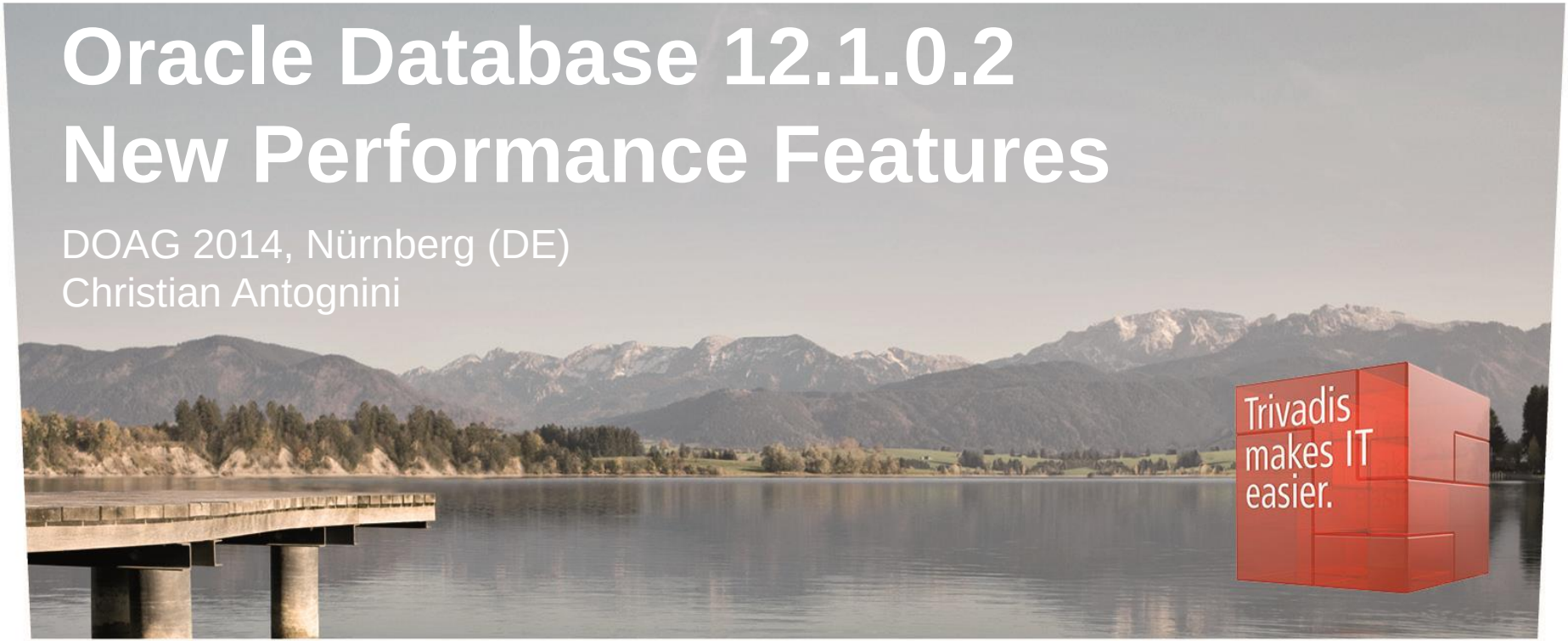
Oracle Database 12.1.0.2 New Performance Features

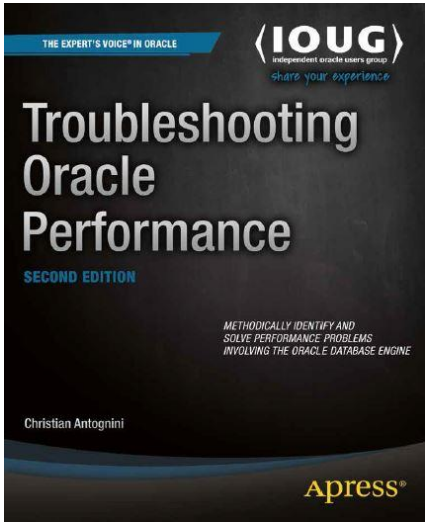
DOAG 2014, Nürnberg (DE)

Christian Antognini

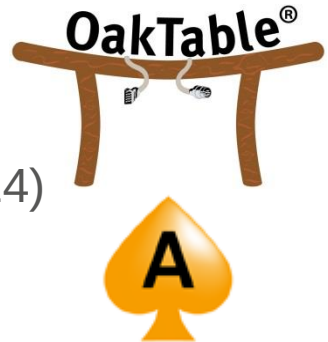


BASEL BERN BRUGG LAUSANNE ZUERICH DUESSELDORF FRANKFURT A.M. FREIBURG I.BR. HAMBURG MUNICH STUTTGART VIENNA

@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, 2008/2014)
- OakTable Network, Oracle ACE Director



