

Subscribe to DeepL Pro to edit this document.  
Visit www.DeepL.com/Pro for more information.

|  |
| --- |
| Winter semester 2021/21 |
| Comparison of MonetDB, InnoDB and MariaDB ColumnStore in terms of performance, usability and ease of installation |
|  |
| Semester assignment |
| submitted by  Benedikt Scheffbuch |

Table of contents

[1 Introduction 5](#_Toc64569875)

[1.1 DBMS vs. DB engine vs. database (model) 5](#_Toc64569876)

[1.2 OLTP vs. OLAP 6](#_Toc64569877)

[1.3 Column-oriented data memory 7](#_Toc64569878)

[1.4 MonetDB, InnoDB and MariaDB ColumnStore 8](#_Toc64569879)

[2 Installation 10](#_Toc64569880)

[2.1 Installing a Virtual Machine and Operating System 11](#_Toc64569881)

[2.2 Installation MonetDB 11](#_Toc64569882)

[2.3 Installation MariaDB 11](#_Toc64569883)

[2.4 Installation InnoDB 12](#_Toc64569884)

[3 Operation & Results 13](#_Toc64569885)

[4 Conclusion and evaluation 15](#_Toc64569886)

# Introduction

In this work, the three database engines, or database management systems InnoDB, MonetDB and Columnstore are essentially to be tested for performance, ease of installation and user-friendliness. Other questions defined are "Powerfulness of the language (which applications are particularly well supported?)" and "How can certain queries be executed particularly quickly?" All findings are to be summarized in the form of a Word document (including source code) of about 10 pages.

In addition to this task, I have created a [GitHubrepository](https://github.com/BingeCode/columnstore-innodb-monetdb) under which I have deposited the entire program code as well as further information about the project in order to keep this Word document clear. Furthermore, the dataset used in the comparison can be found on [Kaggle.](https://www.kaggle.com/bingecode/us-national-flight-data-2015-2020) I pulled the data from the US Bureau of Transportation Statistics and painstakingly cleaned it to make it usable. In doing so, I ended up doing a lot more work than perhaps necessary, but my hope is that it might still be useful to someone else. For more info on GitHub and my approach, see the README.md of my repository.

Before this paper deals with the main properties to be investigated, a theoretical introduction follows.

## DBMS vs. DB engine vs. database (model)

A database consists of two parts: the management software, called database management system (**DBMS**), and the set of data to be managed, the database (DB) in the narrower sense. The basis for structuring the data and their relationships to each other is the database model, which is defined by the DBMS manufacturer. A distinction is made between hierarchical, network, relational, object-oriented and document-oriented models. The most common form of a database is the relational database. Also in this work only **relational Dabenbankmodelle** are treated. [[1]](#footnote-1)

A **database engine** (or storage engine) is the underlying software component that a database management system (DBMS) uses to create, read, update, and delete data from a database. Many modern DBMSs support multiple storage engines within the same database system. [[2]](#footnote-2)As an example, MariaDB supports both InnoDB (default) and ColumnStore.

## OLTP vs. OLAP

**OLTP** (Online Transaction Processing) is characterized by large database transactions that perform insertions, updates or deletions. OLTP-type databases specialize in processing queries quickly and maintaining data integrity while being accessed in different environments. Their effectiveness is measured by the number of transactions per second (tps). It is common that the tables with parent-child relationships (after implementing the normalization form) reduce redundant data in a table.

Records in a table are usually processed and stored sequentially in a row-oriented manner and indexed with unique keys to optimize data retrieval or writing. This is also common in MySQL, especially when dealing with large inserts or high concurrent writes or bulk inserts. Most of the storage engines supported by MariaDB are suitable for OLTP applications - **InnoDB** (the default storage engine since 10.2), XtraDB, TokuDB, MyRocks or MyISAM/Aria.

Applications such as CMS, FinTech, web apps often deal with large writes and reads and often require high throughput. Making these applications work often requires deep expertise in high availability, redundancy, resilience and recovery.

