Performance Study of Analytical Queries of Oracle and Vertica

Conference Paper · May 2013

DOI: 10.13140/2.1.3667.0726

2 authors:

Hristo Kyurkchiev
Sofia University "St. Kliment Ohridski"
7 PUBLICATIONS **17** CITATIONS
SEE PROFILE

Kalinka Kaloyanova
Sofia University "St. Kliment Ohridski"
69 PUBLICATIONS **115** CITATIONS
SEE PROFILE

Performance Study of Analytical Queries of Oracle and Vertica

Hristo Kyurkchiev[1], Kalinka Kaloyanova[1]

[1]Faculty of Mathematics and Informatics, Sofia University, 5 James Bourchier blvd., 1164, Sofia, Bulgaria

{hkyurkchiev, kkaloyanova}@fmi.uni-sofia.bg

Abstract. Analyzing and mining transactional data straight from the DBMS has become increasingly more popular, provoking research in read optimization for RDBMS. One branch of such research – column orientation claims significant improvement in read performance in comparison with row oriented DBMS. Having previously looked into the strengths and weaknesses of both approaches we take on studying the performance of commercial grade DBMSs, which employ the two views. The first step in this is examining their data models. We then develop a benchmark, which is subsequently used to measure each DBMS's performance. Evaluating the results we draw conclusions about each DBMS's suitability and main advantages over the other.

Keywords: Relational databases, Database systems, Data warehouses, Column stores, Performance evaluation, Analytical query load, Oracle, C-Store, Vertica

1 Introduction

Codd's relational database model [3] has dominated the database world for the last couple of decades due to its atomicity, consistency, isolation, and durability properties and ease of use even for non-IT specialists [4]. It is used by most traditional database management systems (DBMS) not only as a logical data model, but also as a physical one. As a result, the data tuples in them are stored contiguously on the disk [13]. This approach is usually denoted as the N-ary storage model (NSM). Column-stores also use the relational model as a logical data model. They, however, use the decomposed storage model (DSM) for physical data storage [1]. Having previously examined both ways of treating relational data [7] in general, we look into concrete, commercial grade DBMSs e.g. Oracle and Vertica to see how these abstract models affect the performance in practice. We start by first analyzing the data models and performance optimizations of both systems. Next we develop a performance benchmark, which we then use to quantify the performance differences between employing row, or column orientation. Based on these results we draw conclusions about each DBMS's suitability for different setups and query loads.

2 Architectural overview

Both Oracle and Vertica use as a logical model the relational data model [3]. There are certain architectural differences between the two DBMSs, however, which make them perform differently in certain conditions. In order to find why that is so and what these performance differences are with regard to analytical queries we start by examining the architectures of both DBMSs.

2.1 Oracle's architecture

Physical database structure. We use Oracle 11g to gain more information about Oracle's architecture, as this is the version later used in the tests. As every other Oracle DBMS it uses the relational model as the logical representation of the values in the database, although it also features extensions for object-oriented modeling [9]. Each implementation of a relational DBMS (RDBMS), however, specifies how the data is physically stored on the hardware. In Oracle's case the datafiles are stored in tablespaces, with each datafile being in only one tablespaces and each tablespace containing multiple datafiles [5, 9]. Other important structural components of an Oracle database include control files, redo log files, archived logs, block change tracking files, Flashback logs, and recovery backup (RMAN) files [5]. Essentially the datafiles, control files and redo log files physically represent the database on the disk [5].

For us the most important in structural regard is the datafile as it contains the real data [5, 9]. It is composed of Oracle database blocks (between 2KB and 32KB), which are in turn composed of operating system blocks [5]. The datafiles also have a logical organization of three levels – data blocks, extends and segments [5]. Bellow the structure of a typical datafile is shown. It is important to note that in the datafile header there is a checkpoint structure, which is in the form of a timestamp and is used to determine when the last changes to the file were written.