AGENDA

1. Zone Maps
2. Attribute Clustering
3. ~~In-Memory Column Store~~
4. In-Memory Aggregation
5. Approximate Count Distinct
6. ~~Automatic Big Table Caching~~
7. ~~Full Database Caching~~

Zone Maps

Zone Maps

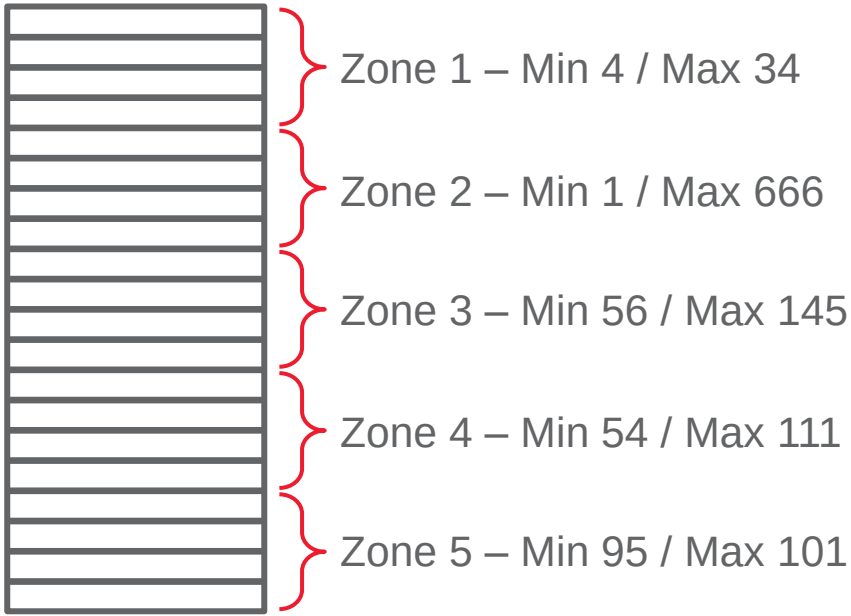
- A zone map is a redundant access structure associated to a table.
  - At most one zone map per table can be created.
- Zone maps are intended to reduce the number of logical/physical I/O during table scans.
  - Zone pruning
  - Partition pruning
- Requirements to use zone maps:
  - Oracle Partitioning option
  - Exadata or Supercluster ☹

How Is a Zone Map Built and What Does It Contain?

- The target table is divided in zones.
  - SYS_OP_ZONE_ID(ROWID,SCALE)
- A zone consists of a specific number of blocks.
  - The SCALE attribute controls it

$$\#blocks = 2^{scale}$$

- For every zone, the zone map stores the minimum and maximum values of several columns.



Basic Zone Map

- A basic zone map stores information about the columns of a single table.

```
CREATE MATERIALIZED ZONEMAP p_bzm ON p (id, n1, n2)
```

- A zone map is a materialized view with some particular properties.

```
SQL> SELECT object_type, object_name
  2  FROM user_objects
  3  WHERE object_name LIKE '%P_BZM';

OBJECT_TYPE        OBJECT_NAME
------------------ -------------
MATERIALIZED VIEW P_BZM
TABLE              P_BZM
INDEX              I_ZMAP$_P_BZM
```

Basic Zone Maps – Execution Plan

```
SELECT count(*) FROM p WHERE n1 = 42

-----------------------------------------------------------
| Id  | Operation                                 | Name  |
-----------------------------------------------------------
|   0 | SELECT STATEMENT                          |       |
|   1 |  SORT AGGREGATE                           |       |
|*  2 |   TABLE ACCESS STORAGE FULL WITH ZONEMAP  | P     |
-----------------------------------------------------------

2 - storage("N1"=42)
    filter((SYS_ZMAP_FILTER('/* ZM_PRUNING */ SELECT "ZONE_ID$",
           CASE WHEN BITAND(zm."ZONE_STATE$",1)=1 THEN 1 ELSE CASE
           WHEN (zm."MIN_2_N1" > :1 OR zm."MAX_2_N1" < :2) THEN 3
           ELSE 2 END END FROM "CHRIS"."P_ZM" zm
           WHERE zm."ZONE_LEVEL$"=0 ORDER BY zm."ZONE_ID$"',
           SYS_OP_ZONE_ID(ROWID),42,42)<3 AND "N1"=42))
```

