universität wien

MASTERARBEIT | MASTER'S THESIS

Titel | Title

Enhancing Healthcare Data Processing: Leveraging Hadoop and Hive for Querying Big Data in Data Warehousing Solutions

verfasst von | submitted by

Tobias Fühles B.Sc.

angestrebter akademischer Grad | in partial fulfilment of the requirements for the degree of

Master of Science (MSc)

Wien | Vienna, 2024

Studienkennzahl lt. Studienblatt | Degree programme code as it appears on the student record sheet:	UA 066 926
Studienrichtung lt. Studienblatt | Degree programme as it appears on the student record sheet:	Masterstudium Wirtschaftsinformatik
Betreut von | Supervisor:	Univ.-Prof. Dipl.-Math. Dr. Peter Reichl Privatdoz.
Mitbetreut von | Co-Supervisor:	Dr.techn. Florian Güldenpfennig B.Sc.

Abstract

Continuous developments in interoperability, cloud computing and hardware development have increased the amount of data generated and processed in many industries, including healthcare. Historically grown relational databases offer the possibility of supporting strategic decisions and patient-oriented analyses in the healthcare sector. However, since the implementation of such optimizations depends heavily on the efficiency and speed of the required data queries, relational databases reach their limits depending on hardware, data volume and data complexity. This work examines the improved analysis possibilities through the additional implementation of a Hadoop-based database alongside existing relational database solutions. For a meaningful comparison, both database solutions are compared using the benchmarking tool TPC-DS within the cloud computing platform Azure and the results are critically evaluated in terms of opportunities and obstacles for healthcare institutions and recommendations for possible implementations are given.

Kurzfassung

Die kontinuierlichen Entwicklungen in den Bereichen Interoperabilität, Cloud Computing und Hardwareentwicklung haben die Menge der generierten und verarbeiteten Daten in vielen Branchen erhöht, so auch im Gesundheitswesen. Historisch gewachsene relationale Datenbanken bieten die Möglichkeit strategische Entscheidungen und patientenorientierte Analysen im Gesundheitswesen zu unterstützen. Da eine Implementierung solcher Optimierungen allerdings stark von der Effizienz und Geschwindigkeit der benötigten Datenabfragen abhängt, stoßen relationale Datenbanken abhängig von Hardware, Datenvolumen und Datenkomplexität an ihre Grenzen. Diese Arbeit untersucht die verbesserten Analysemöglichkeiten durch die zusätzliche Implementierung einer auf Hadoop basierenden Datenbank, neben bestehenden relationalen Datenbanklösungen. Für einen aussagekräftigen Vergleich werden beide Datenbanklösungen mit Hilfe des Benchmarking Tools TPC-DS innerhalb der Cloud Computing Platform Azure verglichen und die Ergebnisse hinsichtlich Möglichkeiten und Hindernissen für Institutionen des Gesundheitswesen kritisch evaluiert sowie Empfehlungen für mögliche Implementierungen gegeben.

Contents

List of Figures

Listings

1. Introduction

Continuous developments in interoperability, cloud computing and hardware development increased the generated and processed amount of data in many industries, especially in the healthcare sector. Even before the increased focus on "Big Data" over the last decade, healthcare institutions set up data warehouses to generate strategic value - not of the data warehouse itself, but the knowledge derived from the data warehouse and the application of that knowledge to achieve overall improvements [DH05]. The definition of a data warehouse as copy of transaction structured for query and analysis can be applied to clinical laboratory, medication orders, textual documents (e.g., discharge summaries, radiology reports, surgical pathology reports) and general administrative data [Kim96, LSH08].

Efficient data management, querying and retrieval are critical for supporting clinical decision-making, research and patient care [EL14, RC18]. Those requirements also make the foundation for any third party application in the health sector domain. In addition to the big data characteristics, data searching, storage and analysis a promising software platform to develop suitable applications is the open-source distributed data processing platform Apache Hadoop [RC18]. Hadoop is a software framework for distributed storage and processing of large datasets, making use of parallelized execution on large clusters of hardware [DG04]. However Hadoop's "schema on read" model stores data in a raw and semi structured format in contrast to to structured "schema on write" model used in traditional data warehouses. To tackle this issue open-source database systems such as HBase and Hive have been developed and built on top of Hadoop. To efficiently process the legacy data of existing relational data warehouses, the data is (1) loaded via ETL (Extract, Transform, Load) pipelines in a format suitable for Hadoop and Hive, (2) stored in Hadoop Distributed File System (HDFS) and (3) queried via Hive tables pointing to the HDFS with the inherent SQL-like syntax [TSJ+09]. Utilizing the advantages of Hadoop, such as expanding horizontal scalability for extensive datasets through the addition of inexpensive hardware, thus enhancing cost efficiency, fault tolerance, and availability, alongside the provision of robust tools and libraries for machine learning, graph processing, and complex event handling, is expected to boost the analytical capability and utilization of current extensive legacy datasets.

As this master thesis aims to compare the querying performance of traditional Relational Database Management Systems (RDBMS) and Hadoop-based databases, a reliable and proven benchmark is required to gain meaningful results. Despite creating an individual healthcare-specific benchmark, there exist other well-established and technically sophisticated tools provided by the Transaction Processing Performance Council (TPC) [NP06]. The general characteristics of healthcare data, formed by the broader design of overall digitization and object-oriented programming, allows the application of data

schema and formats of other domains to be used as a benchmark in the health sector [HAK91]. Therefore, applying benchmarking using non-healthcare domain data still results in valuable suggestions and implications through the compatibility of benchmarking methodologies across sectors, ensuring valuable insights in diverse contexts. This thesis applies the Decision Support Benchmark TPC-DS, which models several generally applicable aspects of decision support systems and provides a representative evaluation of database performances, especially for Big Data Systems [tpc]. In the pursuit of this aim, this thesis will implement and thoroughly document the specific configurations for both database systems during the execution of the TPC-DS benchmarking process. Subsequently, it will critically evaluate the approach, in terms of effectiveness as well as limitations and providing valuable insights for future research endeavors in the domain.

Finally, this master thesis will make a recommendation for which organizations in the healthcare sector, depending on data volume, diversity, and quality, an extension of the existing traditional RDBMS capacities by a Hadoop-supported database for the efficient implementation of Big Data Analysis presents a worthwhile investment.

1.1. Research Questions

The research questions guiding this study include:

1. How does the querying performance of traditional RDBMS compare with Hadoop-based databases in the healthcare sector?
2. What are the implications of utilizing benchmarking tools such as the Decision Support Benchmark TPC-DS for evaluating database performance in healthcare settings?
3. What insights can be gained from the comparison of database performance evaluation between traditional RDBMS and Hadoop-based databases for future research endeavors in the healthcare domain?
4. Based on data volume, diversity, and quality, which organizations in the healthcare sector would benefit from extending traditional RDBMS capacities with Hadoop-supported databases for efficient Big Data Analysis implementation?

1.2. Structure of Thesis

This thesis is structured as follows. First, a review of related work is presented. This comprises information and research on RDBMS, providing insight into the respective limitations and challenges in handling different volumes of healthcare data. Additionally, an exploration of the background of Hadoop is conducted, examining its architecture, benefits, and challenges in the context of healthcare data management.

Second, an examination of the methodologies employed in this study is provided. Different benchmarking methods are considered to evaluate the performance of both

traditional relational databases and Hadoop-based solutions. Subsequently, the TPC-DS benchmarking process and its application in the healthcare domain are discussed. Detailed configurations and details for executing the TPC-DS benchmark on both database systems are also elaborated upon.

Following the methodology, implementation details and comparative querying results are presented. The databases utilized in the study are described, along with the processes of data import, indexing, query translation, and execution within the Azure cloud computing platform. Comparative analysis of query performance is provided, highlighting the strengths and limitations of each database system. Additionally, any resource limitations encountered during the implementation phase are discussed.

In the subsequent section, a comprehensive discussion and conclusions are drawn based on the findings. An analysis of the implications of data volume and complexity on database performance in healthcare settings is conducted. Moreover, recommendations for healthcare organizations are provided based on the research insights.

2. Relational Databases in Healthcare

Relational Databases have been used by various health sector facilities from hospitals to governmental institutions. Aside from storing the data, the most important aspects consist of supporting decision-making activities and ad-hoc exploration of large data volumes. Berndt et al. propose a three level data pyramid to support all involved parties relying on the existing data, including database administrators and management decision makers 2.1.

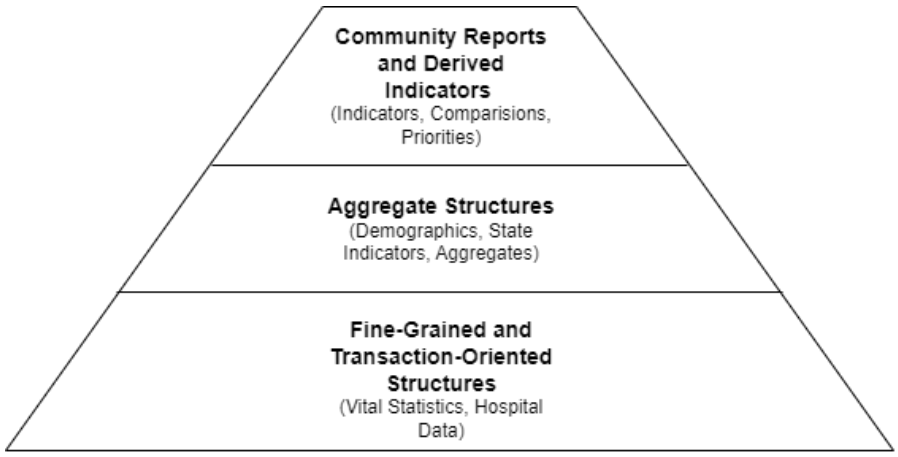

Figure 2.1.: Levels of Data Warehousing in Healthcare [BFHS02]

Each data layer presents a different level of granularity and aggregation. The top layer shows highly aggregated data to support management decision making assuring a fast responsive with web-based interactive dashboards or data browsing tools. The middle layer is supported by dimension tables added with different data modeling techniques as described in the next section. This data can be accessed by databases editors and all tools that support online analytical processing (OLAP) techniques. The base layer retains the finest data granularity, possibly on event-level with data obtainment from medical machinery, Electronic Health Records (EHR) and other institution specific administrative data [BFHS02]. This level collects data from computer-based patient record software and personal computer hardware which is gaining importance and volume with the technological advancements [PLG+97]. As early as 2002 relational databases were also extended to support content-based retrieval of medical images, creating a storing option for the varieties of medical imaging techniques [ATT+02]. Regardless of the originating system a relational database sets data requirements in form of data modeling, schemas and normalization principles [LSC+09]. To gain further insights on the stored data those

aspects will be discussed in the following sections.

2.1. General Aspects of Healthcare Data

Healthcare Databases mainly consist of Electronic Medical Records (EMRs) and Electronic Health Records (EHRs) from various healthcare institutions such as pharmacies and general practitioners. Data dependent organizations collect different data from multiple institutions and aggregate it into a central system, historically typically an Enterprise Data Warehouse (EDW), to enable data analysis and transformation. However, the Healthcare sector generates a variety of different data and formats, e.g. text, digital, images, videos, or multimedia with corresponding structured, semistructured, or unstructured formats [TBTK19]. The need for different data formats is illustrated by a general radiology, which uses various image types, and even paper medical records. A prominent biomedical database is MEDLINE containing over 24 million records, around 122.3 GB of data [AAS+15]. Another challenging aspect in health data is identical data stored in multiple machines and different formats, e.g. claims data vs. clinical data [TBTK19]. With RDBMS being the de-facto standard of storing healthcare data, especially of historically grown systems, a high degree of relationships, which is inherent to health data, is hard to depict with RDBMS. Healthcare data is also known to grow 40 percent annually, due to an increase in storage capacity and processing hardware. To address the challenges presented by data growth, format variety and high degree of relationships various suggestions have been made to migrate existing databases to NoSQL or graph databases, optional in in a cloud environment. However this aspect will not be further discussed in this thesis as the focus lies in querying large existing databases more efficiently through the use of Hadoop databases, while it is still important to keep those aspects in mind to set the general domain context .

2.1.1. Big Data in Healthcare

Digitized workflows and processes with the associated increase in data volume introduced possibilities and challenges of Big Data to the health sector. Senthilkumar et al. define Big data as a large volume of high velocity, complex and variable data which requires advanced techniques and technologies for capturing, storing, distributing, managing and analyzing information [SRM+18]. The most relevant sources of Big Data in healthcare are A) Machine Generated Data, e.g. sensors, wearable devices and other healthcare vital statistics measuring devices B) Biometric Data, consists of human individual data such as finger prints, genetics, signature, retinal scans, heart rate, blood pressure, pulse and pulse-oximetry readings as well as x-ray and other medical images C) Human Generated Data, data generated by human beings in the healthcare system. These type of data include unstructured and semi structured clinical data such as case notes, laboratory results, hospital admission records, discharge summaries and electronic mails. Human generated data also include structured Electronic Health Record (EHR) data. D) Transactional Data, includes data from healthcare claims and billing records. E) Behavioural Data,

includes data generated from social interactions and communication tools such as websites and social media sites such as Twitter and Facebook. F) Epidemiological Data, includes vital statistical data, health surveys and disease registries. G) Publication Data, these include data from clinical researches and medical reference materials. Big data in the context of healthcare can also be seen as a collection of tools, technologies, methods and procedures which are used to create, store, process, analyze and retrieve large sets of electronic health data in an efficient manner [OO16b]. Those aspects are addressed and improved by an introduction of Hadoop databases compared to RDBMS. Alongside the data sources, [MSJK17, RZL+18, OO16a] also discuss the characteristics of Big Data in Healthcare. Additional to the typical characteristics volume, variety and velocity, this thesis will discuss variability and veracity to address the change in the nature of healthcare data.

