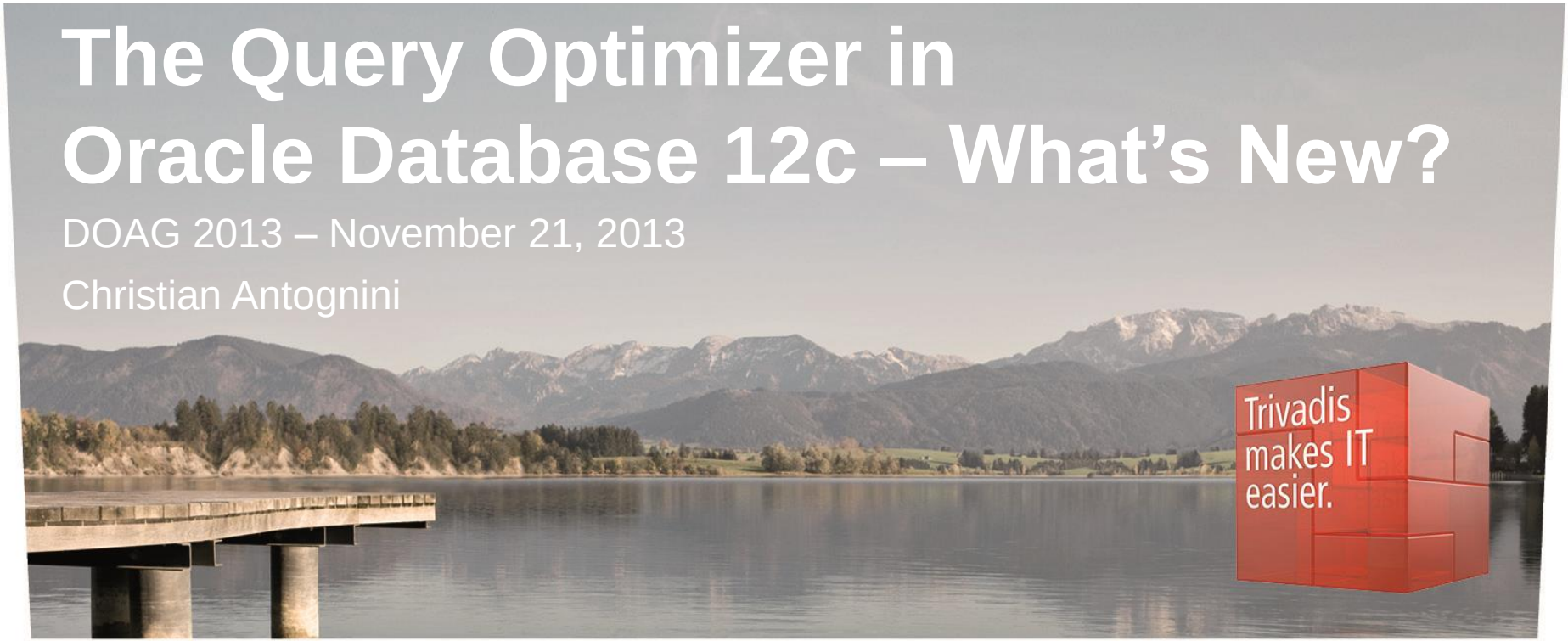
The Query Optimizer in Oracle Database 12c – What's New?

DOAG 2013 – November 21, 2013

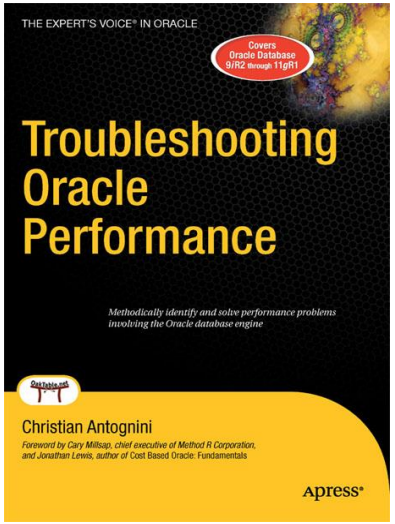
Christian Antognini



BASEL BERN BRUGG LAUSANNE ZUERICH DUESSELDORF FRANKFURT A.M. FREIBURG I.BR. HAMBURG MUNICH STUTTGART VIENNA

@ChrisAntognini

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



trivadis
makes IT easier.