Join Zone Maps

- A join zone map stores information about the columns of several tables.

```
CREATE MATERIALIZED ZONEMAP p_jzm AS
SELECT sys_op_zone_id(p.rowid) AS zone_id$,
       min(p.n1) AS min_p_n1, max(p.n1) AS max_p_n1,
       min(p.n2) AS min_p_n2, max(p.n2) AS max_p_n2,
       min(c.n1) AS min_c_n1, max(c.n1) AS max_c_n1,
       min(c.n2) AS min_c_n2, max(c.n2) AS max_c_n2
FROM p LEFT OUTER JOIN c ON p.id = c.p_id
GROUP BY sys_op_zone_id(p.rowid)
```

Join Zone Maps – Execution Plan

```
SELECT count(*) FROM p JOIN c ON p.id = c.p_id WHERE c.n2 = 42

-----------------------------------------------------------
| Id  | Operation                                 | Name |
-----------------------------------------------------------
|   0 | SELECT STATEMENT                          |      |
|   1 |  SORT AGGREGATE                           |      |
|*  2 |   HASH JOIN                               |      |
|*  3 |    TABLE ACCESS STORAGE FULL              | C    |
|*  4 |    TABLE ACCESS STORAGE FULL WITH ZONEMAP | P    |
-----------------------------------------------------------

2 - access("P"."ID"="C"."P_ID")
3 - storage("C"."N2"=42)
    filter("C"."N2"=42)
4 - filter(SYS_ZMAP_FILTER('/* ZM_PRUNING */ SELECT "ZONE_ID$",
           CASE WHEN BITAND(zm."ZONE_STATE$",1)=1 THEN 1 ELSE … )
```

Staleness of Zone Maps

- Zone maps are redundant access structures that are not always up-to-date.
- Basic zone maps:
  - DML → selective invalidation
  - Direct-path insert → automatic maintenance
- Join zone maps (created on the parent table):
  - DML on parent → selective invalidation
  - Direct-path insert on parent → automatic maintenance
  - DML or direct-path insert on child → full invalidation

Refreshing Zone Maps

- To refresh a zone map two techniques are available:
  - ALTER MATERIALIZED ZONEMAP REBUILD statement
  - REFRESH procedure of DBMS_MVIEW package
- Complete as well as fast refreshes are available.

Attribute Clustering

Clustering Data

- Clustering data that is processed together is an old optimization technique.
  - Index and hash clusters
  - Heap table reorganization for improving the clustering factor of an important index
- Clustered data makes a number of features more effective
  - B-tree index range scans
  - Compression
  - Exadata/In-Memory storage indexes
  - Zone maps pruning

Attribute Clustering

- Attribute clustering is a new table directive that specifies
  - on which column(s) data has to be clustered
  - how data is ordered (linear, interleaved)
  - when clustering takes place

```
ALTER TABLE t CLUSTERING BY LINEAR ORDER (id) YES ON LOAD
```

- Keeping data clustered may be expensive. Therefore, the database engine doesn't enforce it.
  - It only takes place in case of direct-path inserts and data movement operations.

Join Attribute Clustering

- It's also possible to cluster data based on column(s) of another table
  - The joined table must have a PK or UK (ORA-65418)

```
CREATE TABLE p (id NUMBER,
                n NUMBER,
                pad VARCHAR2(100),
                CONSTRAINT p_pk PRIMARY KEY (id))
```

```
CREATE TABLE c (id NUMBER,
                p_id NUMBER,
                p_n NUMBER,
                pad VARCHAR2(100))
CLUSTERING c JOIN p ON (c.p_id = p.id) BY LINEAR ORDER (p.n)
```

Attribute Clustering and Zone Maps

- It is possible to combine attribute clustering with zone maps
- Both clustering and the zone map are based on the same columns