In the following, the 5 V's are defined in their terminology. A) Volume: Volume refers to the amount of data generated by the stakeholders in the healthcare industries. Healthcare data are usually complex, noisy, heterogeneous, longitudinal and voluminous [MSJK17]. For instance in 2011, the data in the United States of American healthcare system was about 150 Exabyte [GSB15]. Furthermore, in 2012, digital healthcare data worldwide was estimated to be equivalent to 500 Petabytes and it is expected to reach 25,000 Petabytes by 2020 [MSJK17]. B) Velocity defines the speed additional healthcare data is stored, analyzed, visualized and exchanged amongst healthcare providers. The healthcare sector depends on a seamless, secured and meaningful exchange of health information for effective and efficient patient care as patients usually receive care from various healthcare providers at different geographical locations. This information is required to be accessed by healthcare practitioners in a real time manner for decision making [OSGO13]. An error or omission in this process can lead to the untimely death of a patient. Hence, velocity is required in healthcare because it allows healthcare providers to exchange and use data in a timely manner. C) Variability indicates to changes within the data format, rate, semantics and structure. This aspect is especially important in the healthcare sector with its vast of different potential formats such as image files, audio files, flat files, relational tables and texts. The domain is also prone to words and abbreviations with different meanings. For example, APC means Activated Protein C but in other literature AlloPhyCocyanin and Antibody Producing Cells [OO15]. D) Variety presents the fourth characteristic and describes the diversity of the data sources in the healthcare environment which can be unstructured, semistructured or structured. Data sources originate despite EHRs and MHRs from blogs, web server logs, social media up to geographical data. Additionally data is often stored in multiple different systems which can be legacy or application systems, various databases and transaction processing software. [GSB15] The last V is E) Veracity referring to the produced data quality. The various data sources and formats present a serious issue regarding the generation of high quality data. Unstructured data can also be composed of different encoding formats, languages and the discussed ambiguity of medical terms and abbreviations. Low quality data results in higher cost and error rates and should be reduced at all costs as also patients lifes could be at risk. Hence, healthcare data "must be relevant, reliable and error free [GSB15]" [OO15].

2.2. Relational Data Modeling

Data Modeling was initially established to provide a possibility to specify the structures of data in actual file systems along with database management systems (DBMSs). First network and hierarchical models have been introduced followed by the now de facto standard of the relational model as mathematical basis for data analysis and modeling. Relational modeling provides data independence and addresses a variety of common database problems such as integrity constraints, security schemes, data distribution and data replication [Nav92, Cod70].

The relational model forms a collection of relations, a relation can be thought of a table of values with each row representing a collection of related data values. Those rows represent a domain entity or relationship. The following vocabulary or concepts supports a broader understanding of relational modeling in DBMS: (1) Attributes are relation defining properties e.g. "ID", "name". Those relations are saved in the (2) table format consisting of rows and columns, representing records and attributes. (3) Tuples describe single table rows containing a single record. (4) Relation schema represent relation names and the corresponding attributes, the total number of attributes defines the (5) degree of the relation. Furthermore (6) Cardinality is defined by the total number of rows in a table, a (7) relation instance is a unique finite set of tuples in the database and every table row has a unique/non-unique (8) relation key consisting of one or up to multiple attributes. SQL is also commonly associated with the relational model but should not be taken synonymous. Structured Query Language (SQL) became the standard programming language for the interaction with relational database management systems after its introduction by IBM in the 1970s [IBMb]. SQL is a declarative language which are nonprocedural - "what should be done" in opposition to "how it should be done" - and enables database operations such as a) creating, replacing, altering, and dropping objects, b) inserting, updating, and deleting table rows, c) querying data, d) controlling access to the database and e) guaranteeing database consistency and integrity [Ora].

2.3. Schema and Modeling

Before implementing a relational database it is important to gather all available knowledge from domain or application experts to create a description of the database. This description is refereed to as the database schema designed for given data requirements. A specific data model is used for description in form of a language syntax or diagrammatic conventions. Schemas typically remain stable during the lifetime of a database but may also be changed referred to as schema evolution. An established schema is defined and applied to the relational data using SQL [Nav92]. Today many applications offer different interfaces for creating database schemas, e.g. by translating Entity Relationship diagrams to SQL code.

2.3.1. Schema Types

Two of the most popular relational data modeling approaches are Entity-Relational and Dimensional Modeling. While Entity-Relational Modeling, based in Online Analytical

Processing (OLAP) using complex queries to analyze aggregated historical transaction data, uses a conceptual ER design which is translated into a relational schema, Dimensional Modeling consists of a fact table with several respective dimension tables [Kim96].

A relational schema is the standard database schema used in RDBMS following the principles of normalization to minimize redundancy and ensure data integrity. The data is organized into a set of related tables based on the initial ER model, the relationships are usually maintained by foreign key constraints. The schema may have multiple tables, with each table representing a specify entity or concept with a high level of relationship complexity among themselves [SS05].

Dimension tables have a single attribute unique relation key, in the following primary key, corresponding exactly one to one to an attribute of the multi-attribute key of the fact table. The graphic representation of an inner fact table with multiple outer dimension tables is called the star schema. By removing low cardinality attributes in the dimensions tables with separate dimension tables, linked to the origin dimension table with surrogate keys the schema is extended to a "snowflake" [Kim96, SS05].

Besides the relational and star schema being the most famous schemas a variety of other schemas exist, also representing hybrid or extended approaches. As this extends the scope of this thesis some relevant remaining schemas will be briefly outlined in the following. The fact constellation schema consists of multiple facts tables sharing dimension tables. The same applies for the galaxy schema but in opposition to the fact constellation schema the fact tables do not need to be directly related. The naming for the galaxy schema results in the graphical view as a collection of stars [SA14].

Many healthcare third party applications and connected devices are set to the most appropriate of the described data schema. However many processes in the health sector involve repetitive patient visits or series of outcomes with a different modeling not as well understood as the standard industry environments [PSA05]. A significant portion of health assessment measurements involve a series of visits or outcome measurements, making multiple health assessment measures across multiple patient visits difficult to analyze. A single health measurement application can be modeled by applying dimensional modeling, the star schema. To model multiple patient care episodes multiple star schemas could be applied, but a different level of analysis might be required. For example multilevel analysis provide institutions to link different levels of data analysis, from general-health-level outcomes, to specific disease outcomes, to fine-grained analysis of time series, a level of analysis similar to 3.4. To provide the required multilevel analysis a combination of multiple star schemas, with some containing an entire care episode in comparison to a single visist, as well as transactional, i.e. relational schemas, to enable traditional statistical analysis [PSA05, SA14].

2.4. Limitations of RDBMS

Technological development with an exponential rise in connected devices and services set the stage for a new informatics discipline, Big Data. However relational databases were not originally designed to handle petabytes of data in a non structured format. As this thesis

evaluates the querying performance of structured data, the disadvantages of RDBMS dealing with semi or unstructured data will not be further discussed. Taking those aspects aside RDBMS has still potential downsides when used with extensive amounts of data.

Firstly, scaling is not trivial to achieve and can only be done vertically. Vertical scaling refers to the process of increasing the capacity or performance of a database server by adding more resources to the original server, typically in a more powerful or larger configuration by updating hardware components. But hardware can only be scaled as far as available components allow. So scaling possibilites are limited and get very expensive from a certain threshold. An additional scaling issue consists in the inherent data insertion in the third normal form. Query complexity can not be determined, which is required to extract the desired results from the database. For very complex query the data has potentially to be de-normalized. For query or data sizes exceeding the available memory the system will rely on disk-based operations, which are significantly slower than in-memory operations. This behavior can also impact the general performance of the RDBMS, queries may experience latency resulting in a decreased system performance [med, ESE+22].

The normalization comes with the additinal disadvantage of very complex queries. If a high normalization is present the likelihood of complex queries increases as much data needs to be merged to obtain insights. Those joins and calculations can be complex and may cause the machine to decrease performance significantly. As databases start to grow to petabytes, the performance decrease is likely to be significant [CGK15].

The previous discussed downsides could be tried to be overcome by horizontal scaling or data sharding. Sharding refers to the partitioning of large databases to more manageable pieces called shards. This data can afterwards be distributed over multiple servers or hardware components. As one can imagine, this does not present an easy solution but rather a loss of an RDBMS benefits: a) data present in multiple locations can not efficiently allocate all of this data for analysis, b) complete joins are not possible, c) aggregations are not possible, d) additional indexing must be done for all shards equally and e) data modeling and schema changes must be applied to all shards respectively [dS17, med].

Further challenges exist when using RDBMS in the big data context especially with an increasing amount of semi and unstructured data. However especially the performance downsides when delivering complex and complete queries of large amounts even of structured data make it difficult to apply a higher level of dependent decision processes. The future health sector will heavily rely on querying big data for real-world applications, though needs to overcome the presented challenges. Those circumstances present the main motivation of the comparison of existing RDBMS systems with more suitable big data technologies presented in this thesis.

3. Hadoop

3.1. Hadoop

Apache Hadoop, originally created by Doug Cutting and Mike Caferella in 2005, is an open source framework dealing with distributed computing of large datasets across multiple clusters of hardware using simple programming models. The complete Apache Hadoop software library presents a framework that enables distributed processing of large datasets across multiple clusters of hardware. Hadoop's design aims to scale up from a RDBMS like single server setting to a potential infinite addition of hardware, where each hardware entity offers local computation and storage. A major advantage of this set up is the potential use of inexpensive hardware [Sal12, Mad12]. Hadoop: The Definitive Guide [Whi15] serves as a cornerstone reference for the subsequent sections, offering a comprehensive technical exploration of Hadoop's fundamental features.

3.1.1. MapReduce

MapReduce is a batch query processor or programming model for data processing with the ability to run ad hoc queries against the complete dataset, delivering results in a reasonable amount of time. This promise opens new possibilities for innovative applications and analysis on top of large amounts of data. Previously hidden insights locked by performance issues over the data amount can now be unlocked to gain new insights and raise even more advanced derived questions [Whi15, Sal12].

Hadoop can run MapReduce programs written various languages from Java to Ruby to Python. A main advantage is MapReduce's inherent parallel execution of programs which enables data analysis on a large scale only dependent on the available hardware resources. To benefit from Hadoop's inherent parallelization techniques all queries must be expressed as MapReduce jobs. MapReduce breaks the processing into two different phases: a map and a reduce phase where each phase has its own key-value pairs of input and output of a user defined type. The user also has to specify a map and a reduce function. An input to the map phase can be thought of as an EHR in text input format, as this format can easily be exported from various RDBMS solutions. Each line in the dataset presents an individual text value, the key is the offset of the start of the line to file begin. An example question to be solved could be "Of what age is the oldest recorded patient for each year?". A map function would extract "year" and "age" for the as those are the only relevant fields. In other words, the map functions prepares the data for further assessment by the reduce function. The map function also takes care of missing or bad data values. The following lines can be taken as exemplary input data, unused columns are indicated as periods:

```
0067011990999991950051507004...9999999N23+99999999999...
0043011990999991950051512004...9999999N34+99999999999...
0043011990999991950051518004...9999999N35+99999999999...
0043012650999991949032412004...0500001N80+99999999999...
0043012650999991949032418004...0500001N83+99999999999...
```

Figure 3.1.: Sample Lines Input Data

The above input lines are presented to the map function as key-value pairs:

```
(0, 0067011990999991950051507004...9999999N23+99999999999...)
(11, 0043011990999991950051512004...9999999N34+99999999999...)
(103, 0043011990999991950051518004...9999999N35+99999999999...)
(122, 0043012650999991949032412004...0500001N80+99999999999...)
(758, 0043012650999991949032418004...0500001N83+99999999999...)
```

Figure 3.2.: Input as Key Value Pairs to Map Function

In this example, the map function extracts the year and the ID which are indicated as bold text and gives only those respective lines as output. The output is processed by the MapReduce framework, sorting they key-value pairs by key, before the reduce function is applied. For this example, this would mean the upper three values are sorted to the year 1950 and the lower to the year 1949, creating a list for each year with the corresponding id values. The reduce function now only has to iterate through the list to find the respective maxima. The whole flow can be seen in the following image [Whi15, Mad12].

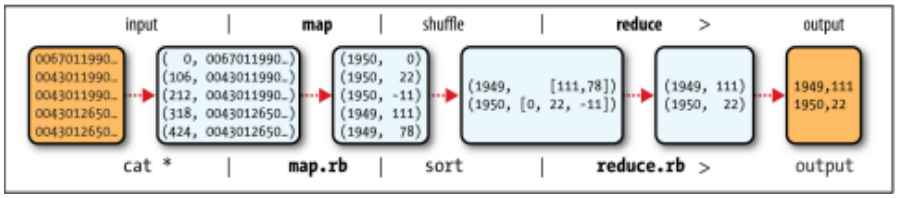

Figure 3.3.: MapReduce Logical Data Flow [Whi15]

3.1.2. Scaling

The above example presents a small input for MapReduce but not for the indented large inputs. Instead of storing files on a local file system the data is stored in a distributed filesystem, typically a Hadoop Distributed File System (HDFS). This distribution enables

to shift the MapReduce computation to the respective data hosting machines, using Hadoop's resource management system YARN [Whi15, hadb].

YARN is an abbreviation for "Yet Another Resource Negotiator" and was a feature of Hadoop 2.0 which was introduced in October 2013. Despite being a resource management, it presents a job schdeduling technology and allocates system resources to various applications running in a Hadoop cluster, coordinating tasks to be executed on various cluster nodes [Whi15, IBMa].

Hadoop splits input into equally sized pieces for a given MapReduce job referred to as input splits. Each split creates a map task, applying the user-defined map function to each record in the split. An important aspect lies in the load balancing of the splitting. Small splits may result in an overhead of managing the splits and map task creation compared to the total job execution time. A suggested best-practice split is given by the size of an HDFS block, which is by default 128 MB. To improve performance it is also beneficial to run the map task on the node where the respective data HDFS is stored due to no wastage of cluster bandwidth. Map tasks store their output to the local disk and not to HDFS because in is only intermediate output processed by the reduce tasks to generate the final output [Whi15, Wil].

Reduce tasks can not benefit from data locality as usually the input of a reduce tasks consist in the output of all or many map tasks. In case of multiple reduce tasks the map tasks partition their output where each map tasks creates an individual partition for each reduce task. As all map task results serve as input for all reduce tasks, the respective tuning can have a big impact on job execution time. Many MapReduce jobs are limited by the available bandwidth which makes a minimizing data transfer between map and reduce tasks desirable. Combiner jobs pose a possibility to optimize this process by merging the map task results in a specified manner [Whi15, Datc].

