DBDM Assignment 1

Eva van Weenen (s1376969), Paul Couzy (s1174347),
Alex van Vorstenbosch (s1526146), Irene Haasnoot (s1258745)

October 2018

Abstract

We evaluate the performance of two different database management systems (DBMS): MonetDB (column-oriented) and MariaDB (row-oriented). Using the TPC-H benchmark, we run two different queries on different types of hardware. Furthermore, we compare the performance of DBMSs with running the same queries in the Python 3.6 programming language. We evaluate the performance in terms of the run-time required. The difference between dedicated DBMS's and general programming languages is explored. It is found that the query time for DBMS is orders of magnitude faster than self written code and observed that hot and cold runs have a large influence on the run time of MonetDB and less influence on MariaDB.

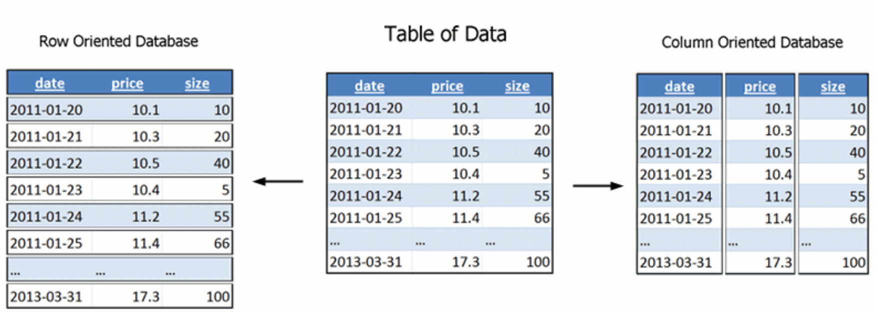
1 Introduction

Database management systems (DBMS) allow users to interact with a database by organizing, storing and retrieving data. Using a DBMS has the advantage that it allows users to access the database while maintaining the integrity of the data. However, it does cost more CPU and memory. Since the first database systems, technology has greatly advanced, leading to hardware and database management systems that have grown orders of magnitude in size, capability and performance. Nowadays we have several DBMS types, such as relational DBMS (applicable to many cases), NoSQL DBMS (applicable to unstructured data), in-memory DBMS (suitable for applications that need fast response times such as a telecommunication network) and columnar DBMS (suitable for data warehouses).

In this project we will focus on relational DBMS and column-oriented DBMS. Column-oriented DBMS are different from relational DBMS in the fact that they store their data in columns, instead of in rows. Figure 1 provides a visualization of the difference between row- and column-oriented DBMS. When large amounts of data are stored, column-oriented databases are often preferred when using database queries. Column-oriented DBMS only read the wanted data in the query by compressing the similar columnar data, while row-oriented DBMS scan all rows and then discard unwanted data. Using column-oriented DBMS therefore maximize performance and practical applications include data warehouses and customer relationship management.

In this project we will analyze the performance of two different DBMS and compare it with implementing the same database queries in Python 3.6. We use the TPC decision support benchmark H (TPC-H). The DBMS we will analyze are MonetDB (Idreos et al., 2012), the pioneer in column-store database management, and MariaDB, a fork of the relational DBMS MySQL which is row-oriented. We will therefore compare the performance of column-oriented and row-oriented DBMS. We run the queries on different hardware systems to analyze the effects of the hardware on the performance of Python and the DBMS.

In section 2 we explain the data and present the queries for which we conduct the experiments (2.1). We specify the tools we use and on which systems we run the queries (2.2, 2.3). In order to evaluate MonetDB, MariaDB and the python implementation, we specify the analysis of the different methods in order to compare them (2.4). In section 3 we present our results and verify the results for `SF-1` (4). A short conclusion is provided in section 5.



Row Oriented Database

date	price	size
2011-01-20	10.1	10
2011-01-21	10.3	20
2011-01-22	10.5	40
2011-01-23	10.4	5
2011-01-24	11.2	55
2011-01-25	11.4	66
...	...	...
2013-03-31	17.3	100

Table of Data

date	price	size
2011-01-20	10.1	10
2011-01-21	10.3	20
2011-01-22	10.5	40
2011-01-23	10.4	5
2011-01-24	11.2	55
2011-01-25	11.4	66
...	...	...
2013-03-31	17.3	100

Column Oriented Database

date	price	size
2011-01-20	10.1	10
2011-01-21	10.3	20
2011-01-22	10.5	40
2011-01-23	10.4	5
2011-01-24	11.2	55
2011-01-25	11.4	66
...	...	...
2013-03-31	17.3	100

