|  |
| --- |
| Wintersemester 2021/21 |
| Vergleich von MonetDB, InnoDB und MariaDB ColumnStore hinsichtlich Performanz, Benutzerfreundlichkeit und Einfachheit der Installation |
|  |
| Semesteraufgabe |
| vorgelegt von  Benedikt Scheffbuch |

Inhaltsverzeichnis

[1 Einleitung 5](#_Toc64569875)

[1.1 DBMS vs. DB-Engine vs. Datenbank(-modell) 5](#_Toc64569876)

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

[1.3 Spaltenorientierter Datenspeicher 7](#_Toc64569878)

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

[2 Installation 10](#_Toc64569880)

[2.1 Installation einer virtuellen Maschine und Betriebssystem 11](#_Toc64569881)

[2.2 Installation MonetDB 11](#_Toc64569882)

[2.3 Installation MariaDB 11](#_Toc64569883)

[2.4 Installation InnoDB 12](#_Toc64569884)

[3 Bedienung & Resultate 13](#_Toc64569885)

[4 Fazit und Auswertung 15](#_Toc64569886)

# Einleitung

In dieser Arbeit sollen die drei Datenbank-Engines, bzw. Datenbankverwaltungssysteme InnoDB, MonetDB und Columnstore im Wesentlichen auf die Leistung, Einfachheit der Installation und Benutzerfreundlichkeit geprüft werden. Als weitere Fragen wurde definiert: „Mächtigkeit der Sprache (welche Anwendungen werden besonders gut unterstützt?)“ und „Wie können bestimmte Abfragen besonders schnell ausgeführt werden?“ Alle Erkenntnisse sollen in Form eines Word-Dokuments (inkl. Quellcode) von ca. 10 Seiten zusammengefasst werden.

Zusätzlich zu dieser Aufgabenstellung habe ich ein [GitHub Repository](https://github.com/BingeCode/columnstore-innodb-monetdb) angelegt, unter dem ich den gesamten Programmcode sowie weitere Informationen zum Projekt hinterlegt habe, um dieses Word-Dokument übersichtlich zu halten. Weiterhin ist der Datensatz, der im Rahmen des Vergleichs genutzt wurden auf [Kaggle](https://www.kaggle.com/bingecode/us-national-flight-data-2015-2020) zu finden. Die Daten habe ich vom US-amerikanischen Amt für Verkehrsstatistiken gezogen und aufwändig gesäubert, um sie nutzbar zu machen. Damit habe ich mir am Ende viel mehr Arbeit gemacht als vielleicht notwendig, doch meine Hoffnung ist, dass sie jemand anderem noch nützen könnte. Für weitere Infos zu GitHub und meinem Vorgehen findet man mehr Infos auf der README.md meines Repository.

Bevor in dieser Arbeit auf die im Wesentlichen zu untersuchenden Eigenschaften eingegangen wird, folgt zunächst eine theoretische Einführung.

## DBMS vs. DB-Engine vs. Datenbank(-modell)

Eine Datenbank besteht aus zwei Teilen: der Verwaltungssoftware, genannt Datenbankmanagementsystem (**DBMS**), und der Menge der zu verwaltenden Daten, der Datenbank (DB) im engeren Sinn. Grundlage für die Strukturierung der Daten und ihrer Beziehungen zueinander ist das Datenbankmodell, das durch den DBMS-Hersteller festgelegt wird. Man unterscheidet zwischen hierarchischen, netzwerkartigen, relationalen, objektorientierten und dokumentenorientierten Modellen. Die gebräuchlichste Form einer Datenbank ist die relationale Datenbank. Auch in dieser Arbeit werden nur **relationale Dabenbankmodelle** behandelt.[[1]](#footnote-1)

Eine **Datenbank-Engine** (oder Speicher-Engine) ist die zugrundeliegende Softwarekomponente, die ein Datenbankverwaltungssystem (DBMS) verwendet, um Daten einer Datenbank zu erstellen, zu lesen, zu aktualisieren und zu löschen. Viele moderne DBMS unterstützen mehrere Speicher-Engines innerhalb desselben Datenbanksystems. [[2]](#footnote-2)Als Beispiel unterstützt MariaDB sowohl InnoDB (Standard) als auch ColumnStore.