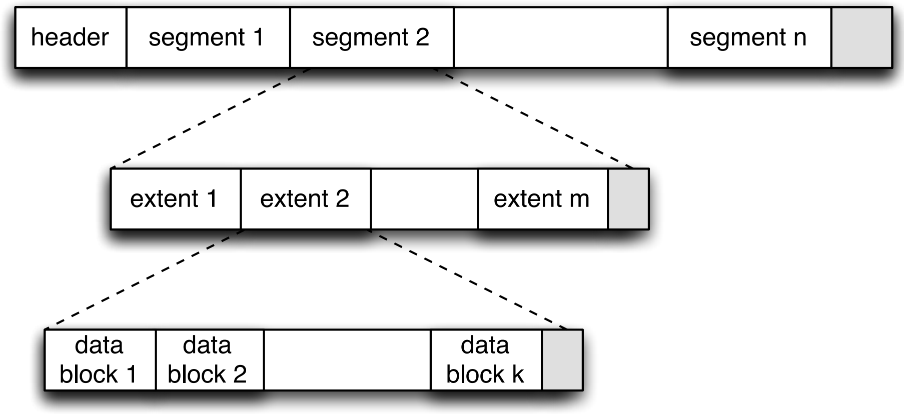

Fig. 1 Structure of a datafile

Each table has its own segment and the data in the table is stored in the segment's extents [9]. Thus the tables are stored in segments and their rows are separated between the extents of a segment. It is worth noting that the rows are usually stored with each column in the order, in which they are defined by the CREATE TABLE SQL statement [14].

Read optimization. Oracle provides a set of features to optimize read performance - query optimizer, indices, parallelization, materialized views, etc.

The key database component that enables Oracle's customers to achieve better performance is Oracle Query optimizer, which consists of four major elements [12]:

- SQL transformations, based mostly on heuristic and cost-based query transformations;
- Execution plan selection (different joint methods, indices, parallel execution);
- Cost model and statistics (object level, system, user defined, etc.);
- Dynamic runtime optimization focused on effective management of the hardware resources, especially memory and CPU.

Oracle uses two main index architectures - b-Tree indices and bitmap indices and some their variations like cluster indices, bitmap join indices, function-based indices, reverse key indices, text indices, etc.

2.2 Vertica's architecture

Vertica is essentially the commercialization of C-Store [8]. Thus it resembles the original column-oriented database greatly. Since in prior research [7] the main architectural traits of C-Store were outlined, and since Vertica resembles C-Store in most of them, only the differences are analyzed, while the other characteristics are just briefly mentioned.

Logical database structure. Like C-Store, Vertica stores the relations by first decomposing them into **projections**. It also supports pre-joined projections to be defined, just like C-Store. However, it is a requirement that every table should have at least one *super projection* associated with it [8]. This means easier storage and lifts the necessity to implement another of C-Store's features – the **join-indices.**

Compression is another of C-Store's architectural characteristics, which made its way to Vertica. Vertica implements a slightly different list of compression methods, however, which can be found bellow [8]:

- *Auto:* the system automatically selects the compression type;
- *RLE:* the system replaces all occurrences of a single value with one pair *<value, # of occurrences>*;
- *Delta Value:* the system represents each value as a difference from the smallest value in the data block;
- *Block Dictionary:* the system replaces each value with an index in a dictionary built based on all of the distinct column values in the data block;

- *Compressed Delta Range:* the system replaces each value as a difference from the value that precedes it in the column.
- *Compressed Common Delta:* the systems stores indices in a dictionary built based on all the deltas in the data block using entropy coding.

Partitioning and **segmentation** are also supported in Vertica, as in C-Store. Both notions have evolved further with Vertica supporting intra-note and inter-node partitioning [8].