**OLAP** (Online Analytical Processing ) addresses the same challenges as OLTP, but uses a different approach (especially for data retrieval). OLAP deals with larger data sets and is often used for data warehousing and for business intelligence applications. It is commonly used for business performance management, planning, budgeting, forecasting, financial reporting, analytics, simulation modeling, knowledge discovery, and data warehouse reporting.

Data stored in OLAP is usually not as critical as that stored in OLTP. This is because most data can be simulated coming out of OLTP and then fed into your OLAP database. This data is typically used for bulk loading, which is often needed for business analysis that is eventually rendered into visual charts. OLAP also performs multidimensional analysis of business data, providing results that can be used for complex calculations, trend analysis, or sophisticated data modeling.

OLAP usually stores data persistently in a columnar format. In MariaDB **ColumnStore**, however, records are broken down based on their columns and stored separately in a file. In this way, data retrieval is very efficient, as only the relevant column referenced by your SELECT statement in the query is searched.

You can think of it this way: OLTP processing takes care of the day-to-day and important data transactions that run your business application, while OLAP helps manage, predict, analyze, and better market the product - the building blocks that make up a business application. [[3]](#footnote-3)

## Column-oriented data store

When a database grows to millions of records spread across many tables and business intelligence/science becomes the predominant application domain, a columnar database management system is required. Unlike traditional row stores such as MySQL and PostgreSQL, a column store offers a modern and scalable solution that does not require a large hardware investment. [[4]](#footnote-4)

A column-oriented DBMS (or columnar database management system) is a database management system (DBMS) that stores data tables by column rather than by row. The practical use of a column store versus a row store differs little in the relational DBMS world. Both column-based and row-based databases can use traditional database query languages such as SQL to load data and perform queries. Both row and column databases can become the backbone in a system to provide data to common extraction, transformation, loading (ETL) and data visualization tools. However, by storing data in columns rather than rows, the database can more accurately access the data it needs to answer a query, rather than scanning and discarding unwanted data in rows.

In short, column-oriented data stores are more efficient when not all properties of an instance are needed, while row-based data stores are more efficient when (almost) all properties of an instance are needed.

Comparisons between row-oriented and column-oriented systems typically focus primarily on the efficiency of disk access, which consumes considerable time compared to other computer operations. Column-oriented systems are more efficient when an aggregate has to be formed over many rows but only a few columns, because then, in contrast to the row-oriented system, one only has to read these columns and not all of them. Row-oriented systems are more efficient when many columns of a single row are needed at the same time and when the row width is very large, because then the whole row can be read with a single disk access. [[5]](#footnote-5)

## MonetDB, InnoDB and MariaDB ColumnStore

**MonetDB** is a free, column-oriented relational database management system and also the database engine. It is an open source project that has its origins at Centrum Wiskunde & Informatica in Amsterdam. MonetDB is ideally suited for applications where the hot set of the database - the part that is actually touched - can be largely kept in main memory, or where a few columns of a wide relational table are sufficient to handle individual queries. MonetDB has been heavily influenced by scientific experiments in understanding the interplay between algorithms and application requirements. It has made MonetDB an extensible database system with hooks at all levels in the software stack. [[6]](#footnote-6)

**InnoDB** is a free, row-based storage subsystem (database engine) for the MySQL database management system. Its main advantage over other storage subsystems of the MySQL relational database management system (RDBMS) is that transaction security and referential integrity are ensured via foreign keys.[[7]](#footnote-7) It is set as a default in most binaries distributed by MySQL. InnoDB is also present in MariaDB, but it is now such a heavily modified version that MariaDB refers to it as a standalone version.