## OLTP vs. OLAP

**OLTP** (Online Transaction Processing) ist durch große Datenbanktransaktionen gekennzeichnet, die Einfügungen, Aktualisierungen oder Löschungen vornehmen. Datenbanken vom Typ OLTP sind auf die schnelle Verarbeitung von Abfragen und die Wahrung der Datenintegrität spezialisiert, während auf sie in verschiedenen Umgebungen zugegriffen wird. Ihre Effektivität wird an der Anzahl der Transaktionen pro Sekunde (tps) gemessen. Es ist üblich, dass die Tabellen mit Eltern-Kind-Beziehungen (nach der Implementierung der Normalisierungsform) redundante Daten in einer Tabelle reduzieren.

Datensätze in einer Tabelle werden üblicherweise zeilenorientiert sequenziell verarbeitet und gespeichert und mit eindeutigen Schlüsseln indiziert, um das Abrufen oder Schreiben von Daten zu optimieren. Dies ist auch bei MySQL üblich, insbesondere wenn es um große Einfügungen oder hohe gleichzeitige Schreibvorgänge oder Masseneinfügungen geht. Die meisten der von MariaDB unterstützten Speicher-Engines sind für OLTP-Anwendungen geeignet - **InnoDB** (die Standardspeicher-Engine seit 10.2), XtraDB, TokuDB, MyRocks oder MyISAM/Aria.

Anwendungen wie CMS, FinTech, Web-Apps haben oft mit großen Schreib- und Lesevorgängen zu tun und erfordern oft einen hohen Durchsatz. Damit diese Anwendungen funktionieren, ist oft ein tiefes Fachwissen über Hochverfügbarkeit, Redundanz, Ausfallsicherheit und Wiederherstellung erforderlich.

**OLAP** (Online Analytical Processing ) befasst sich mit den gleichen Herausforderungen wie OLTP, verwendet jedoch einen anderen Ansatz (insbesondere bei der Datenabfrage). OLAP befasst sich mit größeren Datenmengen und wird häufig für Data Warehousing und für Business Intelligence-Anwendungen verwendet. Es wird häufig für Business Performance Management, Planung, Budgetierung, Forecasting, Finanzberichte, Analysen, Simulationsmodelle, Knowledge Discovery und Data Warehouse Reporting verwendet.

Daten, die in OLAP gespeichert werden, sind in der Regel nicht so kritisch wie die in OLTP gespeicherten. Das liegt daran, dass die meisten Daten simuliert werden können, die aus dem OLTP kommen und dann in Ihre OLAP-Datenbank eingespeist werden können. Diese Daten werden typischerweise für das Bulk-Loading verwendet, das oft für Geschäftsanalysen benötigt wird, die schließlich in visuelle Diagramme gerendert werden. OLAP führt auch multidimensionale Analysen von Geschäftsdaten durch und liefert Ergebnisse, die für komplexe Berechnungen, Trendanalysen oder anspruchsvolle Datenmodellierung verwendet werden können.

OLAP speichert Daten in der Regel persistent in einem spaltenförmigen Format. In MariaDB **ColumnStore** hingegen werden die Datensätze auf Basis ihrer Spalten aufgeschlüsselt und separat in einer Datei gespeichert. Auf diese Weise ist der Datenabruf sehr effizient, da nur die relevante Spalte durchsucht wird, auf die sich Ihre SELECT-Anweisung in der Abfrage bezieht.