Introduction

- Over the years Oracle has extremely improved the capabilities of the query optimizer
- Most of the improvements fall into one of the following areas
  - Enhance the quality of the inputs (e.g. objects statistics)
  - Make the gathering and management of object statistics easier and faster
  - Implement or enhance query transformations
  - Improve plan stability
  - Cope with poor estimations that leads to poor execution plans
- 12c is no exception, every one of these areas were improved

AGENDA

1. Adaptive Query Optimization
2. Object Statistics
3. Plan Stability
4. Indexing
5. Optimization Techniques
6. Miscellaneous

Adaptive Query Optimization

Adaptive Plans – Challenge

- Object statistics do not always provide sufficient information to find an optimal execution plan
- To get additional insights the query optimizer can take advantage of dynamic sampling
  - Does not solve all issues, though

Adaptive Plans – Concept (1)

- As of 12c, the query optimizer can postpone some decisions until the execution phase
- The idea is to leverage information collected while executing part of an execution plan to determine how another part should be carried out
- The query optimizer uses adaptive plans in two situations:
  - To switch the join method from a nested loops join to a hash join
  - To switch the PX distribution method from hash to broadcast

Adaptive Plans – Concept (2)

- The query optimizer adds *subplans* to execution plans
  - One of the alternatives is the default plan
- A subplan is chosen during the first execution based on the number of rows actually processed
  - The query optimizer computes an inflection point
- A new row source operation is used to partially buffer and count the rows
  - STATISTICS COLLECTOR
- The selection of the final plan is only performed during the first execution
  - V$SQL.IS_RESOLVED_ADAPTIVE_PLAN

Adaptive Plans – Join Method Switch Example

```
SELECT * FROM t1, t2 WHERE t1.id = t2.id AND t1.n = 666

-----------------------------------------------
| Id  | Operation                     | Name  |
-----------------------------------------------
|   0 | SELECT STATEMENT              |       |
|   1 |  HASH JOIN                    |       |
|   2 |   NESTED LOOPS                |       |
|   3 |    NESTED LOOPS               |       |
|   4 |     STATISTICS COLLECTOR      |       |
|   5 |      TABLE ACCESS FULL        | T1    |
|   6 |     INDEX UNIQUE SCAN         | T2_PK |
|   7 |    TABLE ACCESS BY INDEX ROWID| T2    |
|   8 |   TABLE ACCESS FULL           | T2    |
-----------------------------------------------
```

Adaptive Plans – Join Method Switch Example

```
SELECT * FROM t1, t2 WHERE t1.id = t2.id AND t1.n = 666

-----------------------------------------------
| Id  | Operation                     | Name  |
-----------------------------------------------
|   0 | SELECT STATEMENT              |       |
|   1 |  HASH JOIN                    |       |
|   2 |   NESTED LOOPS                |       |
|   3 |    NESTED LOOPS               |       |
|   4 |     STATISTICS COLLECTOR      |       |
|   5 |      TABLE ACCESS FULL        | T1    |
|   6 |     INDEX UNIQUE SCAN         | T2_PK |
|   7 |    TABLE ACCESS BY INDEX ROWID| T2    |
|   8 |   TABLE ACCESS FULL           | T2    |
-----------------------------------------------
```

Adaptive Plans – Join Method Switch Example

```
SELECT * FROM t1, t2 WHERE t1.id = t2.id AND t1.n = 666

-----------------------------------------------
| Id  | Operation                     | Name  |
-----------------------------------------------
|   0 | SELECT STATEMENT              |       |
|   1 |  HASH JOIN                    |       |
|   2 |   NESTED LOOPS                |       |
|   3 |    NESTED LOOPS               |       |
|   4 |     STATISTICS COLLECTOR      |       |
|   5 |      TABLE ACCESS FULL        | T1    |
|   6 |     INDEX UNIQUE SCAN         | T2_PK |
|   7 |    TABLE ACCESS BY INDEX ROWID| T2    |
|   8 |   TABLE ACCESS FULL           | T2    |
-----------------------------------------------
```

Cardinality Feedback

- With cardinality feedback, the query optimizer can take advantage of actual cardinalities obtained by a previous execution
  - Not used for every SQL statement
  - Monitoring is only performed for the first execution of a given child cursor
- Simply put, the feedback has two main consequences:
  - The suboptimal child cursor is invalidated
  - A new child cursor is generated
- When you rerun a statement you might get a different execution plan
- Used only for single-table cardinalities