At latest when a dataset outgrows a single physical machine in size it becomes necessary to partition the data on various machines, requiring distributed filesystem which manages storage across a network of machines. The challenges of network technology add an additional level to regular filesystems, e.g. the prevention of data loss in case of node failure. Hadoop's introduction of HDFS is designed for storing large files with streaming data access patterns, running on clusters of commodity of hardware. The data size may vary from megabytes to petabytes while HDFS is based on the data processing pattern "write once, read-many-times" to ensure efficiency. A copied or generated dataset will have multiple analyses performed on over time with each analysis involving a large proportion of the dataset. Hence the time to read the dataset is more important than the reading latency for the first record [Whi15].

3.1.3. HDFS Concepts

HDFS is built around a variety of concepts of which the most important will be discussed in the following section.

3.1.4. Blocks

Each physical disk has a block size representing the minimum amount of data to be read or written. For a single disk setup a filesystem builds data in blocks as integral multiple of the disk block size. Filesystem blocks are usually in the range of a few kilobytes in comparison to 512 bytes of disk blocks. In HDFS a block is configured much larger with 128 MB by default. Files in HDFS are split into block-sized chunks, stored as independent units. A file smaller than an individual block does not occupy a full block storage in difference to single disk filesystems. Another reason for the increased size of HDFS blocks lies within the minimization of cost of seeks as a large enough block provides improved time to transfer the file compared to the seek to the block start. The obvious benefits of a block structure are the Independence of files to singular disk sizes and simplifying the storage subsystem, especially important for distributed systems. Additionally, blocks are well suited for replication to provide fault tolerance and availability. Typically each block is replicated up to three times to ensure against corruption or machine failure [Whi15, Hada].

3.1.5. Nodes

All HDFS Clusters consist of two different types of node which are cooperating in a master-worker pattern: a namenode (master) and multiple datanodes (worker). As its name suggests, the namenode administrates the filesystems namespace and metadata for all files in the cluster. The metadata is stored persistently on a local device in form of the namespace image and the edit log. The filesystem can be accessed through a client which allows the user to communicating with the name-nodes and data nodes. Datenodes execute the computational work by storing and processing blocks when orchastrated by the namenode and keeping it informed about the currently stored blocks. As all metadata is stored and processed within the name-node it needs to be protected against failure. Without a namenode no reconstruction of the files from the blocks on the datanodes would be possible. Therefore Hadoop offers an approach to configure the namenode to write to multiple filesystems, those are atomic and synchronous. A second appraoch introduces a secondary namenode which does not act as namenode itself but rather a periodically updated namespace image to prevent an overflow of the edit log [Whi15, Alt].

3.1.6. Block Caching

Block Caching presents another HDFS concept allowing some blocks to be cached in memory of the datanodes for frequently required files, more exactly in an off-heap block cache. Algorithms and job schedulers can benfit from cached blocks by running tasks on the respective datanodes to improve read performance. Third party applications or users can configure a cache directives to a cache pool, which are administrative groups for managing resource usage and caching [Whi15].

3.1.7. HDFS Federation and High Availability

In HDFS a namenode still presents a possible single point of failure and restricts potential HDFS clusters with finite memory to keep the metadata of files, blocks and nodes. Hadoop took this into account and introduced multiple namenodes, with each individual managing a portion of the filsystem namespace. Each namenode has a respective namespace volume assigned which consists beside the metadata of the block pool containing the blocks of the namespace. The independence of namespace volumes allows for further scaling and an improved failure insurance as namespaces are independent of each other [IBMa, Whi15].

In the beginning the failure of a namenode was tackled by a) loading the namespace image into memory, b) replaying the edit log and c) received sufficient block reports from the datanodes to leave the safe mode. This process take 30 minutes to several hours of restarting time, which also possessed a maintenance struggle. Hadoop introduced several improvements to prevent this behavious by adding HDFS high availability. Here, a pair of namenodes is configured in an active-standby configuration which allows the standby element to take over in an event of failure. Creation and Maintenance of a shared edit log is improved by the Quorum Journal Manager (QJM) designed by the Hadoop architects to provide a highly available edit log. QJM consists of a group of journal nodes and each log change must be recorded in the majority of these nodes. This ensures that the system remains functional even after the loss of a jounral node [Whi15, Wil].

3.2. Benefits of Hadoop

Hadoop presents several characteristics which are making it a suitable tool for the deployment in big data frameworks. Many of them have already been highlighted but will nonetheless be summarized in this section. Firstly, Hadoop is cost-efficient due to the low requirements for used hardware. For example can already cheap commodity servers be sufficient to host and run a cluster, giving users a high flexibility to set up matching hardware for different project needs. Another cost-efficient aspect is the open-source availability of the software, making it free of charge. Still, similar to traditional data storage a appropriate CPU to memory to disk ratio needs to be determined by the estimated workload. Hadoop benefits from JBOD, just a bunch of disks, configuration, referring to a collection of disks in a system or array combined as a single logical volume as the I/O pattern fits Hadoop's model. JBOD constitutes the opposite approach to partitioning where individual drivers are separated to smaller volumes. [GP] However, master and worker nodes typcially differ in the hardware as master nodes need to be more robust to failures and provide essential cluster services. On the other hand, worker nodes are known to show failure on a regular basis. The process of hardware selection is usually handled best in terms of cost efficiency when reducing the proliferation of hardware profiles by selecting a single hardware profile for all masters and worker nodes respectively [Hada].

The second advantages goes hand in hand with the first as Hadoop's inherent shared-nothing architectures handles all nodes independently, enabling users to scale clusters

both vertically and horizontally. Vertical Scaling can be achieved by adding resources to existing machines in form of CPU or RAM, while the addition of new machines enables horizontal scaling. Horizontal Scaling is supported by Hadoop 3 with up to 10000 data nodes in a single cluster [Alt].

Thirdly, Hadoop allows data import for both structured and unstructured data, which allows the storage of images, audio, text and video files. All data is stored in HDFS without any reprocessing and later modeled with a appropriate schema. Besides its native HDFS storage service, Hadoop is compatible with other storage services such as FTP files systems, Amazon S3, Microsoft Azure Data Lake and Aliyun Object Storage System [Hada, Alt].

Hadoop's HDFS system ensures a high level of fault tolerance, referring to the work intensity of the system confronted with failure and hardware errors. Hadoop 3 provides fault tolerance in the form of erasure coding, a method storing data persistently and thus saving space. In case of hardware failure the data block can be recovered from a parity unit [Das].

Three core elements of Hadoop have already been introduced in this chapter, HDFS, MapReduce, and Yarn. However, Hadoop's open-souirce nature and modularity resulted in many software contributions from various stakeholders to improve its capabilities. This development led to an ecosystem with many interoperable instruments to address various in the context of Big Data. While some tools facilitate data ingestion, especially from relational databases, others provide SQL syntax for Big Data exploration and analysis. In recent years some tools have been introduced to support real-time data pipelines [Alt].

Lastly, Hadoop as a service (HaaS) or Hadoop in the cloud facilitates the usage of Hadoop without the need to invest in any on premise infrastructure, which is in this case managed by a third-party provider. This approach will also be used in the experimental setting of this thesis as it enables the use of appropriate hardware without the investment costs. Popular HaaS providers include Amazon Elastic MapReduce, Microsoft HDInsight, IBM InfoSphere BigInsights, Oracle Big Data Discovery Tool, OpenStack Savanna and Google Cloud Dataproc [Wil].

3.3. Challenges of Hadoop

However, Hadoop also grapples with certain challenges and constraints. Initially, due to the HDFS block size being set at 128 MB, Hadoop demonstrates its strength in handling large capacities but falters when dealing with smaller datasets. Accumulating numerous small files may strain the namenode responsible for storing the namespace, leading to potential overload issues. This hurdle can be mitigated by amalgamating the smaller files into larger ones before their migration into HDFS. Alternatively, storing these diminutive files in HBase, a non-relational database management system operating atop HDFS, offers a viable solution wherein the files are incorporated as binary values within cells [Datc, IBMa].

Hadoop, being geared primarily towards batch processing, isn't the best fit for analyzing streamed or live data. This aspect should be carefully considered before diving into

implementation projects. While Hadoop handles large data volumes efficiently, the output might suffer significant delays for real-time data processing. Moreover, the MapReduce algorithm, which transforms a dataset by converting individual elements into key-value pairs and then further processing them through the reduce part, also contributes to increased latency [Datc, hadb].

Mid-sized companies or institutions with limited experience in implementing robust security measures may encounter hurdles when adopting Hadoop. Security concerns arise as Hadoop lacks encryption at both storage and network levels within the MapReduce process. Although internal support for Kerberos authentication exists, its setup and management complexities are notable. While HDFS incorporates access control lists (ACLs) and follows a conventional file permissions model, third-party solutions facilitate integration with Active Directory Kerberos and LDAP for authentication purposes [Datc].

3.4. Databases in the Hadoop Ecosystem

In order to tap into the advantages of Hadoop without delving into the complexities of creating MapReduce Jobs, configuring nodes, and other tasks, third-party tools have emerged to harness Hadoop's benefits more elegantly. One prominent tool is Apache Hive, an open-source project initiated as a subproject of Apache Hadoop, aimed at facilitating the analysis of large datasets stored in Hadoop's HDFS file system. The evolution of data size from gigabytes to terabytes or even petabytes has posed limitations on MapReduce jobs and increased complexity, leading to the inception of Hive. Hive was primarily created to address two key challenges: firstly, to develop a SQL-based declarative language to minimize disruption and leverage the syntax knowledge of industry experts. Secondly, to simplify the setup of professional Hadoop-supported data usage environments through centralized metadata storage configuration [Data].

Key Features of Hive

The principal feature of Hive lies in its SQL-related query language, HiveQL, which functions akin to an RDBMS with schema-on-read and translates queries into MapReduce, Apache Tez jobs, and Apache Spark. Figure 3.4 illustrates the architecture of Hive.

Hive Server 2 generates the execution plan alongside the requried YARN jobs to execute the HiveQL queries. The inherent Hive optimizer and compiler facilitate improved data processing and extraction. An analytical feature is presented by Hive Data Functions, which enables the processing of big data by facilitating data manipulation and mathematical calculations. As mentioned above, metadata is conveniently stored in an RDBMS to streamline configuration. Hive also supports storing multiple data types such as RCFile, Parquet, plain text, HBase, and more. Hive also provides built-in user-defined functions (UDFs) for data manipulation, which can be extended to meet individual requirements [Ama, Data].

These technical features enable fast and efficient data processing for even petabyte amounts of data in batch processing, utilizing familiar SQL syntax, and enabling scalable

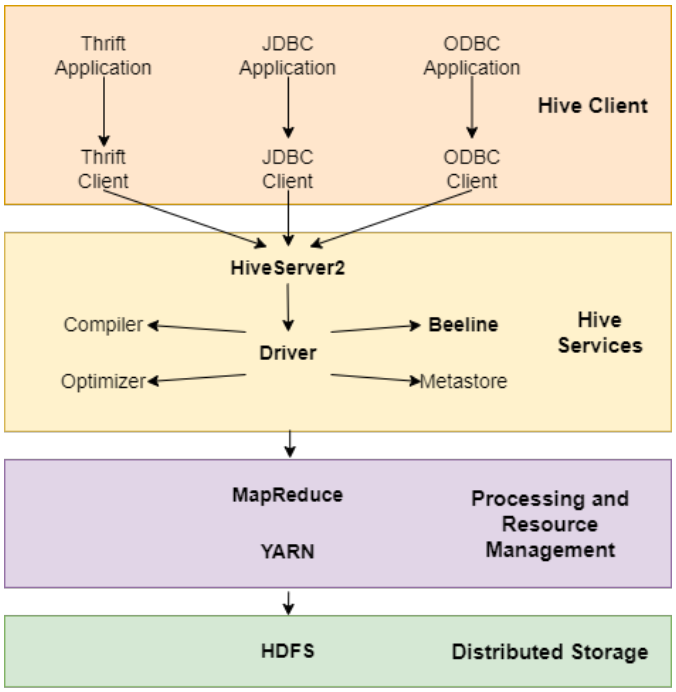

Figure 3.4.: Hive Architecture

deployment on-premises or in the cloud.

4. Methodology

A meaningful comparison between two different database systems requires a well reasoned benchmarking mechanism. This thesis aims to compare the potential advantages of Hadoop databases with traditional healthcare RDBMS environments. To make this comparison as meaningful as possible it could be argued that a benchmarking process should encompass or take into account existing healthcare data as a potential benchmark foundation. Several healthcare databases have undergone deidentification processes, offering authentic datasets derived from real-world scenarios, exemplified by the Medical Information Mart for Intensive Care (MIMIC) [mim]. However the mere existence and accessibility of real-life healthcare data constitutes only a minor aspect of the creation of individual benchmark besides general database design principles outlined in the following.

Database benchmarking serves to assess the performance of database systems, however creating a healthcare-specific benchmark carries risks of bias and incompleteness. To create an effective benchmark, it's crucial to balance healthcare-specific requirements with general database design principles, including appropriate data types. Challenges in this process include generating realistic data, designing relevant queries, and addressing scalability and performance issues. Ensuring these factors are integrated secures a thorough evaluation of healthcare database performance while meeting regulatory standards. Additionally, adapting an existing benchmark from another domain with similar data structures may be more suitable, underscoring the value of repurposing established benchmarking methodologies to meet healthcare-specific needs.

Currently, the predominant approach to database benchmarking relies on well-established and technically sophisticated tools, with products from the Transaction Processing Performance Council (TPC) prominently featured as the most widely utilized non-native database benchmarking solutions [NP06, tpc]. Those benchmarking systems challenge the general database design principle in a professional and general manner. The subsequent section assesses the development of a healthcare-specific benchmarking tool alongside the utilization of an existing benchmark, ultimately making a selection based on this evaluation.