**MariaDB** is a community-developed, commercially supported fork of MySQL that will operate as free and open source software under the GNU General Public License. Development is led by some of the original developers of MySQL, who forked it due to concerns over its acquisition by Oracle Corporation in 2009. MariaDB Corporation announced the release of its first Big Data analytics engine, **MariaDB ColumnStore**, on April 5, 2016. It is based on both a fork of InfiniDB and open source community contributions.[[8]](#footnote-8) MariaDB ColumnStore is a column-based storage engine that uses a massively parallel distributed data architecture. It is a columnar storage system created by porting InfiniDB 4.6.7 to MariaDB and released under the GPL license. [[9]](#footnote-9)

# Installation

For the installation of the two DBMS MonetDB and MariaDB it is recommended to create a virtual machine running a distribution of Linux. The reason for this is that most DBMSs are designed for Linux. MacOS and especially Windows are often secondary in the development. For example, MariaDB ColumnStore does not exist on Windows, while InnoDB and MonetDB run on Windows without any problems (installing and running MonetDB on Windows has also been tested). Another advantage of using a virtual machine, in addition to the central administration of all DBMSs, is that the available resources (CPU, RAM) can be specified. This ensures that an objective comparison is possible. Recommended are the distributions **CentOS** & RHEL as well as Ubuntu & Debian, which specify both MonetDB and MariaDB as possible platforms. In the context of this work with CentOS 8 was worked. The installation went off without a hitch. There is also CentOS Stream, but this is a quasi-beta of CentOS, so it is better to use CentOS 8.

To set up a virtual machine, you need - who would have thought - virtualization software. The most common free applications for desktop systems are VirtualBox from Oracle and VMware Workstation from VMware, Inc.

When doing this work on a Windows machine, I kept running into technical problems with Oracle's VirtualBox, which is why I would rather advise VMware. That doesn't have to mean that VirtualBox is the worse choice - it just didn't work on my system. Ultimately, though, it doesn't matter which virtualization software you use. The following guide is based on Windows as the host system and VMware Workstation as the virtualization software.

## Installing a virtual machine and operating system

For the installation of the complete virtual machine is needed:

1. Install VMware Workstation Player ([Download](https://www.vmware.com/products/workstation-player/workstation-player-evaluation.html))
2. CentOS ISO file ([download](https://www.centos.org/download/))
3. Start VMware and follow this [guide](https://cloudlinuxtech.com/install-centos-in-vmware/)
   1. As hardware requirements, 2 cores of CPU, 8GB of RAM and 20GB of fixed disk space should be enough.
   2. Ensure correct language, time zone and keyboard layout during installation
   3. Set login as root with password root (for simplicity) for necessary permissions later on
4. Log in to CentOS as user root (if necessary restart the system and click on 'Not this user?')
5. Download the flights.csv file from [Kaggle](https://www.kaggle.com/bingecode/us-national-flight-data-2015-2020), unzip it and put it into the home folder.
6. Perform installation of MonetDB and MariaDB (see next chapters)

## Installation MonetDB

1. Follow the instructions for installing MonetDB on CentOS [here](https://www.monetdb.org/downloads/epel/), specifically the steps:
   1. yum install <https://dev.monetdb.org/downloads/epel/MonetDB-release-epel.noarch.rpm>
   2. yum install MonetDB-SQL-server5 MonetDB-client
2. Then mark the instructions from [GitHub](https://github.com/BingeCode/columnstore-innodb-monetdb/blob/main/SQL%20code/monetdb.sql) completely with CTRL + A, copy them with CTRL + C and then execute them in one go in the terminal in CentOS (The password for MonetDB is monetdb by default)
3. The script started the MonetDB server (daemon), connected to the server, created a database in which the monetdb table is created and then fed the complete CSV file. After that MonetDB is ready for various [queries](https://github.com/BingeCode/columnstore-innodb-monetdb/blob/main/SQL%20code/queries.sql).
4. Note that importing from the CSV file may take time. For reference, it is worth looking at the [README](https://github.com/BingeCode/columnstore-innodb-monetdb#csv-import-timings).