Statistics Feedback

statistics_feedback1.sql
statistics_feedback2.sql

- Evolution (and new name…) of cardinality feedback
- Used for single-table cardinalities as well as join cardinalities
- Information about misestimates might be persisted through SQL plan directives
  - For misestimates due to table functions no information is stored
- V$SQL.IS_REOPTIMIZABLE

Dynamic Statistics

- Evolution (and new name…) of dynamic sampling
- Used for single-table cardinalities as well as join and group-by cardinalities
- OPTIMIZER_DYNAMIC_SAMPLING has a new level: 11
- At level 11 the query optimizer decides when and how to use dynamic statistics
- Insights resulting from dynamic statistics can be shared and persisted through *SQL plan directives*

SQL Plan Directives (1)

- SQL plan directives are automatically created when misestimates occur
  - They are temporarily stored in the SGA and flush to disk every 15 min

```
SQL> SELECT type, reason, count(*)
  2  FROM dba_sql_plan_directives
  3  GROUP BY type, reason;

TYPE              REASON                                    COUNT(*)
----------------- ---------------------------------------- ----------
DYNAMIC_SAMPLING SINGLE TABLE CARDINALITY MISESTIMATE           81
DYNAMIC_SAMPLING JOIN CARDINALITY MISESTIMATE                  180
DYNAMIC_SAMPLING GROUP BY CARDINALITY MISESTIMATE                6
```

- You can manage them with DBMS_SPD

SQL Plan Directives (2)

- They are not tied to a specific SQL statement, but to a specific column
- They instruct the database engine to automatically create extended statistics
- If creating extended statistics is not supported/possible, they instruct the query optimizer to use dynamic sampling

OPTIMIZER_ADAPTIVE_FEATURES Initialization Parameter

- Enables or disables adaptive query optimization features
  - Adaptive plans
  - Automatic reoptimization (statistics feedback, performance feedback)
    - Not yet! (bug 16824474)
  - SQL plan directives
- The default value is TRUE

OPTIMIZER_ADAPTIVE_REPORTING_ONLY Initialization Parameter

- Enables or disables reporting mode for adaptive query optimization features
- Useful to assess how execution plans would change
- Use DBMS_XPLAN to get detail information about the analysis
  - Might fail with an ORA-1001 (bug 17270605)

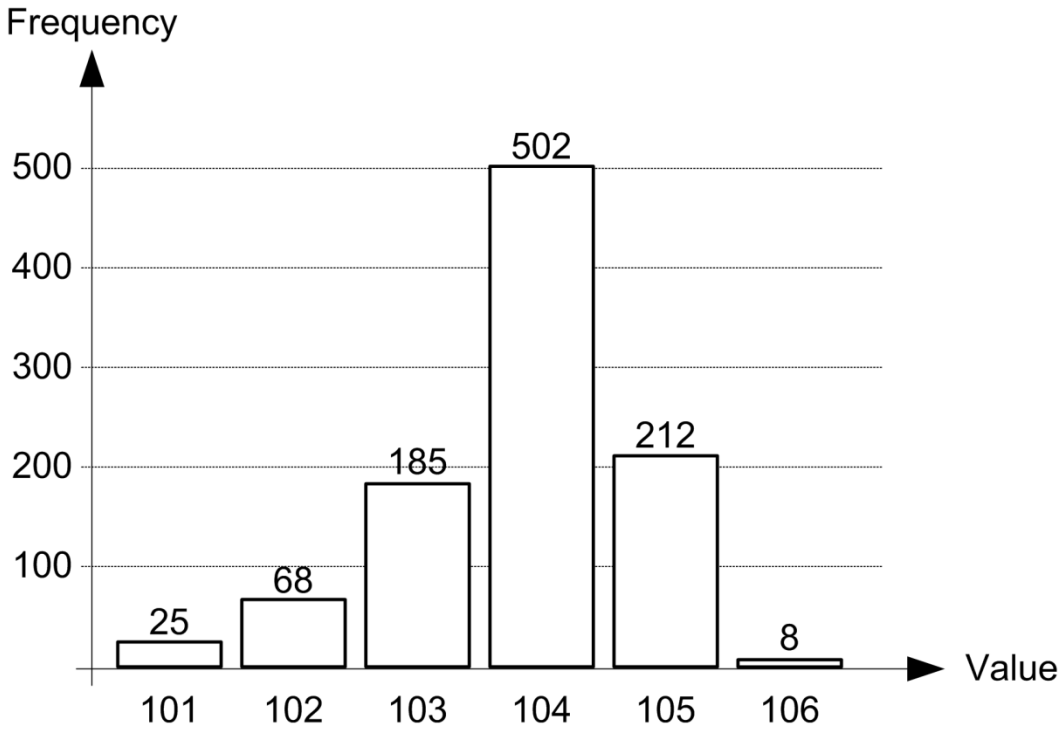
```
SELECT *
FROM table(dbms_xplan.display_cursor(format=>'report'))
```