Figure 1: Difference between row-oriented and column-oriented DBMS, source: `http://www.timestored.com/time-series-data/what-is-a-column-oriented-database`

2 Method

2.1 Data

TPC-H is an ad-hoc, decision support benchmark, consisting of business oriented queries and concurrent data modifications (Transaction Processing Performance Council (TPC), 2017). We use this benchmark to be able to compare the results of our tests, as it aims to use the same conditions for every test. The TPC-H benchmark consists of variable, synthetic data that a small product selling business could have. The database scheme for the TPC-H benchmark can be found in Figure 2. The database consists of items, that are ordered by customers and supplied by suppliers. The data is scalable and labelled as `SF-x` representing the size of the database. `SF-x` means an approximate size of scale factor `x` times 1GB in memory. For our project we will use `SF-1` and `SF-3`. The synthetic data is generated with the generator DBGEN provided by the TPC-H benchmark. TPC-H provides 22 queries that could be used by a product providing business, such as select all orders between a certain date and with a certain discount. For this project we will use Query 1 and Query 6, displayed in Tables 1 and 2 respectively.

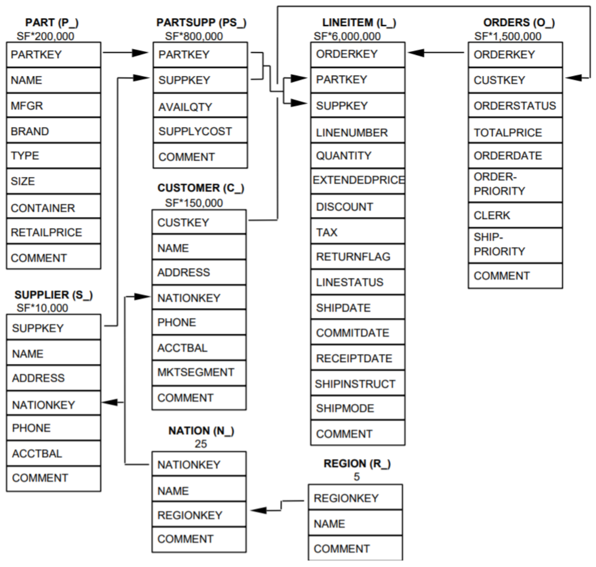


Figure 2: The TPC-H scheme, resembling the database of a small product selling business (Transaction Processing Performance Council (TPC), 2017)

```
select      l_returnflag,
            l_linestatus,
            sum(l_quantity) as sum_qty,
            sum(l_extendedprice) as sum_base_price,
            sum(l_extendedprice * (1 - l_discount)) as sum_disc_price,
            sum(l_extendedprice * (1 - l_discount) * (1 + l_tax)) as sum_charge,
            avg(l_quantity) as avg_qty,
            avg(l_extendedprice) as avg_price,
            avg(l_discount) as avg_disc,
            count(*) as count_order
from        lineitem
where       l_shipdate <= date '1998-12-01' - interval '90' day (3)
group by    l_returnflag, l_linestatus
order by    l_returnflag, l_linestatus;
```

Table 1: TPC-H Query 1

```
select  sum(l_extendedprice * l_discount) as revenue,
from    lineitem
where   l_shipdate >= date '1994-01-01'
        and l_shipdate < date '1994-01-01' + interval '1' year
        and l_discount between 0.06 - 0.01 and 0.06 + 0.01
        and l_quantity < 24;
```

Table 2: TPC-H Query 6

2.2 Tools

We will run the above named TPC-H queries in two DBMS and in Python 3.6

MonetDB MonetDB is a column-store database management system developed by the Centrum Wiskunde & Informatica (CWI) in the Netherlands. It is a pioneer in column-stored database management, consisting of three layers of software, each with its own optimizers. MonetDB makes optimal use of hierarchical memory systems and presents new techniques to handle rapidly changing workloads (Idreos et al., 2012). Here we use MonetDB version 11.31.7

MariaDB MariaDB is an open-source fork of MySQL, a relational DBMS and made by the original developers of MySQL. MariaDB has a wide range of applications with users such as Wikipedia, WordPress.com and Google and is often used because it is fast, scalable and robust (MariaDB Core Team, 2018). Here we use MariaDB version 10.3

Python We implement two queries in the high-level programming language *Python* (version 3.6.6) (Python Core Team, 2018) using the *pandas*-library (pandas Core Team, 2018). This library is especially useful for analyzing and using large data structures.