## Installation MariaDB

1. Follow the instructions for installing MariaDB on CentOS [here.](https://mariadb.com/docs/deploy/community-single-columnstore-cs105-centos8/)
   * Not all steps (optimizing the kernel parameters, disabling the Linux security module, configuring the letter encoding) are probably necessary, but should be done for safety's sake. However, the actual installation starts from [here](https://mariadb.com/docs/deploy/community-single-columnstore-cs105-centos8/#columnstore-installation). The following sections after 'ColumnStore Installation' are optional.
2. Starting the relevant MariaDB processes (servers) is described from [here on.](https://mariadb.com/docs/deploy/community-single-columnstore-cs105-centos8/#start-the-columnstore-processes)
3. Then, analogous to point 2) under 2.2 Installation MonetDB, execute the instructions from [GitHub](https://github.com/BingeCode/columnstore-innodb-monetdb/blob/main/SQL%20code/columnstore.sql) in the terminal
4. Importing the CSV file should be quite fast, at least much faster than with MonetDB (see [README](https://github.com/BingeCode/columnstore-innodb-monetdb#csv-import-timings)).

## Installation InnoDB

InnoDB does not need to be installed separately, since it is a standard part of MariaDB. So as long as MariaDB has been installed, InnoDB can be used as well. To use an InnoDB table, simply follow the steps in the corresponding [Innodb.sql file.](https://github.com/BingeCode/columnstore-innodb-monetdb/blob/main/SQL%20code/innodb.sql)

Note that loading the CSV file into an InnoDB table takes significantly longer than MonetDB or MariaDB. For reference values, look at the [README](https://github.com/BingeCode/columnstore-innodb-monetdb#csv-import-timings).

# Operation & Results

Basically, all three DB engines, or two DBMSs, can be operated in the same way. They can all be accessed via the terminal with SQL. However, my tests showed that MonetDB is significantly more flexible in the use of SQL and significantly better performing than MariaDB ColumnStore. In none of the queries shown in the README did MonetDB get over a value of one second and stayed in the low single digits of a second even for the more complex queries, which unfortunately could not be used with all engines (more on this below).

To compare the engines with each other, I had the goal to ask queries with increasing complexity and compare the performance based on them. But unfortunately it turned out that e.g. subqueries like WITH SOME\_TABLE AS (SELECT COLUMN FROM TABLE) ... or IN (SELECT ...) are not possible with ColumnStore by default. There are [configuration settings you](https://mariadb.com/kb/en/columnstore-system-variables/#operating-mode) can make that ColumnStore understands row-oriented queries, however, even after extensive research, these measures did not work on my system, to my regret. There are a few attempts at more complex queries under queries[.sql,](https://github.com/BingeCode/columnstore-innodb-monetdb/blob/main/SQL%20code/queries.sql) but all of them only run under MonetDB.

InnoDB understands all of the more complex queries that ColumnStore does not, but was not tested further due to the long query timings of 10m upwards per query.

Also worth mentioning are the import procedures of the different storage engines. The results of the different methods can be found under the section. Here it becomes clear that the column-based stores have a great advantage in terms of import speed. Especially ColumnStore scores with its [cpimport](https://mariadb.com/docs/solutions/columnstore/load-columnstore-data/#cpimport) procedure.

The installation of the two DBMSs takes about the same amount of time if you omit the optional steps before and after the actual installation when installing ColumnStore. However, as a beginner, you don't necessarily know that, and you don't find your way very well through the opaque jungle of information on the MariaDB page. The MonetDB page is very sparse, but you can quickly get to a running system. The installation of both systems is relatively simple, but it is complicated by the need to set up a virtual machine and operating system beforehand. Compared to ColumnStore, MonetDB scores not only with its fabulous performance, but also with the fact that it runs on Windows and thus does not require a virtual machine. It should also be mentioned that there were never any complications with the software when using MonetDB, while with ColumnStore you often had to fiddle around and google.