Man kann sich das so vorstellen: Die OLTP-Verarbeitung kümmert sich um die täglichen und wichtigen Datentransaktionen, mit denen Ihre Geschäftsanwendung läuft, während OLAP hilft, das Produkt zu verwalten, vorherzusagen, zu analysieren und besser zu vermarkten - die Bausteine, die eine Geschäftsanwendung ausmachen.[[3]](#footnote-3)

## Spaltenorientierter Datenspeicher

Wenn eine Datenbank auf Millionen von Datensätzen anwächst, die über viele Tabellen verteilt sind, und Business Intelligence/Wissenschaft zur vorherrschenden Anwendungsdomäne wird, ist ein spaltenorientiertes Datenbankmanagementsystem gefragt. Im Gegensatz zu traditionellen Zeilenspeichern wie MySQL und PostgreSQL bietet ein Spaltenspeicher eine moderne und skalierbare Lösung, die keine großen Hardware-Investitionen erfordert.[[4]](#footnote-4)

Ein spaltenorientiertes DBMS (oder spaltenartiges Datenbankmanagementsystem) ist ein Datenbankmanagementsystem (DBMS), das Datentabellen spaltenweise und nicht zeilenweise speichert. Die praktische Verwendung eines Spaltenspeichers gegenüber einem Zeilenspeicher unterscheidet sich in der relationalen DBMS-Welt nur wenig. Sowohl spalten- als auch zeilenbasierte Datenbanken können herkömmliche Datenbankabfragesprachen wie SQL verwenden, um Daten zu laden und Abfragen durchzuführen. Sowohl Zeilen- als auch Spaltendatenbanken können zum Rückgrat in einem System werden, um Daten für gängige Extraktions-, Transformations-, Lade- (ETL) und Datenvisualisierungstools bereitzustellen. Durch die Speicherung von Daten in Spalten statt in Zeilen kann die Datenbank jedoch präziser auf die Daten zugreifen, die sie zur Beantwortung einer Abfrage benötigt, anstatt unerwünschte Daten in Zeilen zu scannen und zu verwerfen.

Kurzum, spaltenorientierte Datenspeicher sind effizienter, wenn nicht alle Eigenschaften einer Instanz gebraucht werden, während zeilenbasierte Datenspeicher effizienter sind, wenn (fast) alle Eigenschaften einer Instanz gebraucht werden.

Vergleiche zwischen zeilenorientierten und spaltenorientierten Systemen konzentrieren sich typischerweise vor allem auf die Effizienz des Festplattenzugriffs, der im Vergleich zu anderen Operationen des Computers erhebliche Zeit verbraucht. Spaltenorientierte Systeme sind effizienter, wenn ein Aggregat über viele Zeilen, aber nur wenige Spalten gebildet werden muss, da man dann im Gegensatz zum zeilenorientierten System nur diese und nicht alle Spalten lesen muss. Zeilenorientierte Systeme sind effizienter, wenn gleichzeitig viele Spalten einer einzigen Zeile benötigt werden und wenn die Zeilenbreite sehr groß ist, da dann die ganze Zeile mit einem einzigen Plattenzugriff gelesen werden kann.[[5]](#footnote-5)

## MonetDB, InnoDB und MariaDB ColumnStore

**MonetDB** ist ein freies, spaltenorientiertes relationales Datenbankverwaltungssystem und gleichzeitig auch die Datenbank-Engine. Es ist ein Open-Source-Projekt, das seine Ursprünge am Centrum Wiskunde & Informatica in Amsterdam hat. MonetDB eignet sich hervorragend für Anwendungen, bei denen das Hot-Set der Datenbank - der Teil, der tatsächlich berührt wird - weitgehend im Hauptspeicher gehalten werden kann oder bei denen einige wenige Spalten einer breiten relationalen Tabelle ausreichen, um einzelne Abfragen zu bearbeiten. MonetDB wurde stark von den wissenschaftlichen Experimenten zum Verständnis des Zusammenspiels zwischen Algorithmen und Anwendungsanforderungen beeinflusst. Es hat MonetDB zu einem erweiterbaren Datenbanksystem mit Haken auf allen Ebenen im Software-Stack gemacht.[[6]](#footnote-6)