2.3 Hardware characteristics

The two queries are run on two different systems with the following characteristics:

Setup #1 STRW University

- CPU: 3.20GHz Intel(R) Core(TM) i5-3470, 32K L1d cache, 32K L1i cache, 256K L2 cache, 6144K L3 cache
- Main memory: 8GB
- Disk: 500GB HDD
- OS: Fedora 21.0

The data used in this setup was stored on a network disk on a different STRW server that was mounted on the computer used. This disk has a storage capacity of 1.5 TB.

Setup #2 Personal Laptop

- CPU: 2.50GHz Intel(R) Core(TM) i5-7200U, 32K L1d cache, 32K L1i cache, 256K L2 cache, 3072K L3 cache
- Main memory: 8GB RAM DDR4
- Disk: 256GB SSD
- OS: Ubuntu 16.04 LTS

The first setup is a university computer of Leiden Observatory (STRW). For this setup we only test MonetDB as we were not authorized to install MariaDB on this computer.

2.4 Performance analysis

We evaluate MonetDB, MariaDB and Python 3.6 using various functions to measure the time elapsed. For Python 3.6 the absolute CPU time to process the query is measured. For MonetDB the 'run' time parameter is used to capture the server-side absolute time to execute the physical (MAL) plan. For MariaDB again this absolute server-side time was used. This absolute clock time is influenced by having a hot or cold run, but gives a representation of the time required to run a query which can be compared between different systems. System time is more consistent, where the counter is monotonically increased based on the CPU frequency. However, the system time usually cannot be compared between different systems. Therefore use was made of these absolute time measures.

3 Results

In Figure 3 and 4 the measured run times are presented for Query 1 and Query 6 respectively. On the top row the run times of MonetDB can be found, in the middle those for MariaDB and on the bottom you find the times for our own implementation in Python. The columns represent the different scaling factors `SF-1` and `SF-3`. The red and blue lines indicate the two different setups.

Note that some graphs are altered for clarity. For instance the run times for both queries with python on setup # 1 are divided by `1.5` purely to increase the visualization quality. The actual run times are thus `1.5` times higher. Furthermore, in some figures the two arrows indicate that the run times for a certain run are both much larger than the other run times and therefore cut off. Next to the arrow the run time for this specific run is portrayed.

For both queries we observe that both DBMS are much faster than Python 3.6. This makes sense as DBMS are developed for the purpose of database queries. Furthermore we observe that MonetDB is much faster than MariaDB. This also makes sense as column-oriented DBMS are much more efficient in database queries than row-oriented DBMS.

If we look at the figures of MonetDB we see that the first run takes significantly longer than the subsequent runs. This is a clear example of the difference between a 'hot run' and a 'cold run'. The first run is cold because the DBMS just started and no (benchmark-relevant) data is preloaded into the system's main memory yet. For the later runs query-relevant data is available and less time is needed for the queries.

In all cases MonetDB performs better than MariaDB and MariaDB performs better than the Python