Physical database structure. Since Vertica, like C-Store consists of a Read Optimized Store (ROS) and a Write Optimized Store (WOS) there are two different ways to store the data, as each store has its own type of container [8]:

- *ROS container:* it contains a number of complete tuples sorted by the projection's sort order and stored as a pair of files per column. Each column is stored in two files – one with the actual column data and one with a position index. The position index represents important data that can speed up performance, e.g. minimum and maximum value of the column, information that can improve tuple reconstruction speed, etc. The data within the ROS container is identified by its position within it, e.g. its ordinal position within the file.
- *WOS container:* the data in these containers is solely in memory, where its orientation (row or column) has little importance. Thus it is of little use for this study.

Read optimization. Prior research of C-Store has shown that the following features are the main reasons for its performance benefits [7] and since Vertica employees all of them it is only natural to conclude that they play a significant part of its read optimization.

- Separate read-optimized and writable store;
- Projections;
- Compression and data encoding;
- Block iteration;
- Late materialization strategies;
- Invisible joins.

To these we can also add [15]:

- Column-orientation;
- Ability to exploit multiple sort orders;
- Parallel shared-nothing design on off-the-shelf hardware.

3 Experiment setup and results

In this section the main aspects of the experiment such as the hardware and software, the database schema, the benchmark, and the measuring tools are discussed. Only a part of the original database schema is presented, as it is proprietary information.

3.1 Setup description

Hardware setup. All performance benchmarks have been done on a server with 4 Intel Xeon processors each with 4 cores each clocked at 2.4 GHz, 128 GB of RAM and 4 TB of hard disk storage. The DBMSs themselves were run as virtual machines on top of VMWare ESXi software. Each virtual machine was equipped with 2 CPU cores, 6 GB of RAM and 16 GB of storage.

Database setup. No specific tweaks have been performed on each database to improve performance, except the standard ones:

- Oracle 11g SE One was used as the Oracle instance with primary key constraints on all of the surrogate keys (which means implicit indices), as well as indices on all of the foreign keys.
- HP Vertica Community Edition was used as the Vertica instance (available as a free download from HP) with super projections on each table.

Database schema. The same database schema and data were imported into both DBMSs. They are derived from a real production configuration. The schema below features only a subset of the original columns, partly because of confidentiality and partly because of space requirements as the original schema includes much more columns (e.g. mainly for the *Trips* table, which has around 500 columns).

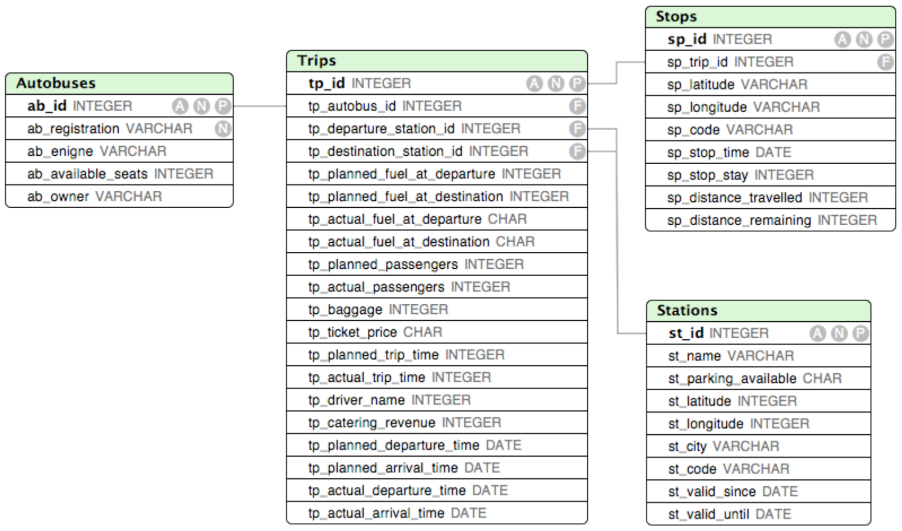
Autobuses
ab_id INTEGER
ab_registration VARCHAR
ab_enigne VARCHAR
ab_available_seats INTEGER
ab_owner VARCHAR
Trips
tp_id INTEGER
tp_autobus_id INTEGER
tp_departure_station_id INTEGER
tp_destination_station_id INTEGER
tp_planned_fuel_at_departure INTEGER
tp_planned_fuel_at_destination INTEGER
tp_actual_fuel_at_departure CHAR
tp_actual_fuel_at_destination CHAR
tp_planned_passengers INTEGER
tp_actual_passengers INTEGER
tp_baggage INTEGER
tp_ticket_price CHAR
tp_planned_trip_time INTEGER
tp_actual_trip_time INTEGER
tp_driver_name INTEGER
tp_catering_revenue INTEGER
tp_planned_departure_time DATE
tp_planned_arrival_time DATE
tp_actual_departure_time DATE
tp_actual_arrival_time DATE
Stops
sp_id INTEGER
sp_trip_id INTEGER
sp_latitude VARCHAR
sp_longitude VARCHAR
sp_code VARCHAR
sp_stop_time DATE
sp_stop_stay INTEGER
sp_distance_travelled INTEGER
sp_distance_remaining INTEGER
Stations
st_id INTEGER
st_name VARCHAR
st_parking_available CHAR
st_latitude INTEGER
st_longitude INTEGER
st_city VARCHAR
st_code VARCHAR
st_valid_since DATE
st_valid_until DATE