- The default value is FALSE

Object Statistics

Frequency Histograms

- Precise estimations
- Limited number of distinct values (254)



Height-Balanced Histograms

- Only used when a frequency histogram cannot be build because of the limited number of buckets
- Precision highly dependent on data skewing
  - It is all about popular values (i.e. values associated to several buckets)
- Sometimes misleading
- When histograms are re-gathered they can lead to instability in estimations

Histograms – 12c New Features

- New maximum number of buckets: 2048
- More efficient way to gather histograms (AUTO_SAMPLE_SIZE only)
- New types of histograms
  - Top-frequency histograms
  - Hybrid histograms
- Top frequency histograms and hybrid histograms are supposed to replace height-balanced histograms
- Except if ESTIMATE_PERCENT is set to a an integer value, height-balanced histograms are no longer created

Top-Frequency Histograms

- Similar to frequency histograms, but only the top-n values are stored
- Conditions
  - Number of distinct values larger than *n*
  - Top-n values account for at least *x* percent of the rows
    - x = 100 – 100 / n
  - ESTIMATE_PERCENT must be set to AUTO_SAMPLE_SIZE
- Minimum and maximum values are always part of the histogram
  - If they are *not* among the top-n values, they are added with a frequency of 1
  - Sometimes either the minimum or the maximum is missing, this is a bug!

Hybrid Histograms

- Combination of height-balanced histograms with frequency histograms
- Improvements compared to height-balanced histograms
  - One value is stored in a single bucket
  - Frequency information added to the endpoint values to recognize almost popular values

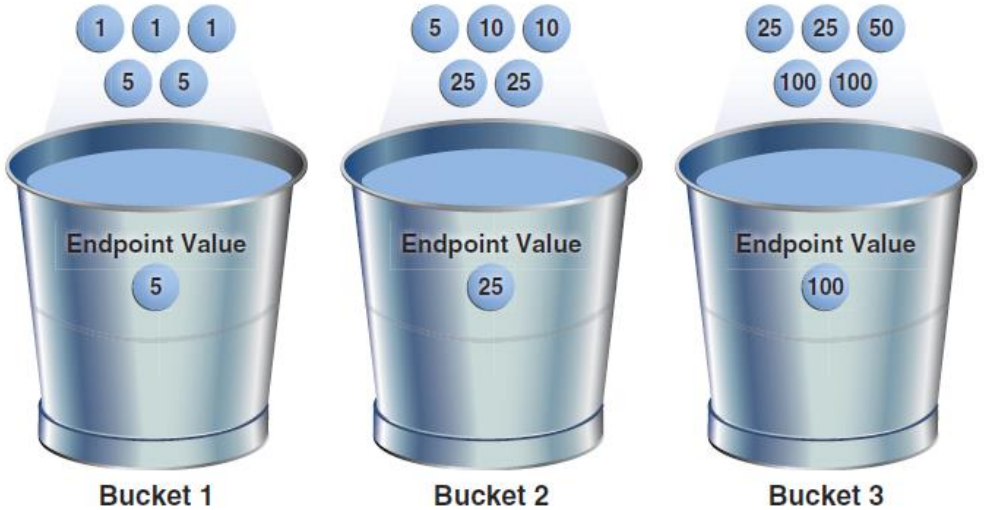
Hybrid Histograms – Example (1)

- Data set to be represented by the hybrid histogram

- METHOD_OPT is set to FOR ALL COLUMNS SIZE 3

Hybrid Histograms – Example (2)