4.1. A Healthcare Domain Specific Benchmark

Creating a database benchmarking requires a clearly defined method for analysing and comparing the measured characteristics. Grey defines 4 aspects to be considered for database benchmarking as shown in 4.1, outlined in the following section. Creating a database benchmarking involves defining a method to analyze and compare the database performance characteristics effectively. Initially, aligning the benchmarking objective with

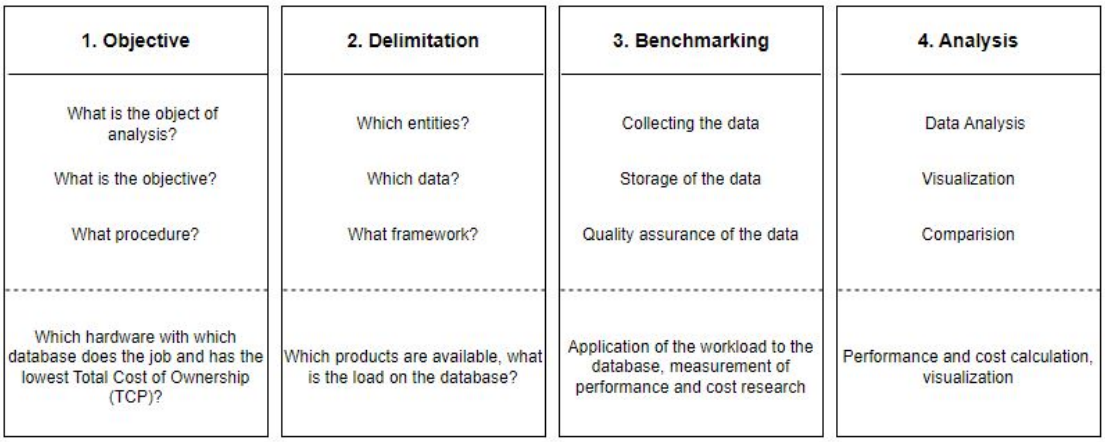

Figure 4.1.: Database Benchmarking by Grey [Ben]

the overarching strategic goal, what database and hardware is optimal for the desired data to be stored, and desired methodology is crucial, aiming to achieve optimal results while minimizing costs or resource utilization [Ben]. For instance, consider a hospital seeking to evaluate the efficiency of its electronic health record system. The benchmarking objective is to accurately measure the system's performance, taking into account factors such as response time for retrieving patient records and the system's ability to handle concurrent user requests, which aligns with the hospital's strategic goal of providing efficient healthcare services while minimizing operational costs.

Next, defining the domain entities, data, and schema becomes essential, considering the expected workload on the database through Create, Read, Update, and Delete (CRUD) operations. For instance, in a healthcare setting, this entails identifying entities like patients, doctors, and appointments, along with associated data such as patient demographics and medical records. Anticipated activities such as adding new patient records, retrieving medical histories, updating treatment plans, and deleting outdated information are then considered to ensure the benchmark accurately simulates real-world database usage in healthcare [Ben].

Subsequently, the benchmark is implemented by collecting, storing, and ensuring the quality of the data in terms of reliability, accuracy, consistency, completeness, and timeliness, while employing associated hardware or performance measurements to track workload, performance, and cost development. For example, a healthcare organization may gather patient data such as medical histories and treatment plans, ensuring accuracy and completeness, while simultaneously measuring the system's response to queries, its ability to handle simultaneous users, and its resource consumption, such as server storage and processing power. This allows for the assessment of the efficiency and effectiveness of the database system in managing healthcare information [Ben, HAK91].

Finally, the benchmarking results undergo thorough analysis, visualization, and ideally, comparison with other databases under consideration. For instance, a hospital may

analyze the benchmarking results of its electronic health record system, visualizing data on system response times and resource usage to pinpoint areas for improvement. These results are then compared with those of similar hospitals to gauge how their system performs relative to industry standards, aiding in making informed decisions to optimize database performance [Ben, MMK+16].

Data Variety in Healthcare

The strategic objective of a benchmark specific to healthcare in this thesis can be succinctly described as *maximizing analytical insights from extensive datasets while efficiently utilizing hardware resources*. It's important to note that Grey's second characteristic emphasizes the significance of aligning the benchmarking process with the unique characteristics of data and its format, rather than being tied solely to a specific domain such as healthcare [Ben].

As it can be derived from the experience and touch points of humans with healthcare specific data, this data comes in various forms and specifications. A typical visit at a general practitioner involves checking in with the individual EHR format, dependent on region and provider [SDD+15]. The diversity in EHR formats persists even within regions of countries, particularly notable in federal states with varying regulatory statutes [SEG+19]. The integration of medical, clinical, and social data into Electronic Health Records (EHR) and Electronic Medical Records (EMR) commonly results in issues pertaining to data sparsity, heterogeneity, and format. Nevertheless, as demonstrated by references [KDZ+19, MMK+16, GSC18], these data formats not only exhibit differences but can also be likened to data formats and structures found in other domains. The consultation might add an findings entry linked to the EHR data and a medical prescription, both again in various data formats. Tools for medical imaging equipment create results based on the software and hardware configurations set by the manufacturer, which results in a variety of data formats and schemas [PT19].

General Digitization in Healthcare

The widespread adoption of object-oriented programming in the healthcare domain, similar to other sectors impacted by digitization, has resulted in the development of applications with comparable outputs in terms of database structure and schema. This is because both healthcare and other domains leverage object-oriented design principles, leading to similar approaches in modeling entities, attributes, and behaviors within their respective databases [HAK91].

Hence using a database benchmarking approach with business domain data and applying its findings to the healthcare sector is feasible, as the similarity in data schema and format, such as the proportion of data types, supports interoperability and information flow between the two domains. This compatibility enhances the adaptability of benchmarking methodologies across sectors, ensuring valuable insights can be effectively transferred and utilized in diverse contexts.

4.2. TPC-DS Benchmarking

When it comes to comparing the performance of different data storage and analysis approaches, TPC has emerged as the industry standard, providing a robust benchmarking framework to ensure meaningful and comparable results [Men19]. TPC achieves this by joining various micro-benchmarks and workload simulations, enabling a thorough comparison of system performance while considering hardware and software characteristics. Given the inherent differences in both the hardware and software components of the database systems evaluated in this thesis, benchmarking becomes especially crucial to identify the solution that maximizes analytical insights for a given input.

Origin of TPC

The Transaction Processing Performance Council was established as a benchmarking tool in 1988 by several companies in response to the escalating marketing claims of database distributors regarding their product's performance [tpc]. Each company had its own individual benchmarking process, but some lacked the inclusion of network and user interaction components of the Online Transaction Processing (OLTP) workload or the necessary oversight and standardization provided by a neutral benchmark institution.

What is TPC-DS?

TPC-DS, or Decision Support Benchmark, models several generally applicable aspects of a decision support system (DSS), including queries and data maintenance [tpc]. The benchmark aims to provide a representative evaluation of performance as a general-purpose decision support tool, enabling the assessment of emerging technologies such as Big Data systems. As data warehouses can be described as data-driven decision support systems relying on Online Analytical Processing (OLAP) and data warehousing methods, TPC-DS benchmarks the database component of a data-driven DSS [NFM07].

TPC-DS exemplifies decision support systems that examine large data volumes, provide answers to real-world questions, and execute queries of various operational requirements and complexities, such as those mirrored in transactional OLTP databases through database maintenance functions [tpc]. The compatibility between business model data and data schema from other sectors, such as retail or finance, and healthcare datasets underscores the feasibility of using existing benchmarks like TPC-DS for healthcare data comparisons. Healthcare data sets often share similar types and requirements with business models, necessitating robust and scalable solutions for data storage and analysis. As demonstrated by the widespread adoption of object-oriented programming in both healthcare and other sectors impacted by digitization, similarities in data schema and format support interoperability and information flow between these domains [HAK91].

Moreover, the utilization of business model data and data schema for benchmarking healthcare datasets ensures that the comparison encompasses not only performance metrics but also considers the broader implications of scalability, interoperability, and adaptability. By leveraging established benchmarks like TPC-DS, which are designed to evaluate the

performance of complex decision support systems across various industries, researchers and practitioners can derive actionable insights that transcend sectoral boundaries.

4.3. Going Forward Utilizing TPC-DS as Benchmark

As this thesis aims to compare the performance of a traditional RDBMS with a database in the Hadoop ecosystem, TPC-DS proves invaluable. Its characteristics, with a focus on RDBMS and Hadoop, make it a valuable solution to test operational, complex queries in the healthcare data domain.

4.3.1. TPC-DS: What is it good for and how does it work

Poess et al. describe TPC-DS as a comparison tool which provides highly comparable, controlled and repeatable tasks in evaluating the performance of database and decision support systems. The workload is meant to test upward boundaries of hardware systems in regards of CPU utilization, memory utilization, I/O subsystem utilization as well as the ability of the OS and database software to perform various complex operations. Some of these complex operations are examining large volumes of data, computing and executing the most performant execution plan for high complexity queries, an efficient scheduling of a large number of user sessions and thereby answering critical analysis questions within a reasonable timeframe [PNW07].

4.3.2. Data Generation and Mapping

The TPC-DS benchmark provides relevant, objective performance data to industry users. In order to achieve this goal underlying systems require a general availability to users and a high data volume in a complex decision support environment. In order to ease the use and generalization of TPC-DS the benchmark is mapped to a typical business environment, more specifically the decision support functions of a retail product supplier are modeled. The supporting schema contains relevant business information, e.g. customer, order and product data. The two most important components of this mature decision support system are represented by user queries converting operational facts into business intelligence and data maintenance focusing on the synchronization of process management analysis with operational external data sources [tpc]. In this subsection we will explore the pivotal parts of TPC-DS in more detail.

The predecessor of TPC-DS focused on real-life commercial databases and assumed a 3rd Normal Form (3NF) which is the most prominent schema design for RDBMS due to the reduction of data duplication and the enforcement of data integrity [Men19]. However, as most data domains have expanded to star schema approaches, TPC-DS implemented a hybrid schema between 3NF and a star schema, also refereed to as multiple snowflake schema, which is widely common for today's data domains. A snowflake schema consists of a fact table which is linked to multiple dimension tables. Fact tables store frequently monitored transactions such as transaction data, medicamental prescriptions or diagnoses. The dimension table contain more static information about more consistent data which is added to supply the fact table with more data. However, in the multiple starflake schema, dimension tables can be linked to multiple fact tables and other dimension tables. This structure reduces the table sizes compared to a strict 3NF modeling approach as the normalization process results in a larger number of tables.

5. Implementation and Comparative Querying Results

For an implementation of a traditional RDBMS and a Hadoop Database with a comparative querying analysis this thesis made use of of the Microsoft public cloud computing platform Azure, some of the key features are displayed in figure 5.1. In difference to on-premise implementations this approach allows to only pay for resources used, choose state of the art data tools with Azure support for configuration and benefit from general developments in the Platform as a Service PaaS and Software as a Service SaaS areas. Within the Azure environment, MySQL was selected as RDBMS and Hive on top of a HDFS file system for the Hadoop database. The following section describes the data generation and import in coordination with the TPC-DS Benchmarking.

Microsoft Azure	
Key Feature	***Description***
Devops	Development barrier between IT operations teams is facilitated
Managed Services	Managed and automated services to simpify many manual and repetitive cloud tasks
Single-Pane Operations	Managemenet-as-a-Service (MaaS) allows taking a single-pane look at hyprid environments
Capacity Management	Solve storage and resource peak issues in a flexible manner
Computational Services	Various computational services and seamingless third-party tool integration

Figure 5.1.: Azure Key Features

5.1. Databases

The managed MySQL database is supported by Azure in terms of provisioning, patching, backups and monitoring of the provided infrastucture. The compute tier consisted of 2-96 vCores for balanced configuration of the workloads and a AMD compute processor with a compute size of 2 vCores, 8 GB memory and 3200 max iops. The storage was set to 20GB with optional increase via auto-growth as required.

The Hive database was created within HDInsight, a managed Azure cloud service enabling deployment of Apache Hadoop, Spark and Hive. The cluster was configured with Hadoop 3.3. (HDI 5.1) and a cluster size for computation of 2 Head nodes (each 4 Cores, 21 GB RAM), 2 worker nodes (each 8 Cores, 64 GB RAM) and 3 Zookeeper (each 2 Cores, 4 GB RAM).

5.2. Data Import and Indexing

The Importing step was executed within Databricks, a unified, public analytics platform for deploying, building and maintaining big data, analytics and AI solutions at scale, it also integrates with cloud solutions such as Azure (see Appendix A.2) [Datb].

Databricks Configuration

The computing foundation of a Databricks Workbook, which can be compared to a Jupyter Notebook, is a an optimized runtime cluster of Apache Spark and the Apache Hadoop ecosystem. The cluster used was configured to Databricks Runtime Version 15.0, including Apache Spark 3.5.0 and Scala 2.12 and had 2 to 8 workers with 14 GB memory and 4 Cores, resulting in 28-112 GB memory and 8-32 Cores. The cluster was also configured with three cluster-scoped init scripts, one to provide Spark SQL performance, another one to create a required JDBC driver and a final one to install the TPC-DS benchmark kit (see Appendix A.1).

MySQL

With this cluster running, two workbooks were created to execute the data creation and importing tasks. The first workbook mounts the cluster cluster to a Azure Blob Storage, which can be imagined as a distributed Cloud File Storage. The second workbook made use of the TPC-DS benchmarking kit and generated the datasets with a scale factor, optional from 1GB to 1PB, as parquet files. In a next step the data generation workbook created a connection to tshe Azure MySQL database and used Spark to write the generated parquet files as separate tables in parallel execution to the database. To increase query performance and enable a meaningful comparison, each MySQL table was attached with a Primary Key and also referenced Foreign Keys. Those relationships were implemented as specified within the TPC-DS benchmarking kit. The TPC-DS data set stored in MySQL had a final size of 5.44GB.

Hive

The data import to the HDFS cluster was executed via a Bash Script through a SSH shell connection in the following way, (A) export of the MySQL tables as UTF-8 separated CSV Files to the Azure HDInsight Cluster, (B) putting the exported CSV files via distributed file share to the HDFS file system and (C) load the HDFS data as tables into Hive and create the connection to the matching HDFS data for following computations. The TPC-DS data set stored in HDFS had a final size of 16.2GB.

The respective bash scripts can be seen in A.5,A.7 and A.6, the Hive Table Generation Script in A.8.