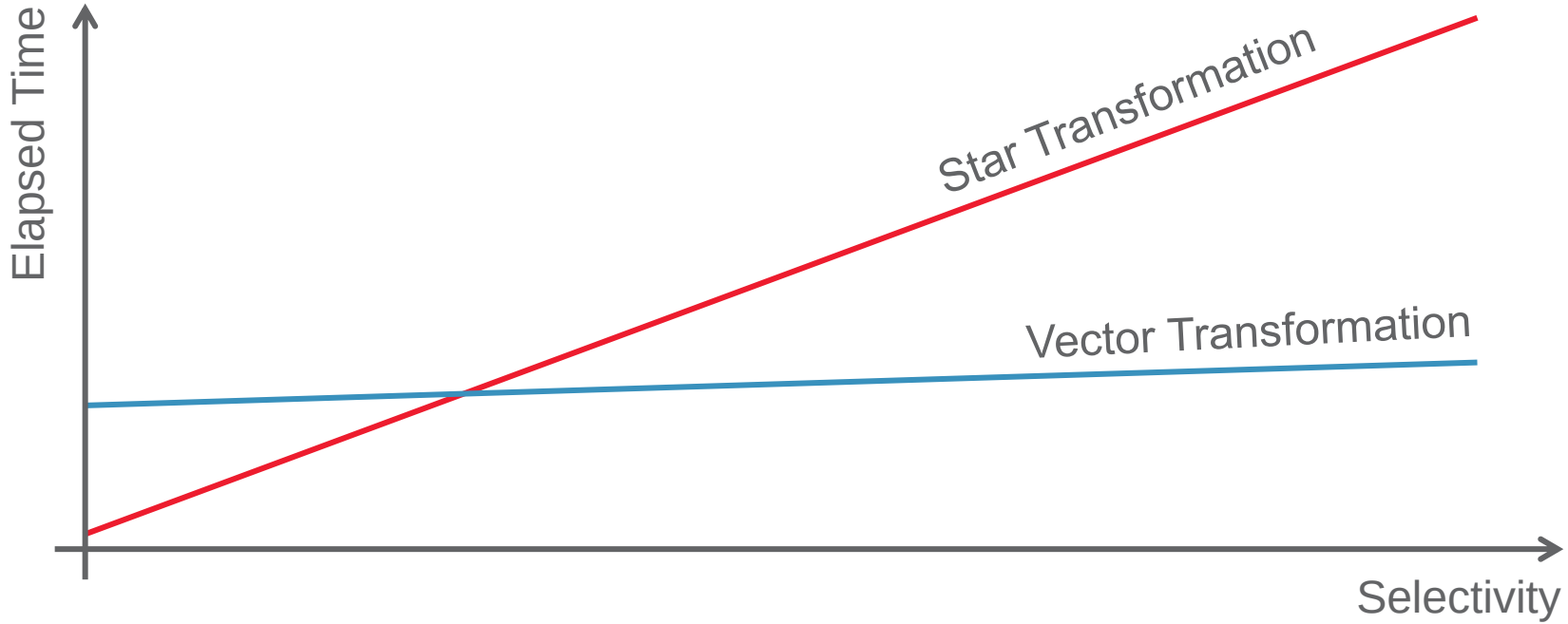
```
CREATE TABLE c (
  id NUMBER,
  p_id NUMBER,
  p_n NUMBER,
  pad VARCHAR2(100)
)
CLUSTERING c JOIN p ON (c.p_id = p.id) BY LINEAR ORDER (p.n)
WITH MATERIALIZED ZONEMAP (c_zm)
```

In-Memory Aggregation

The Star Schema Challenge

- To optimize queries against a star schema, the query optimizer should do the following:
  - Start evaluating each dimension table that has restrictions on it.
  - Assemble a list with the resulting dimension keys.
  - Use this list to extract the matching rows from the fact table.
- This approach cannot be implemented with regular joins.
  - The query optimizer can join only two data sets at one time.
  - Joining two dimension tables leads to a Cartesian product.
- To solve this problem, Oracle Database implements two query transformations:
  - Star transformation
  - Vector transformation (new in 12.1.0.2 – requires the In-Memory option)

Star Transformation vs. Vector Transformation



Vector Transformation (1)

- In the documentation Oracle refers to it as *In-Memory Aggregation*
  - It isn't only about *aggregation*
- It requires the In-Memory option because it takes advantage of some of its features
  - INMEMORY_SIZE must be greater than 0
- The query optimizer considers it when
  - equijoins are used
  - an aggregate function is applied to a column of the fact table
  - the query contains a GROUP BY clause
    - ROLLUP, CUBE and GROUPING SETS are not yet supported

Vector Transformation (2)