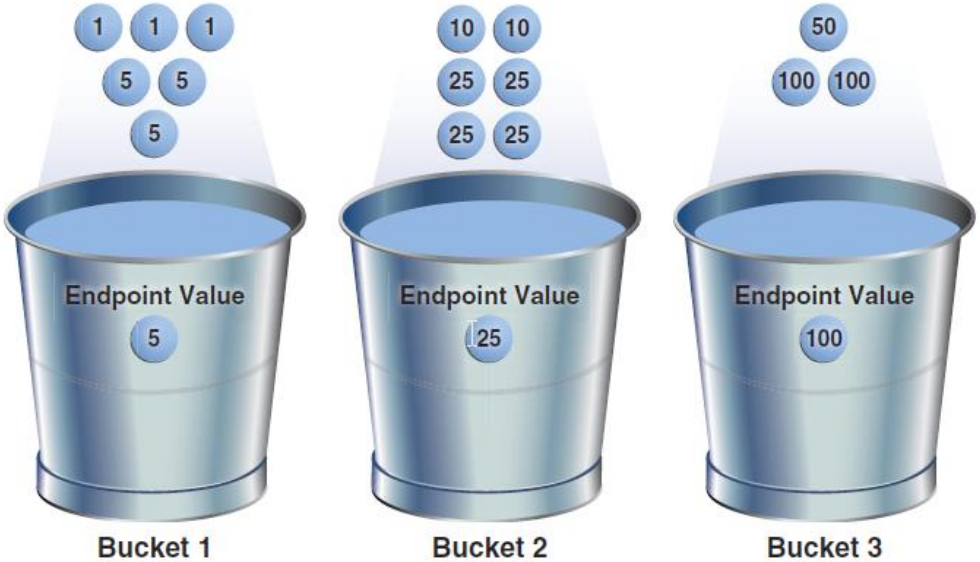
- Step 1: build a height-balanced histogram



Source: Oracle Database 12c – SQL Tuning Guide, Oracle Corporation, Mai 2013

Hybrid Histograms – Example (3)

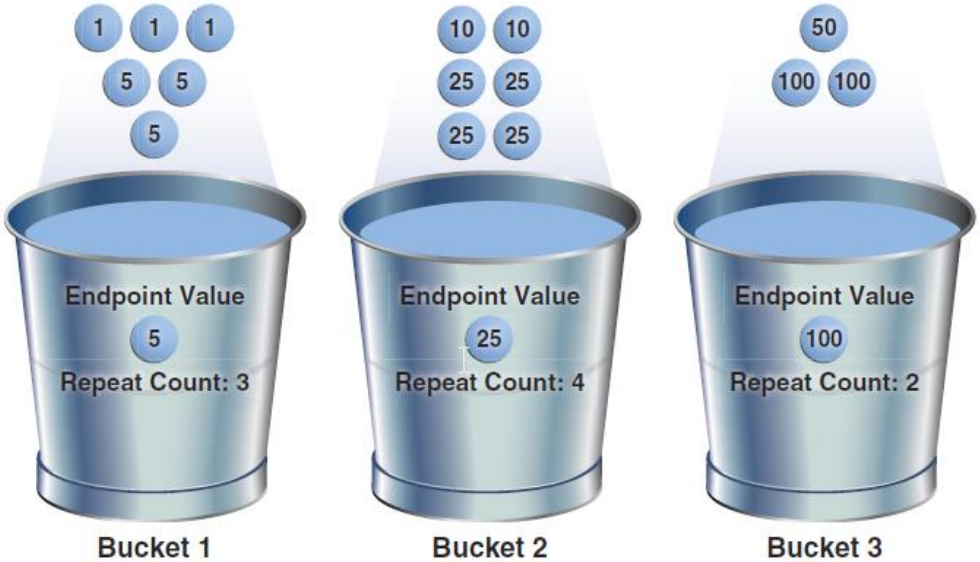
- Step 2: group together endpoint values



Source: Oracle Database 12c – SQL Tuning Guide, Oracle Corporation, Mai 2013

Hybrid Histograms – Example (4)

- Step 3: add frequency of the endpoint values



Source: Oracle Database 12c – SQL Tuning Guide, Oracle Corporation, Mai 2013

Session-Specific Object Statistics for Global Temporary Table

- For GTTs two types of statistics exist
  - SHARED (what we are used to)
    - All sessions use the same statistics
  - SESSION (new in 12c)
    - Every session has its own statistics
    - Every session has to gather its own statistics
- The two types of statistics can coexist
  - Session-specific statistics have precedence over shared statistics
- The feature is controlled by the GLOBAL_TEMP_TABLE_STATS preference
  - The default value is SESSION