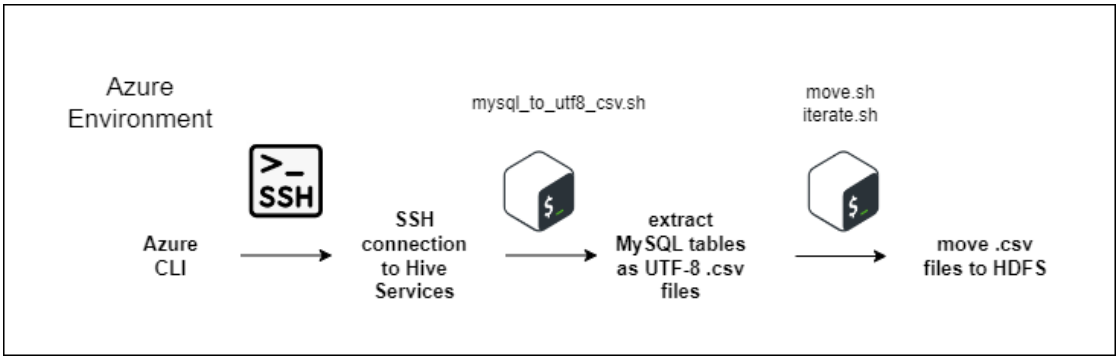

Figure 5.2.: Hive Data Import

5.3. Query Translation and Execution

TPC-DS provides 99 queries to be performed against the tested databases. The database consists of 25 tables, showing complex relationships and constraints between each other to mimic real-world scenarios. The queries support and test workloads involving complex queries joining multiple tables, filter large datasets, and aggregate data. The complexity is expressed by combing subqueries, aggregations, window functions, Common Table Expressions (CTEs), Windowing and Partitioning, Conditional Logic and Performance Optimization Techniques as can be seen in A.3 and A.4.

Query Translation

To execute the queries they had to be translated to the required syntax of MySQL and Hive. While the translation the the MySQL syntax from the provided TPC-DS queries was rather straightforward, the translation to HiveSQL involved changing of datatypes, e.g. from varchar to string, or Hive-inherent syntax, e.g. TRANSFORM for custom map-reduce operations or LATERAL VIEW for handling complex data structure. The result of the translations for both MySQL and HiveSQL can be seen in A.3 and A.4.

Query Execution and Performance Logging

Before the TPC-DS queries were run on the databases, both were configured to activate profiling. Additionally stored procedures logged the performance results of the queries in a separate table to allow the comparison of the parameters in the next section.

Query Complexity

To give an estimation of the complexity of the TPC-DS queries we will examine one example, query 11.

Common Table Expressions (CTEs)

```sql
WITH year_total AS (
    SELECT
        c_customer_id AS customer_id,
        c_first_name AS customer_first_name,
        c_last_name AS customer_last_name,
        [...]
    FROM
        customer,
        store_sales,
        date_dim
    WHERE
        c_customer_sk = ss_customer_sk
        AND ss_sold_date_sk = d_date_sk
    GROUP BY
        c_customer_id,
        c_first_name,
        [...]

    UNION ALL

    SELECT
        c_customer_id AS customer_id,
        c_first_name AS customer_first_name,
        c_last_name AS customer_last_name,
        [...]
    FROM
        customer,
        web_sales,
        date_dim
    WHERE
        c_customer_sk = ws_bill_customer_sk
        AND ws_sold_date_sk = d_date_sk
    GROUP BY
        c_customer_id,
        c_first_name,
        [...]
)
```

Listing 5.1: CTE SQL

CTEs are used to define temporary result sets which can be referenced within the main query. They allow breaking down complex logic into more manageable parts and are deleted when the session is closed. In this query the year total calculates the total sales for each customer for both store sales and web sales.

Self-Joins

```sql
FROM
    year_total t_s_firstyear,
    year_total t_s_secyear,
    year_total t_w_firstyear,
    year_total t_w_secyear
WHERE
```

```
)
```

Listing 5.2: Self-Join SQL

Query 11 joins the year total CTE 11 times with itself. These self-joins are used to compare sales data for different years and sales types for each customer. Self-joins add another level of complexity, especially when dealing with multiple instances of the same table.

Conditions

```
WHERE
    t_s_secyear.customer_id = t_s_firstyear.customer_id
    AND t_s_firstyear.customer_id = t_w_secyear.customer_id
    AND t_s_firstyear.customer_id = t_w_firstyear.customer_id
    AND t_s_firstyear.sale_type = 's'
    AND t_w_firstyear.sale_type = 'w'
    AND t_s_secyear.sale_type = 's'
    AND t_w_secyear.sale_type = 'w'
    AND t_s_firstyear.dyear = 2001
    AND t_s_secyear.dyear = 2001 + 1
    AND t_w_firstyear.dyear = 2001
    AND t_w_secyear.dyear = 2001 + 1
    AND t_s_firstyear.year_total > 0
    AND t_w_firstyear.year_total > 0
    AND CASE
        WHEN t_w_firstyear.year_total > 0 THEN t_w_secyear.year_total /
   t_w_firstyear.year_total
        ELSE 0.0
    END > CASE
        WHEN t_s_firstyear.year_total > 0 THEN t_s_secyear.year_total /
   t_s_firstyear.year_total
        ELSE 0.0
    END
```

Listing 5.3: Conditions SQL

The WHERE clause included complex conditions for filtering the result based on sales type, year, and total sales amounts. These conditions also increase the query's complexity and require careful understanding for accurate interpretation.

Aggregate Functions

```
SELECT
    c_customer_id,
    SUM(ss_ext_list_price - ss_ext_discount_amt) AS total_sales_amount
FROM
    customer
    JOIN store_sales ON c_customer_sk = ss_customer_sk
GROUP BY
    c_customer_id;
```

Listing 5.4: Aggregate Functions SQL

Within the CTE aggregate functions such as SUM() are used to calculate the total sales amount for each customer. Aggregate functions operate on the complete selected dataset, potentially involving large volumes. They often also require grouped data to determine the desired results and are therefore often implemented as Window Functions, which require additional computation resources.

Conditional Logic

```
SELECT
    customer_id,
    CASE
        WHEN total_sales > 1000 THEN 'High'
        WHEN total_sales > 500 THEN 'Medium'
        ELSE 'Low'
    END AS sales_category
FROM
    temp_sales;
```

Listing 5.5: Conditional Logic SQL

The CASE statements in query 11's WHERE clause introduce conditional logic based on the total sales.

Ordering and Limiting

```
SELECT
    customer_id,
    total_sales
FROM
    temp_sales
ORDER BY
    total_sales DESC
LIMIT 10;
```

Listing 5.6: Ordering and Limiting SQL

The ORDER BY and LIMIT clause are used to sort the result and limit the number of rows returned. While those clauses primarily affect the presentation of the query rather than the required computational complexity, sorting large result sets can be resource-intensive. Especially if they can not be fully performed in memory and hence require disk-based sorting. Applying the LIMIT clause does also not improve query performance, as the whole result set needs to be fetched for the ORDER BY clause anyway.

5.4. Query Performance Results

The overall results of the Query Performance against MySQL and Hive showed both the analytical benefits of implementing a Hadoop Database as well as the potential drawbacks and required considerations regarding data volume and complexity.

In a first step, the performance of the queries with the longest execution time against the MySQL database will be described. For comparison, the configuration of the MySQL database in the implemented scenario is comparable to an independent EHR database of a local hospital. [AMDF+14]

MySQL Outliers

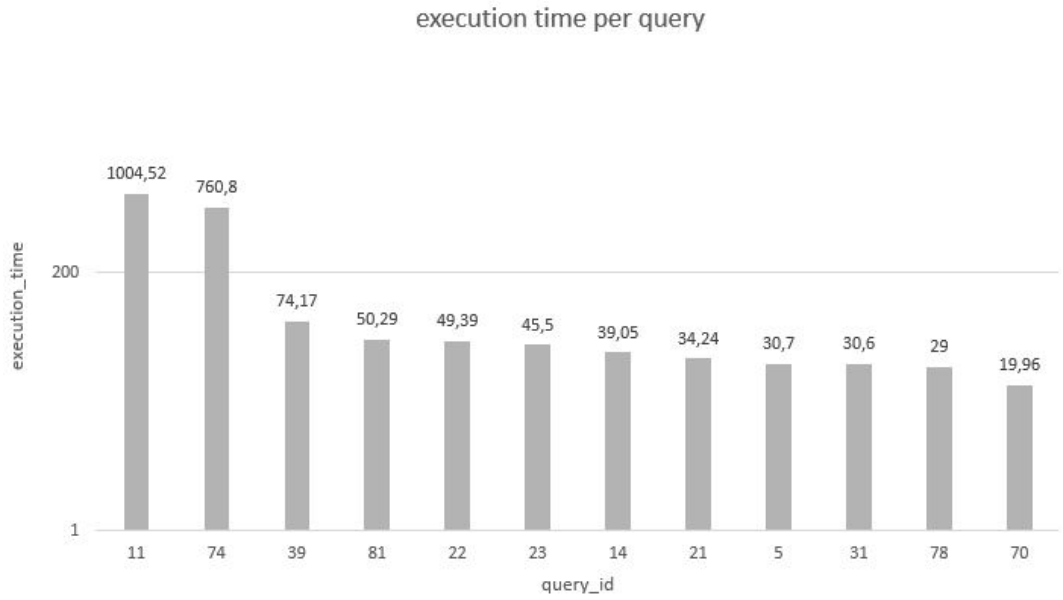

Figure 5.3.: Extended Duration: Queries with Lengthy Execution Time on MySQL

Figure 5.3 displays the time it took for queries to run, arranged from the longest execution time. As can be seen there are two outliers with an execution time of 1005 and 761 seconds, 10 times longer than the third ranking query. A first reason for this long execution time lies in the structure of the queries. As can be seen in A.3 query 11 involves multiple subqueries, multiple joins of large fact tables, aggregate functions, conditional logic within the 'WHERE' clause and 'CASE' statements and ordering as well as limiting. Query 74 shows the same characteristic, but adds CTEs with individual subqueries and comparison and filtering. This already shows, that complex modeled databases with reasonable resources can result in queries with prolonged execution times.

Nevertheless, a second reason can also be found in hardware constraints for those queries. While query 11 and 74 show a high number of involuntary context switches (155.466 and 117.991) the CPU usage never exceeded 60 percent. Taken also the high number of disk writing activity, with a `BLOCK_OPS_OUT` value of 151.552 and 50.400, the bottleneck seems to be storage solution. To optimize disk I/O operations adding additional SSDs will result in improved execution time. The presence of page faults also suggests that the MySQL database occasionally needs to access data not currently stored in memory. Therefore

additional available RAM will also improve query performance.

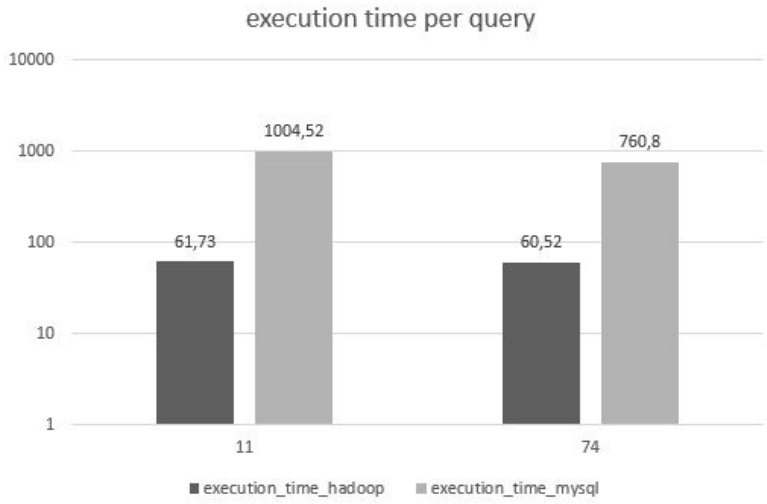

Figure 5.4.: Queries 11 and 74 run against Hive and MySQL

Figure 5.4 shows a substantially improved execution time of both queries when run against Hive with 62 and 61 seconds, resulting in a more than ten times faster execution compared to MySQL. Due to the complex structure of both queries Hive's distributed processing across multiple nodes in the cluster and resulting parallelism showed significant performance boost. Both queries resulted in 6 maps and 5 reduce jobs required, which underscores the scalability of Hive's processing capabilities. This distributed approach enables Hive to handle large datasets better and run complex queries more efficiently than a single-node traditional RDBMS like MySQL. The observed performance improvement reflects Hive's ability to leverage distributed computing for complex queries and big data sets, yielding faster and more scalable analytics possibilities.

Medium performance queries

Figure 5.5 highlights the MySQL queries with a runtime between 19 and 75 seconds and includes the metrics of involuntary contexts and block ops out. Those queries suggest varying performance and resource utilization patterns of the underlying MySQL database. Queries as 39, 81, 22 and 23 present a longer execution time, potentially indicating the complex nature of the TPC-DS queries or inefficient indexing strategies. However as the TPC-DS Schema is deployed, the indexing can be considered sufficient and excluded as a factor. The presence of high involuntary context switches in queries such especially 5 but also 39, 23 and 78, could point to resource contention or inefficient query execution plans, allowing for an investigation of further optimization opportunities.

Of the discussed 10 queries with an execution time between 19 and 75 seconds against MySQL, only 39, 81 and 22 had a substantial better execution time of at least 20 seconds when run against Hive (see Figure 5.6). When comparing the MySQL and Hive results

query_id	execution_time	contexts_involuntary	block_ops_out
39	74,17	11.154	24.832
81	50,29	39	0
22	49,39	10.465	2.159.344
23	45,5	25.690	483.392
14	39,05	9.986	167.664
21	34,24	4.705	96
5	30,7	226.532	738.592
31	30,6	3.644	12.800
78	29	37.329	1.039.008
70	19,96	4.746	190.136

Figure 5.5.: Balanced Efficiency: Queries Executed against MySQL with Moderate Performance

query_id	execution_time	map_jobs	reduce_jobs
14	95,28	40	12
23	72,25	11	10
5	57,61	11	5
78	57,35	7	4
31	52,80	11	6
70	51,15	5	4
39	46,72	5	3
21	34,86	4	2
22	28,48	4	2
81	23,56	5	4

Figure 5.6.: Moderate Performance: Queries Executed on Hive with Balanced Efficiency

it becomes clear, that query complexity with a high execution time against a MySQL database does not necessarily has to result in a high execution time for Hive due to its inherent nature of processing all data and coordinating multiple jobs. For example query 81 finished after 50 seconds against MySQL, solely due to query complexity as can be concluded from the low involuntary contexts and block ops out, but took only 24 seconds against Hive.

However, most queries show a prolonged execution time when executed against Hive. As Hadoop divides data into blocks and distributes them across the cluster, an unpreve ntable overhead due to data retrieval and coordination amongst nodes contributes to a base level of latency. Additionally, the MapReduce paradigm involves a series of map and reduce tasks, each requiring initialization, data transfers and computation, creating more processing time. Those characteristics illustrate the trade-off between scalability and latency inherent in Hadoop's distributed processing model.