Fig. 2 Sample database schema

The data concerns autobus trips from a travelling agency. It includes autobus data (~ 50 records), station data (~ 5000 records), trips data (~ 100000 records), and stops data (~ 1750000 records).

3.2 Benchmark

Benchmark overview. Since most performance studies of C-Store [2, 6, 13] use the TPC-H benchmark or the commissioned by Vertica Star Schema Benchmark (SSB), which is derived from TPC-H [10], we decided to model our own benchmark. Another reason for doing this is that the schema that we use for testing purposes greatly differs in its properties (e.g. number of records, interconnectivity, etc.) from the ones used in the above-mentioned benchmarks. Modeling our own benchmark also allows us to include queries that are of different nature than the ones in SSB and TPC-H benchmarks.

We structured the benchmark into two sets of flights of queries:

- General queries – one flight of queries, which measures the performance of simple queries such as single row inserts, updates, deletes, and querying data for non-analytical purposes;
- Analytical queries – four flights of queries, which measure the performance of different analytical types of queries against the above schema.

Flights of queries. As already mentioned there are four flights of queries. Most of them are inspired by the already mentioned SSB [11], except the first flight of queries, however, is completely new and not a part of either SSB or TCP-H benchmarks. All of them are described below.

The purpose of the ***first flight*** of queries is to test the common operations in transactional DBMSs, e.g. insert, update, etc. It includes:

- Q1 simple inserts:
  insert into autobuses values('ABCD', 'XY12ZT3', 45, 'S');
- Q2 simple updates:
  update autobuses set registration = 'XYZT' where id = 1;
- Q3 simple deletes:
  delete from autobuses where id = 1;
- Q4 simple selects:
  *select * from autobuses [where id = 1];*

The flight essentially consists of twenty queries[*], each of the queries Q1 to Q4 run four times – one for each of the schema's tables, so that we can measure if the number of columns in a table has some effect on the performance.

The ***second flight*** includes queries, which restrict the fact table by one dimension. Such queries have the form:

```
select avg((tp_ fuel_at_departure - tp_fuel_at_destination)/tp_actual_trip_time)
as average_fuel_cost_per_trip
from trips, autobuseses
where tp_autobus_id = ab_id and
        ab_registration = [REGISTRATION] and
        tp_actual_trip_time between [TRIP_TIME] – 1 and
        [TRIP_TIME] + 1 and
        tp_paying_passengers > [PASSENGERS];
```

[*] For the 4th query two options were used with and without a *where* statement.