Online Object Statistics Gathering for Bulk Load Operations

- Table and column statistics are automatically gathered during the following operations:
  - CTAS statement
  - Direct-path insert into an empty table (except with APPEND_VALUES hint)
- Histograms are not gathered
- Index statistics are gathered only when the index definition is part of the CTAS statement
- Missing index statistics might be a major problem
  - Dynamic sampling does not kick in → Default statistics are used

Enhancements to Incremental Statistics

- Staleness can be controlled through the INCREMENTAL_STALENESS preference
  - USE_STALE_PERCENT (use value of STALE_PERCENTAGE preference)
  - USE_LOCKED_STATS
- Generation of synopsis for non-partitioned tables
  - Set new INCREMENTAL_LEVEL preference to TABLE
  - The default value is PARTITION

DBMS_STATS Reporting Mode

- A new set of functions can be used to list the objects that would be processed during the gathering of object statistics
  - REPORT_GATHER_AUTO_STATS
  - REPORT_GATHER_DATABASE_STATS
  - REPORT_GATHER_DICTIONARY_STATS
  - REPORT_GATHER_FIXED_OBJ_STATS
  - REPORT_GATHER_SCHEMA_STATS
  - REPORT_GATHER_TABLE_STATS

```
SELECT dbms_stats.report_gather_table_stats(…) FROM dual
```

DBMS_STATS Reporting Mode – Parameters

- All parameters supported by the corresponding GATHER_* procedures
- DETAIL_LEVEL (BASIC, TYPICAL, ALL)
- FORMAT (TEXT, HTML, XML)

Logging of Management Operations

- Many DBMS_STATS subroutines log information about their execution
- 12c stores much more information
- New columns in DBA_OPTSTAT_OPERATIONS
  - ID, STATUS, JOB_NAME, SESSION_ID, NOTES
- New view DBA_OPTSTAT_OPERATION_TASKS
- Views for multitenant environment (CDB_*) are also available

Example: Which Parameters Were Used by the Default Gathering Job During the Last 24 Hours?

```
SQL> SELECT x.*
  2  FROM dba_optstat_operations o,
  3       XMLTable('/params/param'
  4                PASSING XMLType(notes)
  5                COLUMNS name VARCHAR2(20) PATH '@name',
  6                        value VARCHAR2(30) PATH '@val') x
  7  WHERE start_time > systimestamp - INTERVAL '1' DAY
  8  AND operation = 'gather_database_stats (auto)';

NAME                 VALUE
-------------------- ------------------------------
block_sample         FALSE
cascade              NULL
...
```

Reports on Management Operations

- Two functions are provided to avoid manual queries against the log
- Example 1: List operations that took place between two timestamps

```
dbms_stats.report_stats_operations(detail_level => 'typical',
                                   format       => 'text',
                                   since        => sysdate-1,
                                   until        => sysdate)
```

- Example 2: List details about one specific operation

```
dbms_stats.report_single_stats_operation(opid         => 2684,
                                         detail_level => 'typical',
                                         format       => 'text')
```

Plan Stability

SQL Plan Management Evolve Advisor

- As of 12c there is a new advisor: SPM Evolve Advisor
  - Task name = SYS_AUTO_SPM_EVOLVE_TASK
- Its purpose is to execute an evolution for the nonaccepted execution plans associated to SQL plan baselines
- It runs during the maintenance window

SQL Plan Management Evolve Advisor – Reporting

- To know what the SPM Evolve Advisor did:
  - Find the name associated to the execution

```
SELECT execution_name, execution_start
FROM dba_advisor_executions
WHERE task_name = 'SYS_AUTO_SPM_EVOLVE_TASK'
ORDER BY execution_start DESC
```

  - Generate a report about the performed activities

```
SELECT dbms_spm.report_auto_evolve_task() FROM dual
```

SQL Plan Management and Unreproducible Execution Plans

- In 11g DBMS_XPLAN.DISPLAY_SQL_PLAN_BASELINE is not always able to correctly display the execution plans associated to a SQL plan baseline
  - The function has to be able to reproduce the execution plans
  - Wrong results or even an error message is raised
- To avoid such a problem, from 12c onward the execution plan is stored in the SQL Management Base
  - For reporting purposes only