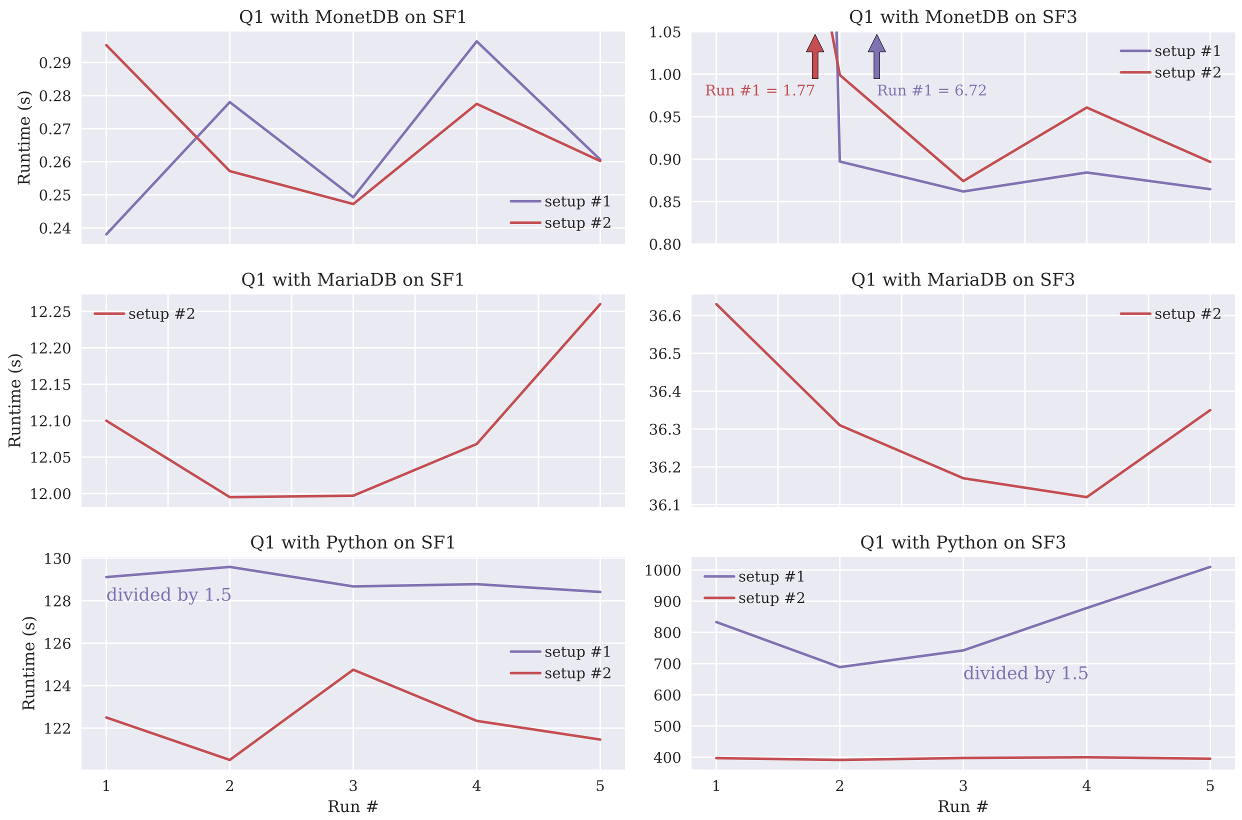


Figure 3: The time performance of Q1 for data sets SF1 (left) and SF3 (right) as given by different tools MonetDB (top), MariaDB (middle) and Python 3.6 (bottom). Setup #1 (blue line) and Setup #2 (red line) indicate two separate hardware systems as specified in section 2.3.