**InnoDB** ist ein freies, zeilenbasiertes Speichersubsystem (Datenbank-Engine) für das Datenbankmanagementsystem MySQL. Sein Hauptvorteil gegenüber anderen Speichersubsystemen des relationalen Datenbankmanagementsystems (RDBMS) MySQL ist, dass Transaktionssicherheit und referenzielle Integrität über Fremdschlüssel gewährleistet werden.[[7]](#footnote-7) Es ist in den meisten von MySQL vertriebenen Binärdateien als Standard gesetzt. InnoDB ist auch in MariaDB vorhanden, allerdings handelt es sich mittlerweile um eine so stark modifizierte Version, dass MariaDB von einer eigenständigen Version spricht.

**MariaDB** ist ein von der Community entwickelter, kommerziell unterstützter Fork von MySQL, der als freie und quelloffene Software unter der GNU General Public License betrieben werden soll. Die Entwicklung wird von einigen der ursprünglichen Entwickler von MySQL geleitet, die es aufgrund von Bedenken wegen der Übernahme durch die Oracle Corporation im Jahr 2009 abgezweigt haben. Die MariaDB Corporation hat am 5. April 2016 die Veröffentlichung ihrer ersten Big-Data-Analyse-Engine, **MariaDB ColumnStore**, angekündigt. Sie basiert sowohl auf einem Fork von InfiniDB als auch auf Open-Source-Community-Beiträgen.[[8]](#footnote-8) MariaDB ColumnStore ist eine spaltenbasierte Speicher-Engine, die eine massiv parallele verteilte Datenarchitektur nutzt. Es handelt sich um ein spaltenförmiges Speichersystem, das durch Portierung von InfiniDB 4.6.7 auf MariaDB erstellt und unter der GPL-Lizenz veröffentlicht wurde.[[9]](#footnote-9)

# Installation

Für die Installation der beiden DBMS MonetDB und MariaDB empfiehlt sich die Erstellung einer virtuellen Maschine, unter der eine Distribution von Linux läuft. Das hat den Grund, dass die meisten DBMS für Linux konzipiert sind. MacOS und vor allem Windows sind oft zweitrangig in der Entwicklung. So gibt es MariaDB ColumnStore nicht unter Windows, während InnoDB und MonetDB ohne Probleme unter Windows lauffähig sind (die Installation und Inbetriebnahme von MonetDB unter Windows wurde auch getestet). Ein weiterer Vorteil bei der Verwendung einer virtuellen Maschine neben der zentralen Verwaltung aller DBMS ist die Festlegung der zur Verfügung stehenden Ressourcen (CPU, RAM). Somit ist sichergestellt, dass ein objektiver Vergleich möglich ist. Empfohlen sind die Distributionen **CentOS** & RHEL sowie Ubuntu & Debian, die sowohl MonetDB als auch MariaDB als mögliche Plattformen angeben. In Rahmen dieser Arbeit wurde mit CentOS 8 gearbeitet. Die Installation verlief problemlos. Es gibt auch CentOS Stream, doch dabei handelt es sich um eine Quasi-Beta von CentOS, weshalb lieber CentOS 8 verwendet werden sollte.

Um eine virtuelle Maschine aufzusetzen, braucht es eine – wer hätte das gedacht – Virtualisierungssoftware. Die verbreitetsten kostenlosen Anwendungen für Desktop-Systeme sind VirtualBox von Oracle und VMware Workstation von VMware, Inc.

Bei der Durchführung dieser Arbeit auf einem Windows-Rechner kam es immer wieder zu technischen Problemen mit VirtualBox von Oracle, weshalb ich eher zu VMware raten würde. Das muss aber nicht heißen, dass VirtualBox die schlechtere Wahl ist – es hat einfach nur nicht auf meinem System funktioniert. Letztlich ist es aber egal, welche Virtualisierungssoftware man verwendet. Der folgende Guide orientiert sich an Windows als Hostsystem und VMware Workstation als Virtualisierungssoftware.