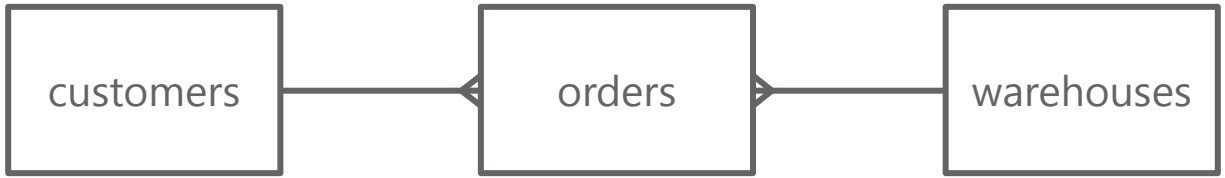
- It's a cost-based query transformation
- It can be controlled through the (NO_)VECTOR_TRANSFORM hints
  - As with the star transformation, sometimes is not possible to force it
- It introduces new row source operations
  - KEY VECTOR CREATE
  - KEY VECTOR USE
  - VECTOR GROUP BY

Vector Transformation – Processing Steps

1. For every dimension table with a filter on it the following operations take place
   - Access table and filter data
   - Create key vector
   - Aggregate data
   - Create temporary table
2. Access the fact table through a FTS and filter data by applying the key vectors
   - On Exadata the filter is not yet offloaded
3. Aggregate data with either a vector or hash aggregation
4. Join data from the fact table to temporary tables
5. Join dimension tables without a filter on them

Example

```
SELECT c.customer_class, sum(o.order_total)
FROM orders o, customers c, warehouses w
WHERE o.customer_id = c.customer_id
AND o.warehouse_id = w.warehouse_id
AND c.dob BETWEEN :b1 AND :b2
AND w.warehouse_name BETWEEN :b3 AND :b4
GROUP BY c.customer_class
```



Execution Plan – Access Dimension Tables, Create Key Vectors, and Create Temporary Tables

```
-------------------------------------------------------------------------------
| Id  | Operation                                | Name                       |
-------------------------------------------------------------------------------
|   0 | SELECT STATEMENT                         |                            |
|   1 |  TEMP TABLE TRANSFORMATION               |                            |
|   2 |   LOAD AS SELECT                         | SYS_TEMP_0FD9DE69C_28565764 |
|   3 |    VECTOR GROUP BY                       |                            |
|   4 |     KEY VECTOR CREATE BUFFERED           | :KV0000                    |
|*  5 |      TABLE ACCESS STORAGE FULL           | CUSTOMERS                  |
|   6 |   LOAD AS SELECT                         | SYS_TEMP_0FD9DE69D_28565764 |
|   7 |    VECTOR GROUP BY                       |                            |
|   8 |     HASH GROUP BY                        |                            |
|   9 |      KEY VECTOR CREATE BUFFERED          | :KV0001                    |
|* 10 |       TABLE ACCESS STORAGE FULL          | WAREHOUSES                 |
...
```

Execution Plan – Access Fact Table by Applying Key Vectors, and Join Temporary Tables

```
...
|  11 |   HASH GROUP BY                      |                                |
|* 12 |    HASH JOIN                         |                                |
|  13 |     MERGE JOIN CARTESIAN             |                                |
|  14 |      TABLE ACCESS STORAGE FULL       | SYS_TEMP_0FD9DE69D_28565764    |
|  15 |      BUFFER SORT                     |                                |
|  16 |       TABLE ACCESS STORAGE FULL      | SYS_TEMP_0FD9DE69C_28565764    |
|  17 |     VIEW                             | VW_VT_72AE2D8F                 |
|  18 |      VECTOR GROUP BY                 |                                |
|  19 |       HASH GROUP BY                  |                                |
|  20 |        KEY VECTOR USE                | :KV0001                        |
|  21 |         KEY VECTOR USE               | :KV0000                        |
|* 22 |          TABLE ACCESS STORAGE FULL   | ORDERS                         |
-------------------------------------------------------------------------------

  ...
  22 - filter(SYS_OP_KEY_VECTOR_FILTER("O"."CUSTOMER_ID",:KV0000) AND
              SYS_OP_KEY_VECTOR_FILTER("O"."WAREHOUSE_ID",:KV0001))
```

Approximate Count Distinct

APPROX_COUNT_DISTINCT Function

- Computing the number of distinct values can be resource intensive
- In some situation performance is more important than precision
- Many algorithms that estimate the number of distinct values have been developed
- Oracle implemented one of this algorithms (HyperLogLog)
- In case an estimated number of distinct values is acceptable, you can replace COUNT(DISTINCT *expr*) with APPROX_COUNT_DISTINCT(*expr*)