Indexing

Multiple Indexes on the Same Columns

- It is *not* possible to create multiple index on the same set of columns

```
SQL> CREATE INDEX i_i ON t (n1);

SQL> CREATE UNIQUE INDEX i_ui ON t (n1);
CREATE UNIQUE INDEX i_ui ON t (n1)
                            *
ERROR at line 1:
ORA-01408: such column list already indexed
```

- A maintenance window is required to change the uniqueness, type, or partitioning of an index

Support for Multiple Indexes on the Same Columns

- Only one of them can be visible at a time
- They have to differ by uniqueness, type, or partitioning

```
SQL> CREATE INDEX i_i ON t (n1);

SQL> CREATE UNIQUE INDEX i_ui ON t (n1) INVISIBLE;

SQL> CREATE BITMAP INDEX i_bi ON t (n1) INVISIBLE;

SQL> CREATE INDEX i_pi ON t (n1) INVISIBLE
  2  GLOBAL PARTITION BY HASH (n1) PARTITIONS 4;
```

Partial Indexes

- For performance purposes it's sometimes *not* necessary to index all data stored in a table
- For example, it might be enough to index data of the last day or week and to leave older data un-indexed
- Through 11.2 some kind of partial indexes are supported by implementing particular tricks
  - Making index partitions unusable
- 12.1 onward provides a specific syntax for partial indexes
  - Supported for partitioned tables only

Partial Indexes for Partitioned Tables

- Turn indexing for (sub)partitions on or off
  - The default value is specified at the table level
  - The default value can be overridden at the (sub)partition level
- The query optimizer takes advantage of the table expansion query transformation to make sure that data is accessed in the optimal way
- Partial indexes are supported for both local and global indexes

Partial Indexes for Partitioned Tables – Example

```
CREATE TABLE t (…)
INDEXING OFF
PARTITION BY RANGE (d) (
  PARTITION t_jan_2013 VALUES LESS THAN (…),
  PARTITION t_feb_2013 VALUES LESS THAN (…),
  PARTITION t_mar_2013 VALUES LESS THAN (…),
  …
  PARTITION t_oct_2013 VALUES LESS THAN (…),
  PARTITION t_nov_2013 VALUES LESS THAN (…),
  PARTITION t_dec_2013 VALUES LESS THAN (…) INDEXING ON
);
```

Optimization Techniques

Execution of UNION and UNION ALL

- Every branch of a UNION and UNION ALL query is processed sequentially
- Only one branch at a time is executed
- Individual branches can be processed in serial or parallel

Concurrent Execution of UNION [ALL]

concurrent_union.sql
concurrent_union_remote.sql

- 12c introduces concurrent execution of branches
  - Branches are executed in parallel and concurrently
- Considered when parallel processing is enabled and considered for all branches
- According to the documentation branches with remote queries are supported
  - I was not able to observe it!
- The concurrent execution is not visible in the execution plan
  - Same execution plan hash value

The (NO_)PQ_CONCURRENT_UNION Hints

- The PQ_CONCURRENT_UNION hint cannot be used to enable concurrent execution if the query optimizer does not consider it
- Both hints do not work if no query block is specified
  - Bug 15851422

Scalar Subquery Unnesting Does Not Processes Correlated Subqueries in the SELECT Clause

```
SELECT t1.*,(SELECT max(t2.id) FROM t2 WHERE t2.id = t1.id)
FROM t1

-------------------------------------
| Id  | Operation           | Name  |
-------------------------------------
|   0 | SELECT STATEMENT    |       |
|   1 |  SORT AGGREGATE     |       |
|*  2 |   INDEX UNIQUE SCAN | T2_PK |
|   3 |  TABLE ACCESS FULL  | T1    |
-------------------------------------

   2 - access("T2"."ID"=:B1)
```

Scalar Subquery Unnesting Processes Correlated Subqueries in the SELECT Clause

```
SELECT t1.*,(SELECT max(t2.id) FROM t2 WHERE t2.id = t1.id)
FROM t1

-------------------------------------
| Id  | Operation           | Name  |
-------------------------------------
|   0 | SELECT STATEMENT    |       |
|   1 |  HASH GROUP BY      |       |
|   2 |   NESTED LOOPS OUTER|       |
|   3 |    TABLE ACCESS FULL| T1    |
|*  4 |    INDEX UNIQUE SCAN| T2_PK |
-------------------------------------

4 - access("T2"."ID"(+)="T1"."ID")
```