## Installation einer virtuellen Maschine und Betriebssystem

Für die Installation der kompletten virtuellen Maschine wird benötigt:

1. VMware Workstation Player ([Download](https://www.vmware.com/products/workstation-player/workstation-player-evaluation.html)) installieren
2. CentOS ISO Datei ([Download](https://www.centos.org/download/))
3. VMware starten und diesem [Guide](https://cloudlinuxtech.com/install-centos-in-vmware/) folgen
   1. Als Hardware Requirements sollten 2 Kerne der CPU, 8 GB RAM und 20GB fester Plattenspeicher reichen.
   2. Bei Installation auf korrekte Sprache, Zeitzone und Tastaturlayout achten
   3. Login als root mit Passwort root (der Einfachheit halber) festlegen für notwendige Berechtigungen später
4. Auf CentOS als user root einloggen (eventuell vorher System neustarten und auf ‚Nicht dieser User?‘ klicken)
5. Die Datei flights.csv von [Kaggle](https://www.kaggle.com/bingecode/us-national-flight-data-2015-2020) runterladen, entzippen und in den Home Folder legen
6. Installation von MonetDB und MariaDB vornehmen (siehe nächste Kapitel)

## Installation MonetDB

1. Den Anweisungen für die Installation von MonetDB unter CentOS [hier](https://www.monetdb.org/downloads/epel/) folgen, konkret die Schritte:
   1. yum install <https://dev.monetdb.org/downloads/epel/MonetDB-release-epel.noarch.rpm>
   2. yum install MonetDB-SQL-server5 MonetDB-client
2. Danach die Anweisungen aus [GitHub](https://github.com/BingeCode/columnstore-innodb-monetdb/blob/main/SQL%20code/monetdb.sql) mit STRG + A komplett markieren, mit STRG + C kopieren und anschließend im Terminal in CentOS auf einen Schlag ausführen (Das Passwort für MonetDB ist standardmäßig monetdb)
3. Durch das Skript wurde der MonetDB Server (Daemon) gestartet, sich mit dem Server verbunden, eine Datenbank angelegt, in der die Tabelle monetdb angelegt wird und anschließend die komplette CSV-Datei eingespeist. Danach ist MonetDB bereit für verschiedene [Queries](https://github.com/BingeCode/columnstore-innodb-monetdb/blob/main/SQL%20code/queries.sql).
4. Beachte, dass das Importieren von der CSV-Datei dauern kann. Als Referenz lohnt sich der Blick auf das [README](https://github.com/BingeCode/columnstore-innodb-monetdb#csv-import-timings).

## Installation MariaDB

1. Den Anweisungen für die Installation von MariaDB unter CentOS [hier](https://mariadb.com/docs/deploy/community-single-columnstore-cs105-centos8/) folgen.
   * Nicht alle Schritte (Optimierung der Kernel Parameter, das Linux Sicherheitsmodul deaktivieren, die Buchstabenkodierung konfigurieren) sind wahrscheinlich notwendig, aber sollten der Sicherheit halber durchgeführt werden. Die tatsächliche Installation beginnt allerdings erst ab [hier](https://mariadb.com/docs/deploy/community-single-columnstore-cs105-centos8/#columnstore-installation). Die folgenden Abschnitte nach ‚ColumnStore Installation‘ sind optional.
2. Das Starten der relevanten MariaDB-Prozesse (Server) wird ab [hier](https://mariadb.com/docs/deploy/community-single-columnstore-cs105-centos8/#start-the-columnstore-processes) beschrieben.
3. Anschließend analog zu Punkt 2) unter 2.2 Installation MonetDB die Anweisungen aus [GitHub](https://github.com/BingeCode/columnstore-innodb-monetdb/blob/main/SQL%20code/columnstore.sql) im Terminal ausführen
4. Das Importieren der CSV-Datei sollte recht schnell gehen, jedenfalls deutlich schneller als bei MonetDB (siehe [README](https://github.com/BingeCode/columnstore-innodb-monetdb#csv-import-timings)).