The query was repeated three times with different registration, trip time and passengers values so that different numbers of records are yielded.

The ***third flight*** includes queries, which restrict the fact table by two dimensions. Such queries have the form:

```
select sum(tp_actual_passengers * tp_ticket_price), ab_registration, st_name
from trips, autobuses, stations
where tp_autobus_id = ab_id and
        tp_destination_station_id = st_id and
        ab_registration = [REGISTRATION] and
        st_city = [CITY]
group by ab_registration, st_name
order by ab_registration, st_name;
```

The query was repeated three times with different registration and city values so that different numbers of records are yielded.

The ***fourth flight*** includes queries, which restrict the fact table by three dimensions. Such queries have the form:

```
select sum(tp_catering_revenue), ab_registration, st_name, sp_code
from trips, autobuses, stations
where tp_autobus_id = ab_id and
        tp_destination_station_id = st_id and
        tp_id = sp_trip_id and
        ab_available_seats = [AVAILABLE_SEATS] and
        st_valid_since > [DATE] and
        sp_distance_travelled > [DISTANCE]
group by ab_registration, st_name, sp_code
order by ab_registration, st_name, sp_code;
```

The query was repeated three times with different available seats, date and code values so that different numbers of records are yielded.

Since the dimensions in our schema are with greatly varying sizes, by choosing the dimension(s) and the values of the parameters we can provide a wide range of result sets from small (e.g. bellow 5%) to large (e.g. around 50%).

4 Read queries performance comparison

All measurements were done with Aqua Fold's Aqua Data Studio software. The query loads is separated into two sets – general and analytical. The general queries include simple inserts, updates, and deletes, as well as querying data for general reporting purposes. The analytical queries include typical data warehouse queries with aggregations and joins, usually with low column selection (bellow 10%).

4.1 General queries performance

The results of the general query performance can be seen in the table below.

Table 1. Flight one results - Oracle

	Q1	Q2	Q3	Q4.1[†]	Q4.2[†]
Autobuses	14 ms	10 ms	8 ms	5 ms	1 ms
Stations	15 ms	10 ms	9 ms	943 ms	1 ms
Trips	80 ms	10 ms	8 ms	3 m 20 s	1 ms
Stops	25 ms	10 ms	11 ms	5 m	1 ms

Table 2. Flight one results - Vertica

	Q1	Q2	Q3	Q4.1[†]	Q4.2[†]
Autobuses	30 ms	45 ms	27 ms	3 ms	1 ms
Stations	30 ms	45 ms	32 ms	922 ms	1 ms
Trips	235 ms	451 ms	53 ms	5 m 50 s	1 ms
Stops	30 ms	40 ms	29 ms	3m	1 ms

It should be noted that multiple operations were performed and the results were averaged. Also, with Oracle the first execution of each statement was significantly slower than the subsequent ones, while with Vertica there barely was any difference.

It is evident that Oracle is better when data manipulation language statements (safe from *select*) are concerned:

- For both updates and deletes of a single record, it is more or less independent of the number of columns or records in the underlying table;
- When inserts of a single record are considered, the number of records and the number of columns have some impact on the performance, with the number of columns having a higher one.

The query performance when selecting a single row is uniform across all tables; hence it is independent of table size and schema. The same does not apply, however, to selecting the whole table. In Oracle's case the number of records has a higher impact on the query execution time (e.g. the *select* for the Stops table took longer than the one for the Trips table). Naturally, it can be assumed that the number of columns would affect the performance as well, provided the number of records stays the same.