Solid MySQL Performance

The following subsection focuses on analyzing the 87 queries with the least execution time in MySQL and their comparative performance when executed against Hive.

From the total of 99 queries 15 had an execution time between 10 and 18, 37 between 1

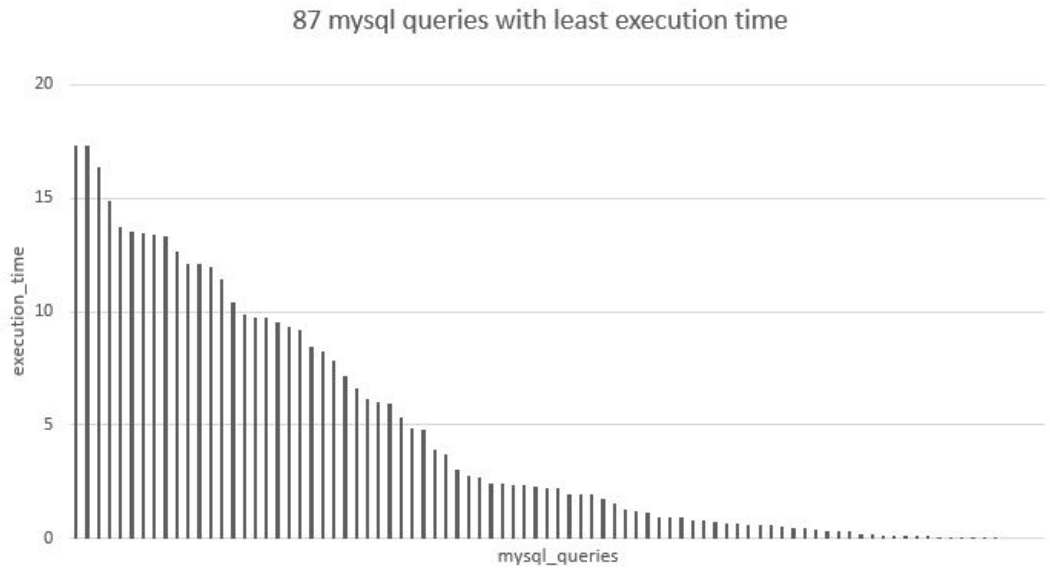

Figure 5.7.: Optimized Efficiency: 87 MySQL Queries with Minimal Execution Time

and 10 seconds, while 35 queries took less than a second to execute (see Figure 5.7). The variance in the execution time can be derived with differences in the complex operations and resource-intensive structure of the queries. The three queries taking longer than 15 seconds to execute share the composition of multiple analytical functions, self-joins and a high level of conditional logic.

On the other hand, the queries taking less than 10 seconds to execute share a simpler structure with fewer CTEs and joins compared to the queries with a higher execution time. Additionally, the filtering conditions in the WHERE clause are more straightforward and less computational intensive. Conditions such as "s_state = 'TN'" also reduce the number of rows required to be processed. The more performant queries also share a limited subquery usage and efficient index usage by mostly referencing primary or foreign key columns in joins and conditions.

The same 87 queries executed against Hive had a significantly higher execution time, being 41 seconds on average (see Figure 5.8). The Hive queries which took longer than 70 seconds to execute share a higher number of map and reduce jobs, e.g. 17 map and 3 reduce jobs for query 75. If high-priority jobs are running concurrently, this prolongs the execution time under the given hardware configuration.

Query 49 on the other hand, with an execution time of 54 seconds, involves 8 map and 12 reduce jobs. In this case, the disk I/O overhead incurred during data processing indicated by the higher level of reduce jobs, such as reading input data and writing intermediate results to disk, can contribute to longer execution times.

It must be emphasized that this longer execution time derives from the inherent nature of Hadoop, relying on MapReduce for query execution, which involves substantial overhead

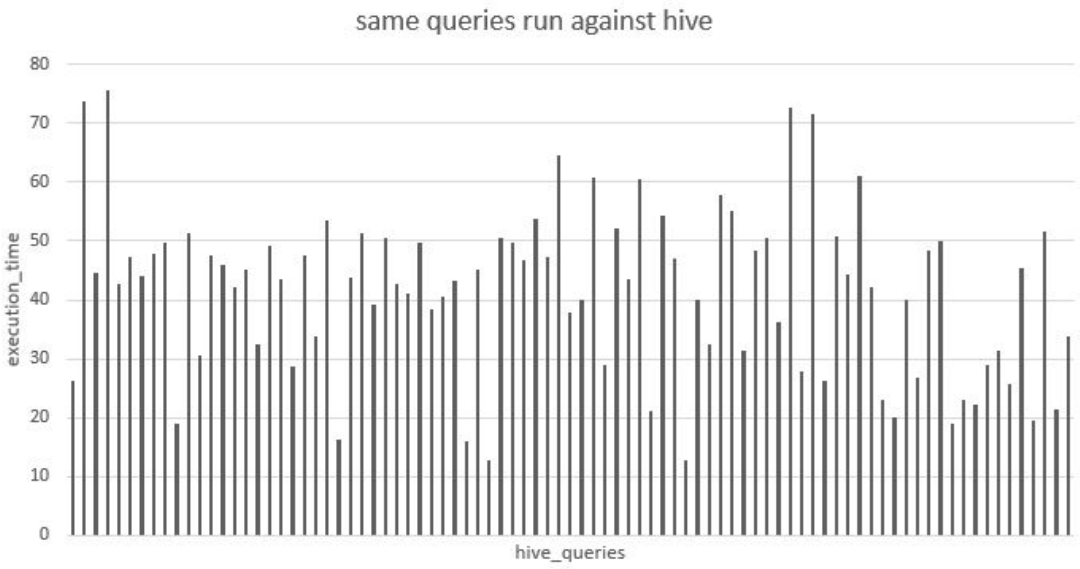

Figure 5.8.: Performance Comparison: Top 87 MySQL Queries Executed on Hive

in job initialization, data shuffling and task coordination. The columnar storage format and processing is designed for large scale data analytics and data warehousing where the latency is negligible, when comparing to much longer execution times in RDBMS. The distributed computing with multiple nodes in the cluster, including CPU, memory, disk I/O and network bandwidth results in execution times, which can not be compared to RDBMS solutions for the size of the dataset implemented in this thesis.

5.5. Resource Limitations

As all used computational resources in this thesis are part of the Azure platform. To increase the size of the 5GB TPC-DS data set used, the scaleFactor in the Databricks data generation notebook just needs to be adjusted to the desired size. However, because there was no suitable university subscription available at the time, the cost of the resources and infrastructure used already added up to about 250 Euros. As the size of the data increases, the costs will also generally rise at a similar rate. This is because the needs for data storage and the computational resources required to edit and move the data sets will grow. Additionally, both database solutions will need more computing power to process and query the data. All of these increased requirements are billed by the hour of compute time used. Because of this, no additional comparison between databases using larger TPC-DS data sets was feasible. However, we will consider and discuss what might have happened if we had in the next chapter.

6. Discussion and Conclusion

The decision of adding Hadoop's Big Data analyzing capabilities to an existing RDBMS landscape should be based on the specific requirements and desired capabilities of any given healthcare institution. An institution which does only need to store data for governance reason without the need for further analyzing has no need to change or extend an existing RDBMS solution. However, many institutions from local hospitals to insurance carriers, may find themselves in situations where they want to implement a use case for organizational or patient care improvement by analyzing the gathered RDBMS data. An insurance carrier might efficiently process and analyze diverse data types such as EHRs, medical imaging, and genomics data to support clinical decision-making, research, and population health management. Public Health Agencies which collect, manage, and analyze population health data, might monitor disease outbreaks, track healthcare trends and inform public health policies. Before discussing meaningful threshold of data sizes and technical limitations, the two implementation strategies of Hadoop are outlined and discussed in the following.

Cloud Computing Services

The implementation in this thesis of a traditional RDBMS alongside a Hadoop database for comparative querying analysis leveraged Microsoft's Azure as cloud computing platform. However, it must be assumed, that many healthcare institutions still operate on a on-premise legacy data warehouse, as the general shift towards a broader deployment of cloud services is still ongoing. Nevertheless, today's cloud computing platforms all offer a variety of options for facilitated data transfer from local databases to cloud instances, e.g. for Azure tools such as Data Factory, Data Migration Assistant, Database Migration Service and manual methods such as Storage Explorer. And of course you can always deploy a back-to-the-roots approach via command-line tools. The usage of cloud computing tools is especially useful for institutions which want to facilitate hardware and network configuration settings. Those can be customized to the desired degree or be left in a managed state by the respective cloud provider. The implementation executed in this thesis might also be considered from institutions taking a general data landscape shift to the cloud by boosting analyzing capabilities through applying Big Data technology like Hadoop.

Hadoop On Premise

In case the usage of a cloud computing service is not feasible, Hadoop can still be deployed on-premise. This involves additional steps such as (1) Infrastructure set-up, by

provisioning physical or virtual servers to host Hadoop components such as name node, data node, resource manager, etc. alongside a matching network configuration to facilitate communication between nodes, (2) Hadoop Installation, installation of the desired Hadoop distribution and configuration of each component on the respective nodes in accordance with the desired cluster architecture, (3) Cluster Configuration, configuration of cluster settings such as replication factor, block size, memory allocation and tuning parameters based on hardware specifications and workload requirements, (4) Data Storage, setting up the distributed file system HDFS to store and manage the data volumes across the cluster, (5) Resource Management, configuration of a resource management framework like YARN to manage cluster resources and allocate resource to MapReduce jobs. This set up might be preferable for healthcare organizations with a high level of security requirements or legal restrictions to store data only on premise. However this implementation comes with a higher cost of infrastructure and deployment maintenance, as no managed infrastructure comparable to cloud computing platforms can be used. Therefore this implementation approach requires a meaningful use case which must be throughout evaluated in meeting its proposed requirements in order to justify the additional costs by adding the desired benefit.

6.1. Data Volume and Complexity

6.1.1. Implementation Findings

The implemented TPC-DS benchmarking in this thesis was done with a final RDBMS size of 5.44GB. Both the MySQL database as well as the Hive database with its HDFS cluster were equipped with sufficient resources for the task of analyzing the generated TPC-DS data set of this sizing. Even though 5.44GB are far below from any vague definition of Big Data, some main findings could be demonstrated which also translate to larger data volumes.

Hadoop Overhead

The discussed characteristics of Hadoop's inherent nature of job initialization, data shuffling and task coordination results in a base level of overhead which makes it impossible to compete with queries resulting in execution times of less than 15 seconds against a RDBMS database. All of the 99 queries had an execution time of more than 10 seconds, 77 even more than 20 seconds when executed against Hive. This result shows clearly, that for a healthcare institution with an RDBMS of moderate data volume and a low level of data complexity, the additional implementation of a Hadoop database will not yield in performance of efficiency optimization.

Hadoop's Advantage

Despite the moderate data volume, 11 queries showed a comparable execution time for MySQL and Hive. Two complex queries even outperformed MySQL with a factor bigger

10. This highlights the performance of Hive's distributed processing across the cluster in a parallel manner, as it would outperform the RDBMS when subtracting the base level of overhead. If a larger data set would be applied in the same database configuration, Hive would outperform the MySQL database already when sizing up to around 50GB. However, this is not a real word scenario for such a low data volume. In case the data volume is increased, the resources of the RDBMS would be adjusted accordingly, as long as scaling of the RDBMS is feasible by physic and costs.

6.1.2. Decision of Implementing a Hadoop Database

As RDBMS are limited in vertical scaling through using CPU with more computational resources or adding RAM, even data volumes starting from 50GB upwards query performance may start to degrade noticeably. [Hau, ESE+22] The decision to implement a Hadoop database alongside an existing RDBMS in a healthcare institution should be carefully evaluated based on various factors discussed throughout this thesis. While the integration of Hadoop can offer significant advantages in terms of processing large volumes of diverse data and enabling advanced analytics, it is essential to consider the specific needs, resources, and constraints of the organization.

For healthcare institutions grappling with increasing data volumes and complex analytical requirements, especially those seeking to leverage data for organizational improvement, clinical decision-making, research, and population health management, Hadoop can be a valuable addition to their technology stack. However, it is crucial to weigh the benefits against the costs and challenges associated with Hadoop implementation, such as infrastructure setup, maintenance, and the learning curve for staff.

Ultimately, the decision to implement a Hadoop database should be driven by a clear understanding of the organization's objectives, technical capabilities, and the expected return on investment. By carefully evaluating these factors and aligning the implementation strategy with the institution's goals, healthcare organizations can harness the power of Hadoop to unlock insights from their data and drive innovation in patient care and healthcare delivery.

Bibliography

[AAS+15] Wahiba Abdessalem, Amira S. Ashour, Dhekra Sassi, Payel Roy, Noreen Kausar, and Nilanjan Dey. *MEDLINE Text Mining: An Enhancement Genetic Algorithm based Approach for Document Clustering, Applications of Intelligent Optimization in Biology and Medicine: Current Trends and Open Problems*, pages 267–87. 03 2015.

[Alt] The good and the bad of hadoop big data framework. `https://www.altexsoft.com/blog/hadoop-pros-cons/`. Accessed: March 24, 2024.

[Ama] Amazon Web Services. What is Apache Hive? | Amazon Web Services. `https://aws.amazon.com/de/what-is/apache-hive/`. Accessed: March 24, 2024.

[AMDF+14] Julia Adler-Milstein, Catherine Desroches, Michael Furukawa, Chantal Worzala, Dustin Charles, Peter Kralovec, Samantha Stalley, and Ashish Jha. More than half of us hospitals have at least a basic ehr, but stage 2 criteria remain challenging for most. *Health Affairs*, 08 2014.

[ATT+02] M.R.B. Araujo, C. Traina, A. Traina, J.M. Bueno, and H.L. Razente. Extending relational databases to support content-based retrieval of medical images. In *Proceedings of 15th IEEE Symposium on Computer-Based Medical Systems (CBMS 2002)*, pages 303–308, 2002.

[Ben] Benchant. Database benchmarking: How and why. `https://benchant.com/blog/database-benchmarking`. Accessed: 2024-02-10.

[BFHS02] Donald Berndt, John Fisher, Alan Hevner, and James Studnicki. Healthcare data warehousing and quality assurance. *Computer*, 34:56–65, 01 2002.