## Installation InnoDB

InnoDB muss nicht extra installiert werden, da es standardmäßiger Teil von MariaDB ist. Sofern MariaDB also installiert wurde, kann InnoDB auch verwendet werden. Um eine InnoDB-Tabelle zu verwenden, müssen einfach die Schritte in der entsprechenden [Innodb.sql](https://github.com/BingeCode/columnstore-innodb-monetdb/blob/main/SQL%20code/innodb.sql)-Datei ausgeführt werden.

Beachte, dass das Laden der CSV-Datei in eine InnoDB-Tabelle deutlich länger dauert als bei MonetDB oder MariaDB. Für Referenzwerte schaue auf die [README](https://github.com/BingeCode/columnstore-innodb-monetdb#csv-import-timings).

# Bedienung & Resultate

Grundsätzlich lassen sich alle drei DB-Engines, bzw. zwei DBMS, gleich bedienen. Sie alle können über den Terminal mit SQL angesprochen werden. Allerdings hat sich im Rahmen meiner Tests gezeigt, dass MonetDB deutlich flexibler in der Benutzung von SQL sowie deutlich performanter als MariaDB ColumnStore ist. Um die Engines untereinander zu vergleichen, hatte ich das Ziel, Abfragen mit zunehmender Komplexität zu stellen und die Performance anhand derer zu vergleichen. Doch leider stellte sich heraus, dass z.B. Subqueries wie WITH SOME\_TABLE AS (SELECT COLUMN FROM TABLE) … oder IN (SELECT …) bei ColumnStore standardmäßig nicht möglich sind. Es gibt zwar [Konfigurationseinstellungen](https://mariadb.com/kb/en/columnstore-system-variables/#operating-mode), die man vornehmen kann, dass ColumnStore auch zeilenorientierte Abfragen versteht, allerdings funktionierten diese Maßnahmen auch nach extensiver Recherche nicht auf meinem System, zu meinem Bedauern. Unter [queries.sql](https://github.com/BingeCode/columnstore-innodb-monetdb/blob/main/SQL%20code/queries.sql) finden sich ein paar Versuche, komplexere Abfragen zu stellen, die allesamt allerdings nur unter MonetDB laufen.

InnoDB versteht alles der komplexeren Abfragen, die ColumnStore nicht versteht, wurde aber aufgrund der langen Query Timings von 10m aufwärts pro Query nicht weiter geprüft.

Auch erwähnenswert sind die Importverfahren der verschiedenen Storage Engines. Unter dem Abschnitt finden sich die Ergebnisse der verschiedenen Verfahren. Hier wird deutlich, dass die spaltenbasierten Speicher einen großen Vorteil hinsichtlich der Importgeschwindigkeit haben. Besonders ColumnStore punktet mit seinem [cpimport](https://mariadb.com/docs/solutions/columnstore/load-columnstore-data/#cpimport) Verfahren.

Die Installation der beiden DBMS dauert ungefähr gleich lang, wenn man bei der Installation von ColumnStore die optionalen Schritte vor und nach der tatsächlichen Installation weglässt. Die Installation beider Systeme ist an sich relativ einfach, wird allerdings durch das vorherige Aufsetzen einer virtuellen Maschine mitsamt Betriebssystem erschwert. MonetDB punktet dabei im Vergleich zu ColumnStore nicht nur durch Performanz, sondern auch durch die Tatsache, dass es auf Windows läuft und somit keiner virtuellen Maschine bedarf.

Die Resultate sind auch nochmal in der [README](https://github.com/BingeCode/columnstore-innodb-monetdb#performance) auf GitHub zu finden.

Weiterhin gibt es weitreichende Möglichkeiten, um die Performanz der einzelnen Engines zu steigern. Vor allem bei MariaDB Columnstore lassen sich zahlreiche Parameter optimieren, um die [Performanz zu steigern](https://mariadb.com/kb/en/columnstore-performance-tuning/). Auch bei InnoDB könnte man mit dem Einsatz von Indizes die Performanz in bestimmten Fällen steigern. Auch MonetDB bietet [viele Möglichkeiten](https://www.monetdb.org/Documentation/Tutorial/AdvancedFeatures), um das System besser auf bestimmte Bedingungen einzustellen, kommt aber schon out-of-the-box mit zahlreichen Optimierungen auch zur Laufzeit daher.