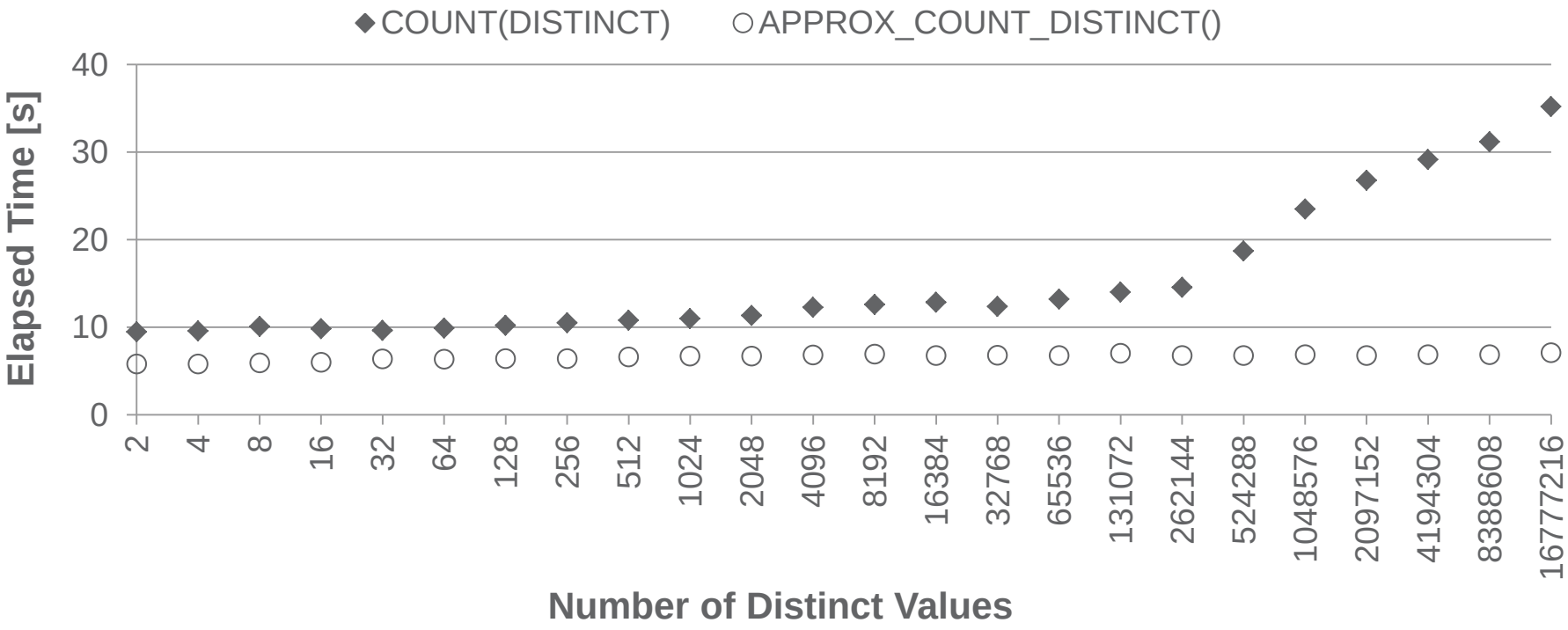
Test Case – Setup

- 100’000’000 rows
- 12.7 GB
- 24 columns
  - Number of distinct values is 2^m, where $\{m \in \mathbb{N}^* | m < 25\}$
- Test queries