[CGK15] Satyadhyan Chickerur, Anoop Goudar, and Ankita Kinnerkar. Comparison of relational database with document-oriented database (mongodb) for big data applications. pages 41–47, 11 2015.

[Cod70] E. F. Codd. A relational model of data for large shared data banks. *Commun. ACM*, 13(6):377–387, jun 1970.

[Das] Payal Das. Fault tolerance in hdfs. `https://payal-das.medium.com/fault-tolerance-in-hdfs-35cc66b265ce`. Accessed: March 23, 2024.

[Data] Databricks. Apache Hive - Glossary | Databricks. `https://www.databricks.com/glossary/apache-hive`. Accessed: March 24, 2024.

[Datb] Databricks. Databricks Documentation. https://docs.databricks.com/en/introduction/index.html. Accessed: March 27, 2024.

[Datc] DataFlair. 13 limitations of hadoop. https://data-flair.training/blogs/13-limitations-of-hadoop/. Accessed: 2024-02-17.

[DG04] Jeffrey Dean and Sanjay Ghemawat. Mapreduce: Simplified data processing on large clusters. volume 51, pages 137–150, 01 2004.

[DH05] Jocelyn Dewitt and Philip Hampton. Development of a data warehouse at an academic health system: Knowing a place for the first time. *Academic medicine : journal of the Association of American Medical Colleges*, 80:1019–25, 12 2005.

[dS17] Nisansa de Silva. *Relational Databases and Biomedical Big Data*, volume 1617. 05 2017.

[EL14] Mehmet Ercan and Michael Lane. An evaluation of nosql databases for electronic health record systems. 12 2014.

[ESE+22] Kamal Eldahshan, Eman Selim, Ahmed Ebada, Mohamed Abouhawwash, Yunyoung Nam, and Gamal Behery. Handling big data in relational database management systems. *Computers, Materials and Continua*, 72:5149–5164, 04 2022.

[GP] Alexander S. Gillis and Brien Posey. Jbod (just a bunch of disks). https://www.techtarget.com/searchstorage/definition/JBOD#:~:text=JBOD%2C%20which%20stands%20for%20just,combined%20as%20one%20logical%20volume. Accessed: March 29, 2024.

[GSB15] Jasmeen Gill, Shaminder Singh, and Devdutt Baresary. Big data: Big innovations in healthcare. 08 2015.

[GSC18] Francesco Gargiulo, Salvatore Silvestri, and Mario Ciampi. A clustering based methodology to support the translation of medical specifications to software models. *Applied Soft Computing*, 71:199–212, 2018.

[Hada] Planning a hadoop cluster. https://www.oreilly.com/library/view/hadoop-operations/9781449327279/ch04.html. Accessed: March 27, 2024.

[hadb] The Top 12 Apache Hadoop Challenges. https://crayondata.ai/the-top-12-apache-hadoop-challenges/. Accessed: February 17, 2024.

[HAK91] Heather Heathfield, Jim Armstrong, and Nigel Kirkham. Object-oriented design and programming in medical decision support. *Computer Methods and Programs in Biomedicine*, 36(4):239–251, 1991.

[Hau] Philipp Hauer. Relational databases: Strengths and weaknesses compared to mongodb. `https://phauer.com/2015/relational-databases-strength-weaknesses-mongodb/`. Accessed: April 29, 2024.

[IBMa] IBM. Hbase. `https://www.ibm.com/topics/hbase`. Accessed: 2024-02-17.

[IBMb] IBM. Ibm relational databases. `https://www.ibm.com/topics/relational-databases#:~:text=Invented%20by%20Don%20Chamberlin%20and,delete%20rows%20of%20data%20easily`. Accessed: April 5, 2024.

[KDZ+19] S Kalogiannis, K Deltouzos, EI Zacharaki, A Vasilakis, K Moustakas, J Ellul, and V Megalooikonomou. Integrating an openehr-based personalized virtual model for the ageing population within hbase. *BMC Medical Informatics and Decision Making*, 19(1):25, Jan 2019.

[Kim96] Ralph Kimball. *The Data Warehouse Toolkit*. John Wiley & Sons, Inc., New York, NY, 1996.

[LSC+09] Der-Fa Lu, William Nick Street, Faiz Currim, Ray Hales Hylock, and Connie White Delaney. A data modeling process for decomposing healthcare patient data sets. 2009.

[LSH08] Jason Lyman, Kenneth Scully, and James Harrison. The development of health care data warehouses to support data mining. *Clinics in laboratory medicine*, 28:55–71, vi, 04 2008.

[Mad12] Sam Madden. From databases to big data. *IEEE Internet Computing*, 16(3):4–6, 2012.

[med] Can rdbms handle big data? `https://medium.com/analytics-vidhya/can-rdbms-handle-big-data-b583c1584d5f`. Accessed: March 24, 2024.

[Men19] Julian Menzler. A summary of tpc-ds. *Hyrise*, June 19 2019. Published in Hyrise. 9 min read.

[mim] MIMIC-IV Database Documentation - Sources. `https://mimic.mit.edu/docs/about/sources/`. Accessed: February 9, 2024.

[MMK+16] Kenneth D Mandl, Joshua C Mandel, Isaac S Kohane, David A Kreda, and Rachel B Ramoni. Smart on fhir: a standards-based, interoperable apps platform for electronic health records. *Journal of the American Medical Informatics Association*, 23(5):899–908, 02 2016.

[MSJK17] Mashooque Memon, Safeeullah Soomro, Awais Jumani, and Muneer Kartio. Big data analytics and its applications. *Annals of Emerging Technologies in Computing*, 1, 10 2017.

[Nav92] Shamkant B. Navathe. Evolution of data modeling for databases. *Commun. ACM*, 35(9):112–123, sep 1992.

[NFM07] Ivana Nizetic, Kresimir Fertalj, and Boris Milainović. An overview of decision support system concepts. 2007.

[NP06] Raghu Nambiar and Meikel Poess. The making of tpc-ds. pages 1049–1058, 01 2006.

[OO15] Iroju Olaronke and J. Olaleke. A systematic review of natural language processing in healthcare. *International Journal of Information Technology and Computer Science*, 08:44–50, 08 2015.

[OO16a] Iroju Olaronke and Oluwaseun Ojerinde. Big data in healthcare: Prospects, challenges and resolutions. 12 2016.

[OO16b] Iroju Olaronke and Ojerinde Oluwaseun. Big data in healthcare: Prospects, challenges and resolutions. In *2016 Future Technologies Conference (FTC)*, pages 1152–1157, 2016.

[Ora] Oracle Corporation. Oracle database sql language reference. `https://docs.oracle.com/cd/E11882_01/server.112/e40540/sqllangu.htm#CNCPT015`. Accessed: April 5, 2024.

[OSGO13] Iroju Olaronke, Abimbola Soriyan, Ishaya Gambo, and J. Olaleke. Interoperability in healthcare: Benefits, challenges and resolutions. *International Journal of Innovation and Applied Studies*, 3:2028–9324, 04 2013.

[PLG+97] Jonathan Prather, David Lobach, L Goodwin, J Hales, M Hage, and Ed Hammond. Medical data mining: Knowledge discovery in a clinical data warehouse. *Proceedings : a conference of the American Medical Informatics Association / ... AMIA Annual Fall Symposium. AMIA Fall Symposium*, 4:101–5, 02 1997.

[PNW07] Meikel Poess, Raghu Nambiar, and David Walrath. Why you should run tpc-ds: A workload analysis. pages 1138–1149, 01 2007.

[PSA05] Bambang Parmanto, Matthew Scotch, and Sjarif Ahmad. A framework for designing a healthcare outcome data warehouse. *Perspectives in health information management / AHIMA, American Health Information Management Association*, 2:3, 02 2005.

[PT19] Jörg Polzehl and Karsten Tabelow. *Medical Imaging Data Formats*, pages 15–24. 09 2019.

[RC18] Blagoj Ristevski and Ming Chen. Big data analytics in medicine and healthcare. *Journal of Integrative Bioinformatics*, 15, 05 2018.

[RZL+18] Shan Ren, Yingfeng Zhang, Yang Liu, Tomohiko Sakao, Donald Huisingh, and Cecilia Almeida. A comprehensive review of big data analytics throughout product lifecycle to support sustainable smart manufacturing: A framework, challenges and future research directions. *Journal of Cleaner Production*, 210, 11 2018.

[SA14] Geetika Saxena and Bharat Bhushan Agarwal. Data warehouse designing: Dimensional modelling and e-r modelling. *International Journal of Engineering Inventions*, 3:28–34, 2014.

[Sal12] Nancy Salim. What's all the buzz around "big data?": Meet dr. aditya vailaya [the good, the bad, and the ugly: Engineering facts]. *IEEE Women in Engineering Magazine*, 6(2):24–31, 2012.

[SDD+15] Hong Sun, Kristof Depraetere, Jos De Roo, Giovanni Mels, Boris De Vloed, Marc Twagirumukiza, and Dirk Colaert. Semantic processing of ehr data for clinical research. *Journal of Biomedical Informatics*, 58:247–259, 2015.

[SEG+19] Stefano Silvestri, Angelo Esposito, Francesco Gargiulo, Mario Sicuranza, Mario Ciampi, and Giuseppe De Pietro. A big data architecture for the extraction and analysis of ehr data. In *2019 IEEE World Congress on Services (SERVICES)*, volume 2642-939X, pages 283–288, 2019.

[SRM+18] Senthilkumar SA, Bharatendara K Rai, Amruta A Meshram, Angappa Gunasekaran, and Chandrakumarmangalam S. Big data in healthcare management: A review of literature. *American Journal of Theoretical and Applied Business*, 4(2):57–69, 2018.

[SS05] Arun Sen and Atish P. Sinha. A comparison of data warehousing methodologies. *Commun. ACM*, 48(3):79–84, mar 2005.

[TBTK19] Dimpal Tomar, Jai Bhati, Pradeep Tomar, and Gurjit Kaur. *Migration of healthcare relational database to NoSQL cloud database for healthcare analytics and management*, pages 59–87. 01 2019.

[tpc] Transaction Processing Performance Council. `https://www.tpc.org/`. Accessed: January 30, 2024.

[TSJ+09] Ashish Thusoo, Joydeep Sarma, Namit Jain, Zheng Shao, Prasad Chakka, Suresh Anthony, Hao Liu, Pete Wyckoff, and Raghotham Murthy. Hive - a warehousing solution over a map-reduce framework. *PVLDB*, 2:1626–1629, 08 2009.

[Whi15] Tom White. *Hadoop: The Definitive Guide*. O'Reilly Media, Inc., 4th edition, 2015.

[Wil] Sarah Wilson. Hadoop as a service (haas). `https://www.techtarget.com/searchstorage/definition/Hadoop-as-a-service-HaaS`. Accessed: March 23, 2024.

A. Appendix

```
#Generating TPC-DS datasets with spark-sql-perf involves the following steps.
#
#
#Add the spark-sql-perf library jar to your Databricks cluster.
#Mount Azure Data Lake Gen2
#Install the Databricks TPC-DS benchmark kit.
#Restart the cluster.
#Run the TPC-DS-Generate notebook to generate the data and setup the database.
#
#
#Install the jar file from the BlueGranite tpc-ds-dataset-generator repo using the Databricks cluster Libraries menu. Use the jar file that matches the Scala version of the cluster. When downloading the file, make sure you click through to the jar file page in the repo and use the Download button to get the actual file.
#
#Alternately you can build the spark-sql-perf library jar yourself using sbt:
#
#
#Install sbt on your local machine using the instructions here.
#Clone the spark-sql-perf repository and navigate to the new directory.
#
#git clone https://github.com/databricks/spark-sql-perf.git
#
#Run sbt package to build for scala 2.11. Run sbt +package to build for scala 2.11 and 2.12.
#The jar file can then be found in the /spark-sql-perf/target/scala-2.1x directory. Install the library using the Databricks cluster Libraries menu.

#cell1
# Define required configurations
clientId = "b13066a3-71e8-4b12-a1cd-e888b7d658ec"
clientSecret = "DB18Q~jUsFPC4mcH09yoJ8mOmdVU4jnJrp8wGcmN"
tenantId = "23aa4c02-d2f6-4ee1-a93c-ec2eb7540427"
storageAccountName = "asdf89762345"
fileSystemName = "qweri"  # Container name

# Define mount point directory
mountPoint = f"/mnt/{fileSystemName}"

# Define configurations
```

```
configs = {
    "fs.azure.account.auth.type": "OAuth",
    "fs.azure.account.oauth.provider.type": "org.apache.hadoop.fs.azurebfs.oauth2.ClientCredsTokenProvider",
    "fs.azure.account.oauth2.client.id": clientId,
    "fs.azure.account.oauth2.client.secret": clientSecret,
    "fs.azure.account.oauth2.client.endpoint": f"https://login.microsoftonline.com/{tenantId}/oauth2/token"
}

# Determine if mount point already exists
mount_exists = False
for m in dbutils.fs.mounts():
    if m.mountPoint == mountPoint:
        mount_exists = True
        print(f"Mount point {mountPoint} already exists. Unmounting...")
        dbutils.fs.unmount(mountPoint)
        break

# Create mount point directory if it doesn't exist
if not mount_exists:
    print(f"Creating mount point directory {mountPoint}")
    dbutils.fs.mkdirs(mountPoint)

# Check if large file shares are disabled
large_file_shares_enabled = True

# Create mount if large file shares are enabled
if large_file_shares_enabled:
    # Create mount
    print(f"Creating mount point {mountPoint}")
    dbutils.fs.mount(
        source=f"abfss://{fileSystemName}@{storageAccountName}.dfs.core.windows.net/",
        mount_point=mountPoint,
        extra_configs=configs
    )
else:
    print("Large file shares are disabled. Mounting is not supported.")

#cell2
# Load the file from mounted storage into Databricks
# Load the file from mounted storage into Databricks
df = spark.read.parquet("/mnt/datalake/raw/tpc-ds/source_files_001GB_parquet/customer")

output_path = f"{mountPoint}/customer1.parquet"

# Write the DataFrame to blob storage as Parquet
df.write.parquet(output_path)

#cell6
# Define the directory path where you want to store the init script
```

```
directory_path = "/dbfs/Volumes/"

# Define the init script file name
script_name = "tpcds-install.sh"

# Define the script content
script_content = """
#!/bin/bash
sudo apt-get --assume-yes install gcc make flex bison byacc git

cd /usr/local/bin
git clone https://github.com/databricks/tpcds-kit.git
cd tpcds-kit/tools
make OS=LINUX
"""

# Define the full path to the init script file
script_path = directory_path + script_name

# Upload the init script to the specified directory
dbutils.fs.put(script_path, script_content, True)

#cell7
# Check if the file exists
file_path = "/dbfs/Volumes/tpcds-install.sh"

if dbutils.fs.ls(file_path):
    print(f"The file {file_path} exists.")
else:
    print(f"The file {file_path} does not exist.")

#Add the init script to your cluster using the steps below:
#
#
#On the cluster configuration page, click Advanced Options.
#At the bottom of the page, click Init Scripts.
#
#Databricks Init Script menu
#
#In the Destination drop-down, select a destination type.
#Specify a path to the init script.
#Click Add.
#Upload your script to the specified location.
#Restart the cluster.
```

Listing A.1: Databricks Notebook TPC-DS Configure

```
#Generating data at larger scales can take hours to run, and you may want
    to run the notebook as a job.
#
#The cell below generates the data. Read the code carefully, as it
   contains many parameters to control the process. See the Databricks
   spark-sql-perf repository README for more information.