Vertica lags significantly behind Oracle for single row inserts, updates and deletes. The gap between the two gets wider with the increase of the number of columns in the table, which is to be expected considering the physical data models that both systems use. The inserts and updates take the highest performance hit, with the update of the Trips table being around 45 times slower than the same with Oracle. This can be explained by the fact that this is the table with the largest number of columns and considering Vertica's architecture it is to be expected that inserts, updates and deletes would be slower than with traditional RDBMSs like Oracle. Although having optimized the write performance with a separate write store [13], Vertica still cannot match the typical RDBMS when insert, update, and delete statements are concerned. The results for selecting single records and an entire table show that with Vertica the performance is dependent on the size of the result set in megabytes, and is rather

[†] For the 4th query two options were used, Q4.1 refers to the query without *where* and Q4.2 refers to the query with *where*.

independent of the number of columns. The comparison between the two systems when the whole relations are queried shows an interesting result – Vertica outperforms Oracle in the case of a large result set for a table with a relatively small number of columns (the Stops table).

4.2 Analytical queries performance

The rest of the flights of queries test the performance of both systems when analytical loads are concerned. They are separated into three flights measuring the performance of the RDBMSs when restricting one, two or three dimensions.

Second flight of queries. The second flight of queries has an original result set of around 100000 records. From it, by changing the values of the *[REGISTRATION]*, *[TRIP_TIME]*, and *[PASSENGERS]* parameters three different result sets were produced – with 800, 1500 and 3500 records. Using the average function one record was generated from the above result sets.

Table 3. Flight two results

	Q1	Q2	Q3
Oracle	605 ms	669 ms	700 ms
Vertica	171 ms	174 ms	192 ms

The results show that both DMBSs perform uniformly across the different queries in this flight with only slight slow downs when the number of records grows larger. Further a significant performance benefit from using the column store architecture is indicated. Judging by the measured response times Vertica is up to 3.65 times faster than Oracle when performing the single dimension restriction queries. The performance benefits are uniform across the selectivity, which varies from 0.8% to 3.5% for this flight. This confirms our previous findings [7] of column store's performance for analytical queries.

Third flight of queries. The third flight of queries has an original result set of around 100000 records. From it, by changing the values of the *[REGISTRATION]* and *[CITY]* parameters three different result sets were produced – with 150, 350 and 750 records. Using the sum function three records were generated from the above result sets (i.e. the group by statement produced three groups).

Table 4. Flight three results

	Q1	Q2	Q3
Oracle	627 ms	697 ms	789 ms
Vertica	207 ms	258 ms	267 ms

Again here we see that on small sets both DBMSs perform relatively the same regardless of the increasing number of records in each set. The reason most likely being the relatively small result sets in this group.

Varying the selectivity to lower the number of records in the original results sets to 0.15% - 0.75% shows that on such smaller results sets and more groups Vertica's performance benefit is slightly overcome, when compared to the previous flight of queries. It is still, however, around 3 times faster than Oracle in all the tests in this flight. It is interesting to compare Q3's results with the ones for Q1 from the previous test, as the result sets are almost equal, meaning that the difference in performance could be attributed to the restriction on two and not on one dimension. The comparison shows that the performance is being significantly impacted (0.6 times) by the restriction of both dimensions instead of just one.

Fourth flight of queries. The fourth flight of queries has an original result set of around 1750000 records. From it, by changing the values of the *[AVAILABLE_SEATS]*, *[DATE]* and *[DISTANCE]* parameters three different result sets were produced – with 35000, 160000 and 870000 records. Using the sum function 3000, 8000 and 12000 records respectively were generated from the above result sets (i.e. produced by the group by statement).

Table 5. Flight four results

	Q1	Q2	Q3
Oracle	4 s	13 s	32s
Vertica	750 ms	1 s	1 s

In this flight we observe a large range of selectivity – between 2% and 50%. These results are very important, as they are more representative for typical data warehouse queries. Even the 50% selectivity, however, stays well within the limits of our previously discovered performance threshold of 80% [7].

When comparing the results for a single DBMS a uniform behavior throughout this flight for Vertica is seen. The situation for Oracle, however, is different with its performance degrading with the increase of the size of the result sets.