# Fazit und Auswertung

Für eine abschließende Bewertung ist wichtig festzuhalten, dass es sich bei diesem Vergleich um ein sehr spezifisches und wahrscheinlich unrealistisches Szenario handelt. Nicht nur handelt es sich beim Datensatz von 2.7GB um eine winzige Datenmenge im Vergleich zu den üblichen Terabyte in Big Data Anwendungen, sondern lässt auch die eingesetzte Hardware in der virtuellen Maschine kein Datenbank-Administratoren-Herz höherschlagen. Nichtsdestotrotz wurden durch diese Vergleichsarbeit deutliche Tendenzen klar, nämlich die Überlegenheit von spaltenorientierten Datenbanken bei Aggregationsabfragen und die Überlegenheit von zeilenorientierten Datenbanken bei einfachen Reads über die komplette Zeile. Wie im Abschnitt 1.2 OLTP vs. OLAP erwähnt wurde, sind spaltenorientierte Datenbanken auch oft als Erweiterung von zeilenbasierten (OLTP) Datenbanken gedacht. Das reguläre Geschäft wird mittels OLTP abgewickelt und das Big Data/Reporting/Data Mining Geschäft über OLAP.

Insgesamt jedoch steht MonetDB in diesem Szenario als deutlicher Gewinner da mit einer drastisch kürzeren Query Time als die anderen beiden DB, einer akzeptablen Importgeschwindigkeit und einer größeren Flexibilität in der Gestaltung von Abfragen als bei ColumnStore. ColumnStore erfreut sich allerdings deutlich größerer Beliebtheit in der Community. Guter Community-Support ist bei Open Source Projekten essenziell, was wohl ein großer Grund für die größere Beliebtheit von ColumnStore ist, wenn auch MonetDB in technischer Hinsicht vielleicht das performantere System ist.

Letztlich stellt sich dennoch die Frage, warum MonetDB nicht bekannter ist. Laut dieser[[10]](#footnote-10) Quelle steht MonetDB nur auf Platz 120 aller DBMS sowie auf Platz 60 aller relationalen DBMS, während MariaDB auf Platz 12 und 8 respektive steht. Auch nach extensiver Internet-Recherche habe ich kaum Erfahrungsberichte aus dem Produktiv-Einsatz von MonetDB gefunden. Wie die offizielle Website allerdings beschreibt, handelt es sich bei MonetDB um ein DBMS, das primär in Forschungsprojekten zum Einsatz kommt.[[11]](#footnote-11) Dennoch weist die jüngste Entwicklung in den letzten Jahren darauf hin, dass MonetDB auch im kommerziellen Sektor zunehmend an Bedeutung gewinnen könnte.

Schließlich lohnt sich noch der Blick auf einen ähnlichen Vergleichsversuch von [ReportServer](https://reportserver.net/blog/2016/06/20/mariadb-columnstore-vs-innodb-vs-monetdb/) bei dem ebenfalls die Performanz dieser drei Datenbank-Engines in verschiedenen Szenarien untersucht wurde. Bei dieser Untersuchung kam es zu ähnlichen Ergebnissen, wenn es auch ein paar wenige Szenarien gab, in denen ColumnStore MonetDB überlegen war. Die Zukunft beider Open-Source-Projekte sollte also weiterhin mit Spannung beobachtet werden und der Einsatz verschiedener DB-Engines im Einzelfall abgewägt werden, auch wenn sich schon jetzt im Vergleich zu zeilenbasierten Engines signifikante Resultate erreichen lassen. Andere interessante spaltenbasierte Speicher wie Clickhouse, Apache Spark usw. sollten natürlich auch Beachtung erhalten.

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)