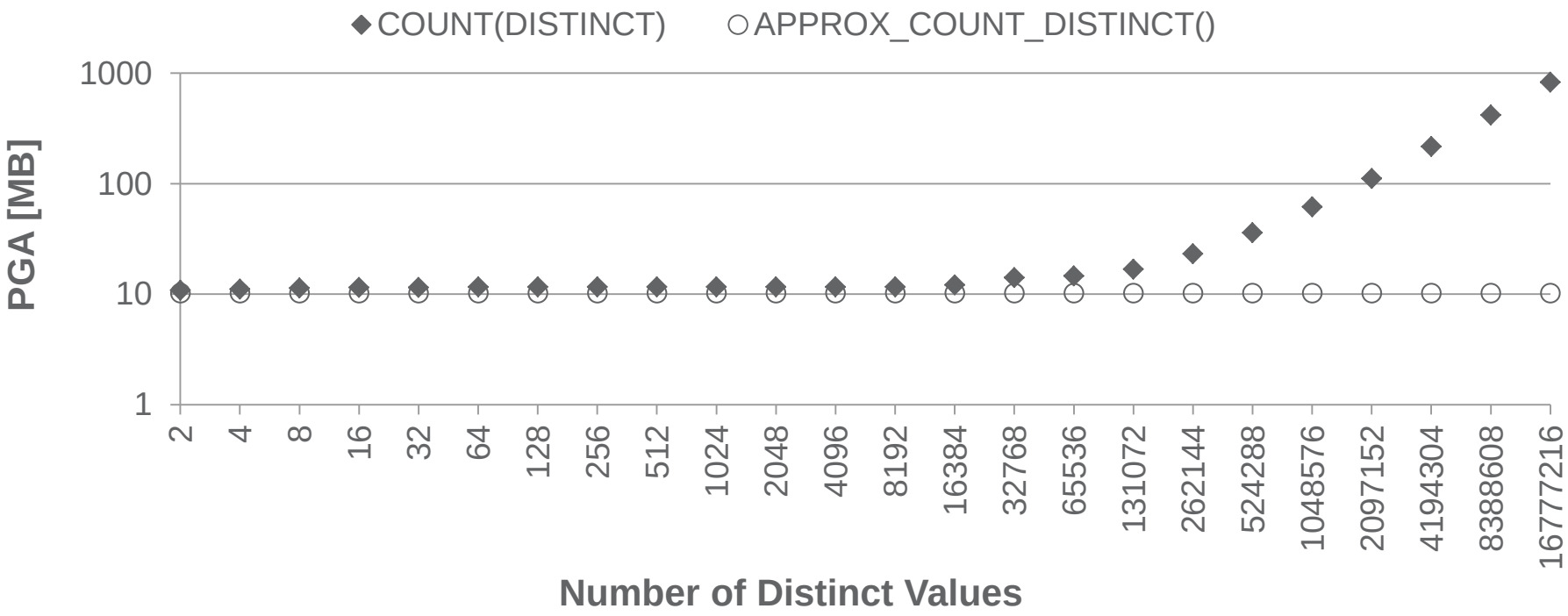
```
SELECT count(DISTINCT n_m) FROM t
```

```
SELECT approx_count_distinct(n_m) FROM t
```