Miscellaneous

Expand Text of Queries Referencing Views

```
SQL> BEGIN
  2    :in := q'[SELECT name, owner, category, enabled
  3                FROM dba_outlines
  4                WHERE enabled = 'ENABLED']';
  5    dbms_utility.expand_sql_text(input_sql_text  => :in,
  6                                 output_sql_text => :out);
  7  END;
  8  /

SQL> SELECT :out AS query FROM dual;

QUERY
--------------------------------------------------------------------------------
SELECT "A1"."NAME" "NAME","A1"."OWNER" "OWNER","A1"."CATE...
```

Summary

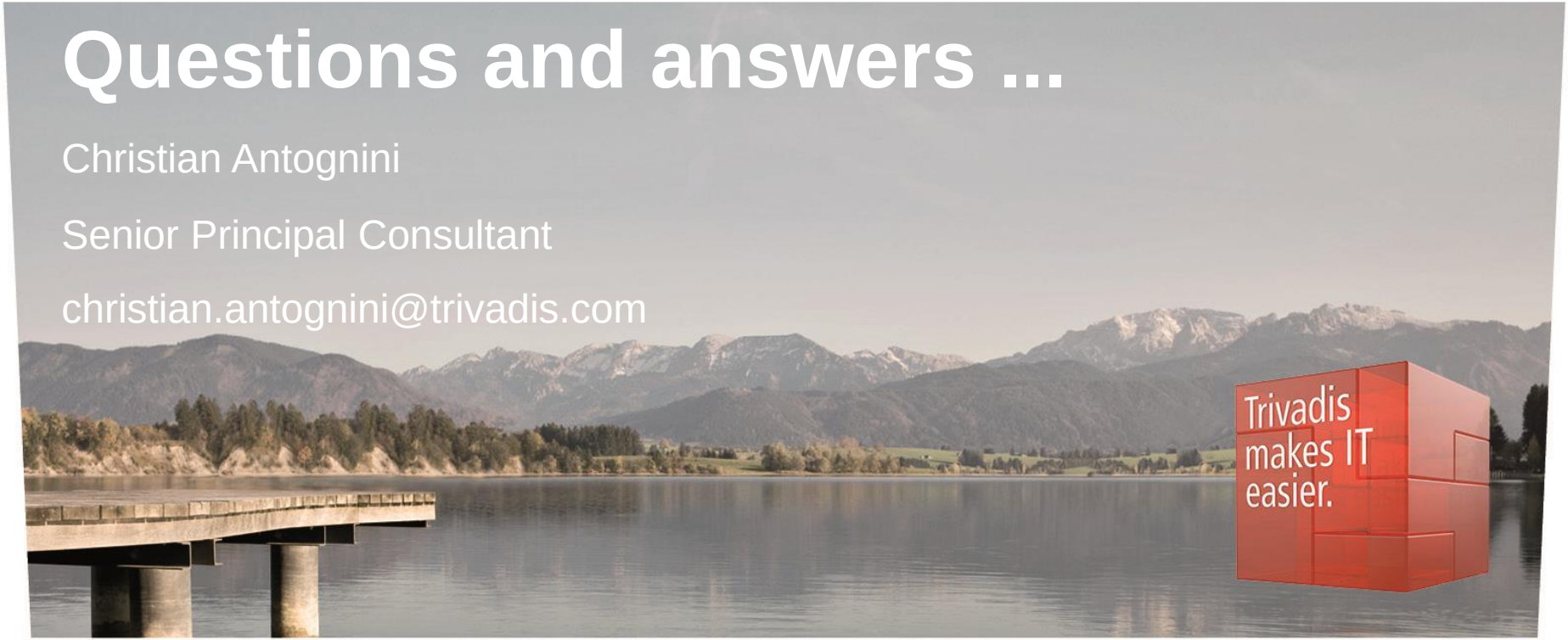
- The query optimizer is getting more and more dynamic
- New types of histogram provide more precise information about data distribution
- No major improvement in SPM
- Partial indexes support, at last

Questions and answers ...

Christian Antognini

Senior Principal Consultant

christian.antognini@trivadis.com



BASEL BERN BRUGG LAUSANNE ZUERICH DUESSELDORF FRANKFURT A.M. FREIBURG I.BR. HAMBURG MUNICH STUTTGART VIENNA