *extended statistics*
  - Only makes sense for expressions used in WHERE clauses
  - With them the query optimizer might improve cardinality estimations

```
dbms_stats.create_extended_stats(ownname => user,  
                                tabname  => 'T',  
                                extension => '(trunc(col))')
```

- CREATE\_EXTENDED\_STATS only adds a hidden virtual column
  - Since the column is hidden, it is transparent for the application
  - Statistics are gathered with the usual procedures

## Object Statistics – Extended Statistics on Column Groups

- Extended statistics are not limited to expressions, they can also be created for column groups
  - Useful to track correlated columns

```
dbms_stats.create_extended_stats(ownname => user,  
                                tabname  => 'T',  
                                extension => '(col1,col2)')
```

- The query optimizer can only take advantage of extended statistics when conditions are based on equality

# Object Statistics – Seeding Column Groups

- As of 11.2.0.2 it is possible instruct DBMS\_STATS to automatically detect which column groups should be created

## 1. Record usage of column groups for x seconds

```
dbms_stats.seed_col_usage(time_limit => 30)
```

## 2. Generate a report of column groups usage (optional)

```
dbms_stats.report_col_usage(ownname => user, tabname => NULL)
```

## 3. Create extensions based on groups of columns seen in workload

```
dbms_stats.create_extended_stats(ownname => user,  
                                tabname => NULL)
```

# Object Statistics – Comparing Statistics

- As of 11.1 DBMS\_STATS provides table functions that make easy the comparison of two sets of statistics
  - DIFF\_TABLE\_STATS\_IN\_HISTORY (backported in 10.2.0.4 as well)
  - DIFF\_TABLE\_STATS\_IN\_STATTAB (backported in 10.2.0.4 as well)
  - DIFF\_TABLE\_STATS\_IN\_PENDING

```
dbms_stats.diff_table_stats_in_pending(ownname      => user,  
                                       tabname       => 'T',  
                                       time_stamp    => NULL,  
                                       pctthreshold => 10))
```

# Object Statistics – Locks not Exported

- Up to 11.1 locks related to object statistics are not exported when there are no object statistics
  - Locks disappear with an export/import
- As of 11.2 this is no longer the case



# JOB\_QUEUE\_PROCESSES



- Up to 11.1 the parameter has no impact on jobs scheduled through DBMS\_SCHEDULER
- As of 11.2 this is no longer the case, e.g. setting the parameter to 0 disables the automatic statistics gathering
- Documentation bug 10008042
  - Fixed in the current version

# AGENDA

1. Observations
2. Indexing
3. Optimization Techniques
4. System and Object Statistics (DBMS\_STATS)
5. Plan Stability

# CURSOR\_SHARING

- At last the value SIMILAR is deprecated
  - 1169017.1 - Deprecating the cursor\_sharing = 'SIMILAR' setting
- Even though it was never recommended to set CURSOR\_SHARING to SIMILAR, it is good news that Oracle officially deprecated it

# SQL Plan Baselines – Purpose and Concept

- SQL plan baselines provide a new way to achieve plan stability
  - Might be seen as “enhanced stored outlines”
  - Only available in Enterprise Edition
- For specific SQL statement several “execution plans” might be stored in the data dictionary
  - In reality list of hints are stored
- When a SQL plan baseline exists, the query optimizer can only use verified (accepted) execution plans

## SQL Plan Baselines – Capture

- They are captured either automatically or manually
- Automatically
  - OPTIMIZER\_CAPTURE\_SQL\_PLAN\_BASELINES = TRUE (default is FALSE)
  - Execution plans associated to new SQL plan baselines are automatically accepted
- Manually – through the DBMS\_SPM package
  - LOAD\_PLANS\_FROM\_SQLSET
  - LOAD\_PLANS\_FROM\_CURSOR\_CACHE
  - Execution plans are automatically accepted

# SQL Plan Baselines – Evolution

- The query optimizer can add new execution plans to available SQL plan baselines
- It cannot use them, though
  - They must be accepted before being considered
- To verify whether new execution plans leads to better performance, an evolution is required
- Manually – through the DBMS\_SPM package
  - EVOLVE\_SQL\_PLAN\_BASELINE
- Automatically – through a SQL tuning task
  - Scheduled during the maintenance window
  - Tuning Pack required

# Stored Outlines

- Excerpt from *Oracle Database Performance Tuning Guide 11.2*

**Note:** Stored outlines will be desupported in a future release in favor of SQL plan management. In Oracle Database 11g Release 1 (11.1), stored outlines continue to function as in past releases. However, Oracle strongly recommends that you use SQL plan management for new applications. SQL plan management creates SQL plan baselines, which offer superior SQL performance and stability compared with stored outlines.

- With Standard Edition only stored outlines are available



# Adaptive Cursor Sharing – Concept



- Fundamental principles up to 10.2
  - Cursors can only be shared in case of full text match
  - Cursors based on statements using bind variables are shared even though it is sub-optimal to do so
- As of 11.1 cursors might be bind aware
  - By default new parent cursors are created not bind aware
  - Cursors automatically becomes bind aware in case of significant variation in row source cardinalities
    - Provided STATISTICS\_LEVEL is not set to BASIC at the system level
- When a cursor is bind aware the bind variable values are used to decide whether it is sensible to share a cursor or not
- Hints: (NO\_)BIND\_AWARE (as of 11.1.0.7 only)

# Adaptive Cursor Sharing – Stability



- Every patchset has fixed some bugs in this area
- Bugs that are fixed in 11.2.0.3
  - 9532657 - Adaptive cursor sharing ignores SELECTs which are not fully fetched
  - 10182051 - Extended cursor sharing generates many shareable child cursors
- Bugs that are not fixed yet
  - 8729064 - Adaptive cursor sharing fails to share
  - 11657468 - Excessive mutex waits with adaptive cursor sharing

# Cardinality Feedback



- As of 11.1 the query optimizer uses actual cardinalities obtained by previous executions to choose an optimal execution plan
  - Provided STATISTICS\_LEVEL is not set to BASIC at the system level
- As a result, every time you rerun a statement you might get a different execution plan!
- DBMS\_XPLAN shows the following note when this technique is used

## Note

-----

- cardinality feedback used for this statement

- Set \_OPTIMIZER\_USE\_FEEDBACK=FALSE to disable the feature
  - Hint: opt\_param('\_optimizer\_use\_feedback','false')

# Core Messages



- The query optimizer is getting better and better
- Let the automatic statistic gathering job running is possible...  
... provided you configure it correctly
- The query optimizer is getting more and more dynamic, i.e. less predictable

# THANK YOU.

Trivadis AG

Christian Antognini

Europa-Strasse 5  
CH-8152 Glattbrugg

Tel. +41 44 808 70 20

info@trivadis.com  
www.trivadis.com

BASEL    BERN    LAUSANNE    ZÜRICH    DÜSSELDORF    FRANKFURT A.M.    FREIBURG I.BR.    HAMBURG    MÜNCHEN    STUTTGART    WIEN