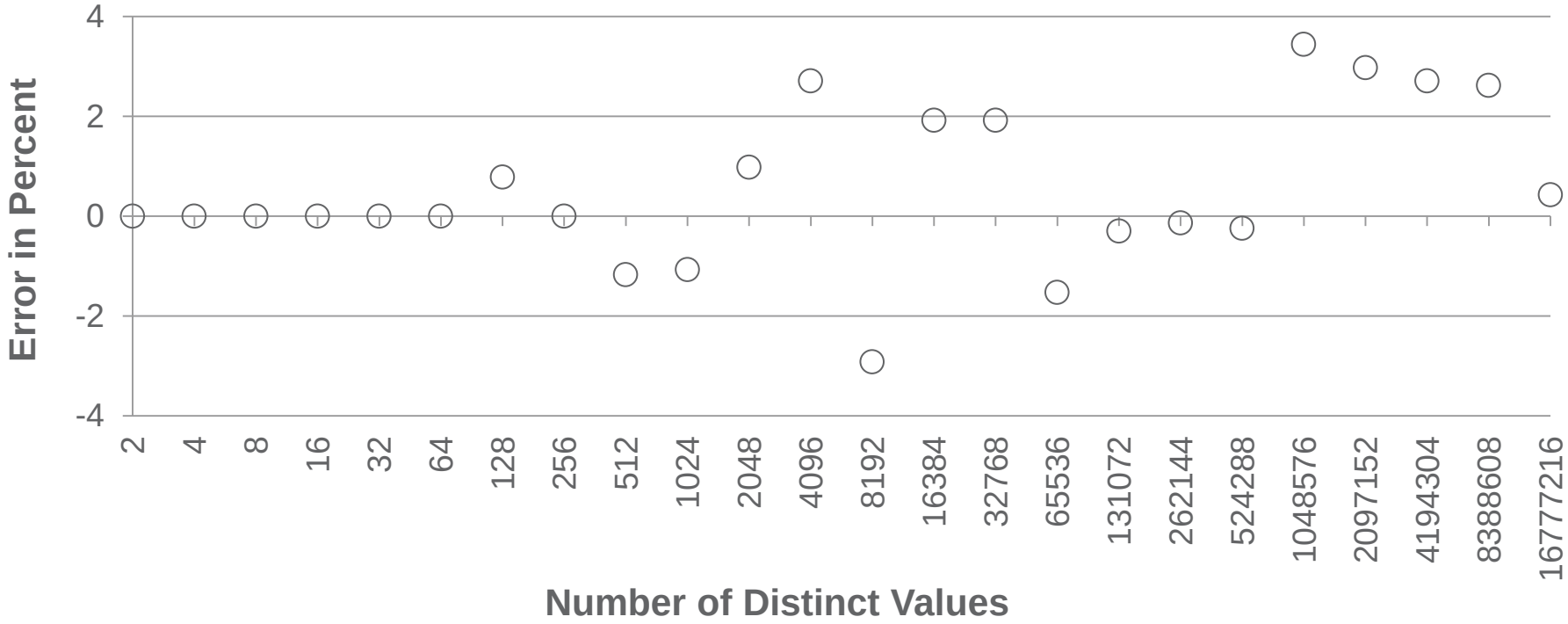
Test Case – Elapsed Time



Test Case – PGA Utilization



Test Case – Precision



Core Messages

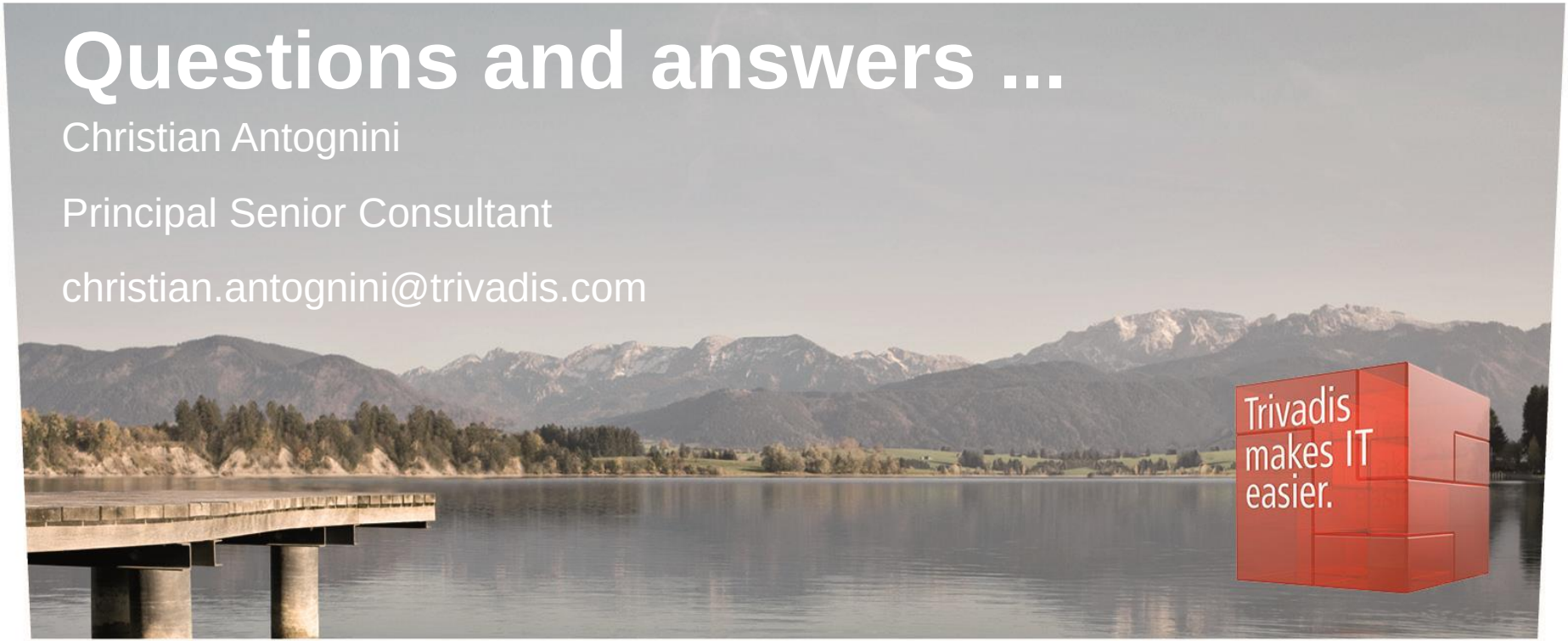
- Zone Maps
  - Interesting concept, but mostly useless because of licensing rule
- Attribute Clustering
  - Declarative way to implement old but good technique for “load-once read-many” data
- In-Memory Aggregation
  - Very interesting concept, complements the star transformation
- Approximate Count Distinct
  - Useful feature, sound implementation

Questions and answers ...

Christian Antognini

Principal Senior Consultant

christian.antognini@trivadis.com



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