The results can also be found again in the [README](https://github.com/BingeCode/columnstore-innodb-monetdb#performance) on GitHub.

Furthermore, there are far-reaching possibilities to increase the performance of the individual engines. Especially with MariaDB Columnstore, numerous parameters can be optimized to increase [performance](https://mariadb.com/kb/en/columnstore-performance-tuning/). In InnoDB, too, the use of indexes could increase performance in certain cases. MonetDB also offers [many possibilities to](https://www.monetdb.org/Documentation/Tutorial/AdvancedFeatures) better adjust the system to certain conditions, but already comes out-of-the-box with numerous optimizations, even at runtime.

# Conclusion and evaluation

For a final evaluation, it is important to note that this comparison is a very specific and probably unrealistic scenario. Not only is the data set of 2.7GB a tiny amount of data compared to the usual terabytes in Big Data applications, but also the hardware used in the virtual machine does not make a database administrator's heart beat faster. Nevertheless, this comparative work made clear tendencies, namely the superiority of column-oriented databases in aggregation queries and the superiority of row-oriented databases in simple reads over the complete row. As mentioned in Section 1.2 OLTP vs. OLAP, column-oriented databases are also often thought of as extensions of row-based (OLTP) databases. Regular business is handled by OLTP and Big Data/reporting/data mining business is handled by OLAP.

Overall, however, MonetDB is the clear winner in this scenario, with a drastically shorter query time than the other two DBs, an acceptable import speed, and greater flexibility in query design than ColumnStore. However, ColumnStore enjoys much greater popularity in the community. Good community support is essential in open source projects, which is probably a big reason for ColumnStore's greater popularity, even though MonetDB is perhaps the better performing system from a technical standpoint.

Ultimately, the question still arises as to why MonetDB is not better known. According to this [[10]](#footnote-10)source, MonetDB is only ranked 120th among all DBMSs and 60th among all relational DBMSs, while MariaDB is ranked 12th and 8th, respectively. Even after extensive Internet research, I found hardly any field reports from production use of MonetDB. However, as the official website describes, MonetDB is a DBMS that is primarily used in research projects. [[11]](#footnote-11)Nevertheless, recent developments in the last few years indicate that MonetDB could become increasingly important in the commercial sector as well.

Finally, it is worth looking at a similar comparison test by [ReportServer](https://reportserver.net/blog/2016/06/20/mariadb-columnstore-vs-innodb-vs-monetdb/), which also examined the performance of these three database engines in various scenarios. The results were similar, although there were a few scenarios where ColumnStore was superior to MonetDB. So the future of both open source projects should continue to be watched with interest and the use of different DB engines should be weighed on a case-by-case basis, even though significant results can already be achieved compared to row-based engines. Other interesting column-based stores such as Clickhouse, Apache Spark, etc. should of course also receive attention.

1. https://de.wikipedia.org/wiki/Datenbank [↑](#footnote-ref-1)
2. https://de.wikipedia.org/wiki/Datenbank-Engine [↑](#footnote-ref-2)
3. https://dbsysupgrade.com/tutorial-mariadb-columnstore-for-mysql-admins/ [↑](#footnote-ref-3)
4. https://www.monetdb.org/content/column-store-features [↑](#footnote-ref-4)
5. https://de.wikipedia.org/wiki/Spaltenorientierte\_Datenbank [↑](#footnote-ref-5)
6. https://www.monetdb.org/content/column-store-features [↑](#footnote-ref-6)
7. https://de.wikipedia.org/wiki/InnoDB [↑](#footnote-ref-7)
8. https://en.wikipedia.org/wiki/InfiniDB [↑](#footnote-ref-8)
9. https://mariadb.com/kb/en/mariadb-columnstore/ [↑](#footnote-ref-9)
10. https://db-engines.com/en/system/MariaDB%3bMonetDB [↑](#footnote-ref-10)
11. https://www.monetdb.org/AboutUs [↑](#footnote-ref-11)