The results confirm the superiority of Vertica over Oracle for queries targeting data analysis; with the performance reaching a 32 fold gain for the highest selectivity query. The observations clearly show that the higher the selectivity, the larger the gap between Vertica and Oracle. Comparing the results from the second flight of queries where the selectivity is close to 2% (Q2 and Q3) with the one for Q1 from this flight indicates that the size of the result set plays a significant role when the performance is considered, with the performance advantage of Vertica growing 1.5 times.

5 Conclusion

The purpose of this paper is to analyze the performance of analytical queries on commercial grade DBMSs - Oracle and Vertica. As a first step in this some aspects of the data models and the query execution optimizations of the two are discussed.

Next, in order to make the performance comparison itself a new benchmark inspired by the SSB, is modeled. This allowed the inclusion of new queries, which are not part of the original benchmark and the usage of different, commercial data.

Performing this new set of test flights gave interesting results for the performance of both systems. It can be concluded that the main disadvantage of a column store system in the face of Vertica is single row DML manipulations, safe from selects, as well as queries of non-analytical type – with large number of selected columns and rows. When analytical queries are considered, however, Vertica has a constant performance benefit of around 3 folds, which grows higher with the result sets that are to be processed to reach 32 times the performance of Oracle for certain queries. Having that in mind we believe that Vertica is the better candidate for data warehouse solutions, as they are usually bulk loaded with records on a predefined period of time and are mostly used for read queries where it has significant advantages over Oracle. For traditional transactional loads with lots of single row DML statements being issued constantly Oracle has superiority over Vertica and is our recommended DBMS.

Acknowledgment. This paper is supported by Sofia University "St. Kliment Ohridski" SRF under Contract 43/2013.

References

1. Abadi, D. et al.: Column-oriented database systems. Proceedings of the VLDB Endowment. pp. 1664–1665 (2009).
2. Abadi, D.J., Madden, S.R.: Column-Stores vs . Row-Stores : How Different Are They Really? SIGMOD. pp. 967–980 (2008).
3. Codd, E.F.: A relational model of data for large shared data banks. Communications of the ACM. 13, 6, 377–387 (1970).
4. Garcia-Molina, H. et al.: Database Systems: The Complete Book. Pearson Prentice Hall, Upper Saddle River, New Jersey 07458 (2009).
5. Greenwald, R. et al.: Oracle Essentials Oracle Database 11g. O'Reilly Media, Inc., Sebastopol (2008).
6. Harizopoulos, S. et al.: Performance Tradeoffs in Read-Optimized Databases. Proceedings of the VLDB Endowment. pp. 487–498 , Seoul, Korea (2006).
7. Kyurkchiev, H., Kaloyanova, K.: Read Optimization Based on Column-oriented DBMS. Doctoral Conference in Mathematics, Informatics and Education. pp. 58–66 St. Kliment Ohridski University Press (2013).
8. Lamb, A. et al.: The Vertica Analytic Database : C-Store 7 Years Later. Proceedings of the VLDB Endowment. pp. 1790–1801 (2012).
9. Loney, K.: Oracle Database 11g The Complete Reference. Oracle Press (2009).
10. Neil, B.O. et al.: The Star Schema Benchmark and Augmented Fact Table Indexing Outline of Talk. Performance Evaluation and Benchmarking, Lecture Notes in Computer Science. 5895, 237–252 (2009).
11. Neil, P.O. et al.: Star Schema Benchmark, (2009).
12. Oracle, A., Paper, W.: Query Optimization in Oracle Database10g Release 2 Query Optimization in Oracle Database 10g Release 2, (2005).
13. Stonebraker, M. et al.: C-store: a column-oriented DBMS. Proceedings of the 31st VLDB Conference. pp. 553–564 (2005).
14. Oracle Database Concepts 11g Release 2 (11.2) - Logical Storage Structures, http://docs.oracle.com/cd/E11882_01/server.112/e25789/logical.htm.
15. The Vertica ® Analytic Database Technical Overview White Paper, (2010).