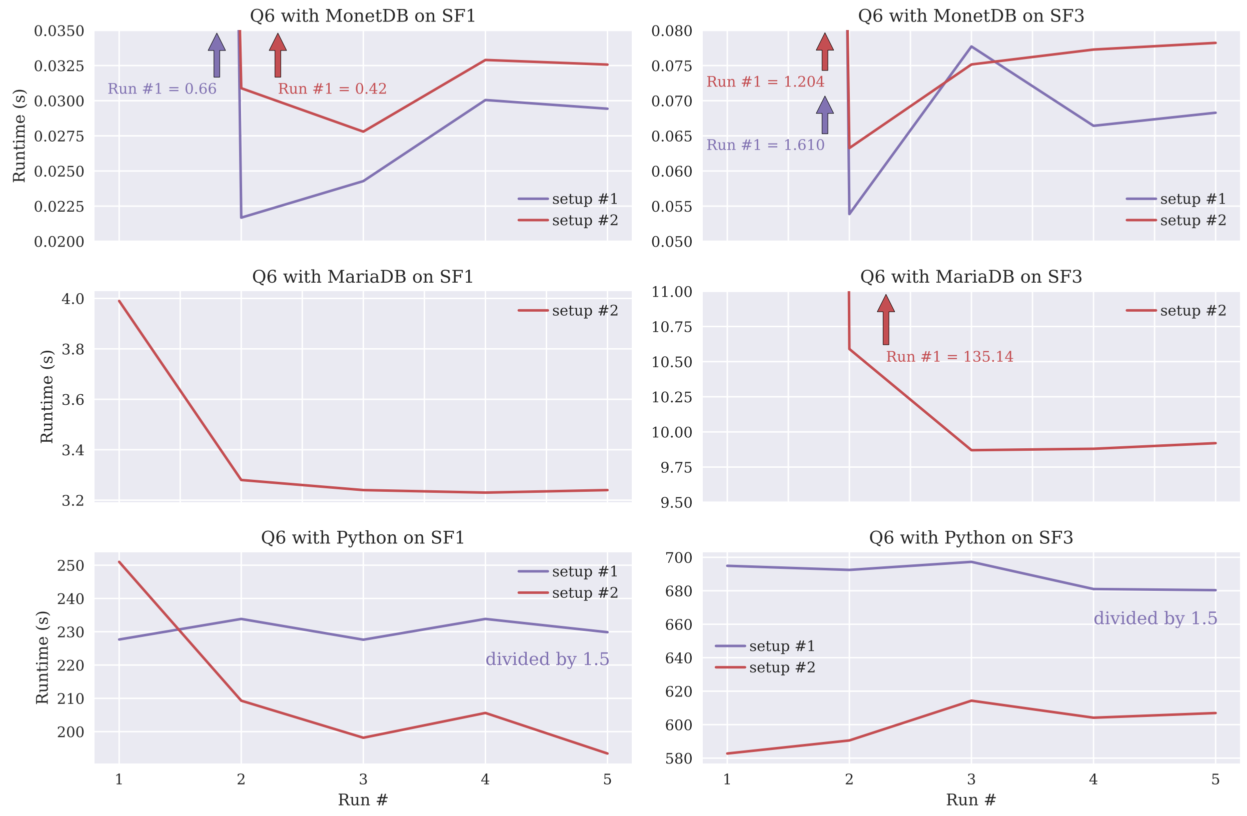


Figure 4: The time performance of Q6 for data sets SF1 (left) and SF3 (right) as given by different tools MonetDB (top), MariaDB (middle) and Python 3.6 (bottom). Setup #1 (blue line) and Setup #2 (red line) indicate two separate hardware systems as specified in section 2.3.

implementation. Although the difference in times change for the queries and the scale factors. The run times of Q1 for MonetDB between SF-1 and SF-3 scale more than the scale factor of 3, but not much. For Q6 it only differs by a factor of 2, which is less than the scale factor. It seems MonetDB is very efficient in handling large data sets even for a cold run.

MariaDB however is less efficient. The run time of Q6 for SF-3 is 30 times larger than that of SF-1. After the hot run it is only a factor of 3. This means that MariaDB is very inefficient in a cold run. The run time of Q1 for SF-3 is 3 times larger than that of Q1 for SF-1 of MariaDB, this means that for Q1 MariaDB is efficient.

The run times of the Python implementation scale linearly with the scale factor of 3 between the SF-1 and SF-3 data sets. Although the Python implementation is not efficient in absolute run times, it does scale efficiently.

For the Python implementation we observe that setup #2 is much faster than setup #1. This could be due to the fact that setup #2 uses an SSD disk and by the fact that the data of setup #1 was stored on a disk on a different server. For the MonetDB implementation we observe that setup #1 performs slightly worse than setup #2. The difference in performance can be explained by the difference in CPU clock speed.

4 Discussion

The output of the queries for MonetDB, MariaDB and Python are checked by the provided answers of the TPC-H Query 1 and Query 6. The output is checked by hand, as the output of the specific queries are only a few lines and this does not take much time. At first MonetDB delivered the wrong answers, all the sums were doubled, but the output of the averages were correct. This trouble was caused by loading in the SF-1 data twice. And this explanation makes sense, as the doubling of the data does not alter the averages, but does double the sum. After this problem was solved by loading in the data only one time, MonetDB, MariaDB and Python delivered the correct answers.

To improve our research it might be interesting to look at the performance of DBMS on hardware that is completely different. For now, our hardware was both Linux-based and had a CPU clock speed of the same orders of magnitude. It might be interesting to see what happens if we use a computer with a much lower or much higher CPU clock speed.

Furthermore it will be interesting to compare MonetDB with other column-oriented DBMS, to compare its performance with other column-oriented DBMS.

5 Conclusion

We observe that for the TPC-H benchmark, a column-oriented DBMS is the best DBMS for running database queries. MonetDB is many orders of magnitude faster than MariaDB and Python 3.6. Relational DBMS are a good second choice as they are still better than using no DBMS at all. We observe that hot and cold runs have a large influence on the run time of MonetDB and less influence on MariaDB.

Bibliography

Stratos Idreos, Fabian Groffen, Niels Nes, Stefan Manegold, K. Sjoerd Mullender, and Martin L. Kersten. Monetdb: Two decades of research in column-oriented database architectures. *IEEE Data Eng. Bull.*, 35 (1):40–45, 2012. URL http://sites.computer.org/debull/A12mar/monetdb.pdf.

MariaDB Core Team. *MariaDB, version 10.3.* MariaDB Corporation, 2018. URL https://mariadb.com/.

pandas Core Team. *pandas: An open source, BSD-licensed library, version 0.23.4*, 2018. URL https://pandas.pydata.org/.

Python Core Team. *Python: A dynamic, open source programming language, version 3.6.6.* Python Software Foundation, 2018. URL https://www.python.org/.

Transaction Processing Performance Council (TPC). *TPC Benchmark*$^{TM}H, 2017. URL.$