```

```
#cell1

%scala
import com.databricks.spark.sql.perf.tpcds.TPCDSTables

// Set:
val scaleFactor = "1" // scaleFactor defines the size of the dataset to generate (in GB).
val scaleFactoryInt = scaleFactor.toInt

val scaleName = if(scaleFactoryInt < 1000){
    f"${scaleFactoryInt}%03d" + "GB"
  } else {
    f"${scaleFactoryInt / 1000}%03d" + "TB"
  }

val fileFormat = "parquet" // valid spark file format like parquet, csv, json.
val rootDir = s"/mnt/datalake/raw/tpc-ds/source_files_${scaleName}_${fileFormat}"
val databaseName = "tpcds" + scaleName // name of database to create.

// Run:
val tables = new TPCDSTables(sqlContext,
    dsdgenDir = "/mnt/tpcds-kit/tools", // location of dsdgen
    scaleFactor = scaleFactor,
    useDoubleForDecimal = false, // true to replace DecimalType with DoubleType
    useStringForDate = false) // true to replace DateType with StringType

tables.genData(
    location = rootDir,
    format = fileFormat,
    overwrite = true, // overwrite the data that is already there
    partitionTables = false, // create the partitioned fact tables
    clusterByPartitionColumns = false, // shuffle to get partitions coalesced into single files.
    filterOutNullPartitionValues = false, // true to filter out the partition with NULL key value
    tableFilter = "", // "" means generate all tables
    numPartitions = 4) // how many dsdgen partitions to run - number of input tasks.

// Create the specified database
sql(s"create database $databaseName")

// Create the specified database
sql(s"create database $databaseName")

// Create metastore tables in a specified database for your data.
// Once tables are created, the current database will be switched to the specified database.
tables.createExternalTables(rootDir, fileFormat, databaseName, overwrite
```

```
    = true, discoverPartitions = true)

// Or, if you want to create temporary tables
// tables.createTemporaryTables(location, fileFormat)

// For Cost-based optimizer (CBO) only, gather statistics on all columns:
tables.analyzeTables(databaseName, analyzeColumns = true)
```

Listing A.2: Databricks Notebook TPC-DS Generate Data

```
-- start query 10 in stream 0 using template query10.tpl
SELECT  cd_gender,
                cd_marital_status,
                cd_education_status,
                Count(*) cnt1,
                cd_purchase_estimate,
                Count(*) cnt2,
                cd_credit_rating,
                Count(*) cnt3,
                cd_dep_count,
                Count(*) cnt4,
                cd_dep_employed_count,
                Count(*) cnt5,
                cd_dep_college_count,
                Count(*) cnt6
FROM    customer c,
        customer_address ca,
        customer_demographics
WHERE   c.c_current_addr_sk = ca.ca_address_sk
        AND ca_county IN ( 'Lycoming County', 'Sheridan County',
                           'Kandiyohi County',
                           'Pike County',
                                              'Greene County' )
        AND cd_demo_sk = c.c_current_cdemo_sk
        AND EXISTS (SELECT *
                    FROM   store_sales,
                           date_dim
                    WHERE  c.c_customer_sk = ss_customer_sk
                           AND ss_sold_date_sk = d_date_sk
                           AND d_year = 2002
                           AND d_moy BETWEEN 4 AND 4 + 3)
        AND ( EXISTS (SELECT *
                      FROM   web_sales,
                             date_dim
                      WHERE  c.c_customer_sk = ws_bill_customer_sk
                             AND ws_sold_date_sk = d_date_sk
                             AND d_year = 2002
                             AND d_moy BETWEEN 4 AND 4 + 3)
              OR EXISTS (SELECT *
                         FROM   catalog_sales,
                                date_dim
                         WHERE  c.c_customer_sk = cs_ship_customer_sk
                                AND cs_sold_date_sk = d_date_sk
                                AND d_year = 2002
```

```
                                  AND d_moy BETWEEN 4 AND 4 + 3) )
GROUP  BY cd_gender,
          cd_marital_status,
          cd_education_status,
          cd_purchase_estimate,
          cd_credit_rating,
          cd_dep_count,
          cd_dep_employed_count,
          cd_dep_college_count
ORDER  BY cd_gender,
          cd_marital_status,
          cd_education_status,
          cd_purchase_estimate,
          cd_credit_rating,
          cd_dep_count,
          cd_dep_employed_count,
          cd_dep_college_count
LIMIT 100;

INSERT INTO ma_profile_final
SELECT * FROM information_schema.profiling
where QUERY_ID > 5 and QUERY_ID < 11;
-- start query 11 in stream 0 using template query11.tpl
WITH year_total
     AS (SELECT c_customer_id                                    customer_id,
                c_first_name
    customer_first_name
                ,
                c_last_name
                customer_last_name,
                c_preferred_cust_flag
                   customer_preferred_cust_flag
                    ,
                c_birth_country
                    customer_birth_country,
                c_login
    customer_login,
                c_email_address
                customer_email_address,
                d_year                                         dyear,
                Sum(ss_ext_list_price - ss_ext_discount_amt) year_total,
                's'                                            sale_type
         FROM   customer,
                store_sales,
                date_dim
         WHERE  c_customer_sk = ss_customer_sk
                AND ss_sold_date_sk = d_date_sk
         GROUP  BY c_customer_id,
                   c_first_name,
                   c_last_name,
                   c_preferred_cust_flag,
                   c_birth_country,
                   c_login,
```

```
                  c_email_address,
                  d_year
        UNION ALL
        SELECT c_customer_id                                    customer_id,
               c_first_name
   customer_first_name
               ,
               c_last_name
               customer_last_name,
               c_preferred_cust_flag
               customer_preferred_cust_flag
               ,
               c_birth_country
               customer_birth_country,
               c_login
   customer_login,
               c_email_address
               customer_email_address,
               d_year                                           dyear,
               Sum(ws_ext_list_price - ws_ext_discount_amt)     year_total,
               'w'                                              sale_type
        FROM   customer,
               web_sales,
               date_dim
        WHERE  c_customer_sk = ws_bill_customer_sk
               AND ws_sold_date_sk = d_date_sk
        GROUP  BY c_customer_id,
                  c_first_name,
                  c_last_name,
                  c_preferred_cust_flag,
                  c_birth_country,
                  c_login,
                  c_email_address,
                  d_year)
SELECT t_s_secyear.customer_id,
               t_s_secyear.customer_first_name,
               t_s_secyear.customer_last_name,
               t_s_secyear.customer_birth_country
FROM   year_total t_s_firstyear,
       year_total t_s_secyear,
       year_total t_w_firstyear,
       year_total t_w_secyear
WHERE  t_s_secyear.customer_id = t_s_firstyear.customer_id
       AND t_s_firstyear.customer_id = t_w_secyear.customer_id
       AND t_s_firstyear.customer_id = t_w_firstyear.customer_id
       AND t_s_firstyear.sale_type = 's'
       AND t_w_firstyear.sale_type = 'w'
       AND t_s_secyear.sale_type = 's'
       AND t_w_secyear.sale_type = 'w'
       AND t_s_firstyear.dyear = 2001
       AND t_s_secyear.dyear = 2001 + 1
       AND t_w_firstyear.dyear = 2001
       AND t_w_secyear.dyear = 2001 + 1
```

```
147         AND t_s_firstyear.year_total > 0
148         AND t_w_firstyear.year_total > 0
149         AND CASE
150               WHEN t_w_firstyear.year_total > 0 THEN t_w_secyear.
    year_total /
151                                                     t_w_firstyear.
    year_total
152               ELSE 0.0
153             END > CASE
154                     WHEN t_s_firstyear.year_total > 0 THEN
155                     t_s_secyear.year_total /
156                     t_s_firstyear.year_total
157                     ELSE 0.0
158                   END
159 ORDER  BY t_s_secyear.customer_id,
160           t_s_secyear.customer_first_name,
161           t_s_secyear.customer_last_name,
162           t_s_secyear.customer_birth_country
163 LIMIT 100;
164
165
166 -- start query 12 in stream 0 using template query12.tpl
167 SELECT
168          i_item_id ,
169          i_item_desc ,
170          i_category ,
171          i_class ,
172          i_current_price ,
173          Sum(ws_ext_sales_price)
                         AS itemrevenue ,
174          Sum(ws_ext_sales_price)*100/Sum(Sum(ws_ext_sales_price)) OVER (
    partition BY i_class) AS revenueratio
175 FROM     web_sales ,
176          item ,
177          date_dim
178 WHERE    ws_item_sk = i_item_sk
179 AND      i_category IN ('Home',
180                         'Men',
181                         'Women')
182 AND      ws_sold_date_sk = d_date_sk
183 AND      d_date BETWEEN Cast('2000-05-11' AS DATE) AND      (
184                   Cast('2000-05-11' AS DATE) + INTERVAL '30' day)
185 GROUP BY i_item_id ,
186          i_item_desc ,
187          i_category ,
188          i_class ,
189          i_current_price
190 ORDER BY i_category ,
191          i_class ,
192          i_item_id ,
193          i_item_desc ,
194          revenueratio
195 LIMIT 100;
```

```
196
197
198 -- start query 13 in stream 0 using template query13.tpl
199 SELECT Avg(ss_quantity),
200        Avg(ss_ext_sales_price),
201        Avg(ss_ext_wholesale_cost),
202        Sum(ss_ext_wholesale_cost)
203 FROM   store_sales,
204        store,
205        customer_demographics,
206        household_demographics,
207        customer_address,
208        date_dim
209 WHERE  s_store_sk = ss_store_sk
210        AND ss_sold_date_sk = d_date_sk
211        AND d_year = 2001
212        AND ( ( ss_hdemo_sk = hd_demo_sk
213                AND cd_demo_sk = ss_cdemo_sk
214                AND cd_marital_status = 'U'
215                AND cd_education_status = 'Advanced Degree'
216                AND ss_sales_price BETWEEN 100.00 AND 150.00
217                AND hd_dep_count = 3 )
218               OR ( ss_hdemo_sk = hd_demo_sk
219                    AND cd_demo_sk = ss_cdemo_sk
220                    AND cd_marital_status = 'M'
221                    AND cd_education_status = 'Primary'
222                    AND ss_sales_price BETWEEN 50.00 AND 100.00
223                    AND hd_dep_count = 1 )
224               OR ( ss_hdemo_sk = hd_demo_sk
225                    AND cd_demo_sk = ss_cdemo_sk
226                    AND cd_marital_status = 'D'
227                    AND cd_education_status = 'Secondary'
228                    AND ss_sales_price BETWEEN 150.00 AND 200.00
229                    AND hd_dep_count = 1 ) )
230        AND ( ( ss_addr_sk = ca_address_sk
231                AND ca_country = 'United States'
232                AND ca_state IN ( 'AZ', 'NE', 'IA' )
233                AND ss_net_profit BETWEEN 100 AND 200 )
234               OR ( ss_addr_sk = ca_address_sk
235                    AND ca_country = 'United States'
236                    AND ca_state IN ( 'MS', 'CA', 'NV' )
237                    AND ss_net_profit BETWEEN 150 AND 300 )
238               OR ( ss_addr_sk = ca_address_sk
239                    AND ca_country = 'United States'
240                    AND ca_state IN ( 'GA', 'TX', 'NJ' )
241                    AND ss_net_profit BETWEEN 50 AND 250 ) );
242
243
244 -- start query 14 in stream 0 using template query14.tpl
245 WITH cross_items
246      AS (SELECT i_item_sk ss_item_sk
247          FROM   item,
248                 (SELECT iss.i_brand_id    brand_id,
```

```
                       iss.i_class_id    class_id,
                       iss.i_category_id category_id
                FROM   store_sales,
                       item iss,
                       date_dim d1
                WHERE  ss_item_sk = iss.i_item_sk
                       AND ss_sold_date_sk = d1.d_date_sk
                       AND d1.d_year BETWEEN 1999 AND 1999 + 2
                INTERSECT
                SELECT ics.i_brand_id,
                       ics.i_class_id,
                       ics.i_category_id
                FROM   catalog_sales,
                       item ics,
                       date_dim d2
                WHERE  cs_item_sk = ics.i_item_sk
                       AND cs_sold_date_sk = d2.d_date_sk
                       AND d2.d_year BETWEEN 1999 AND 1999 + 2
                INTERSECT
                SELECT iws.i_brand_id,
                       iws.i_class_id,
                       iws.i_category_id
                FROM   web_sales,
                       item iws,
                       date_dim d3
                WHERE  ws_item_sk = iws.i_item_sk
                       AND ws_sold_date_sk = d3.d_date_sk
                       AND d3.d_year BETWEEN 1999 AND 1999 + 2) x
        WHERE  i_brand_id = brand_id
               AND i_class_id = class_id
               AND i_category_id = category_id),
    avg_sales
    AS (SELECT Avg(quantity * list_price) average_sales
        FROM   (SELECT ss_quantity   quantity,
                       ss_list_price list_price
                FROM   store_sales,
                       date_dim
                WHERE  ss_sold_date_sk = d_date_sk
                       AND d_year BETWEEN 1999 AND 1999 + 2
                UNION ALL
                SELECT cs_quantity   quantity,
                       cs_list_price list_price
                FROM   catalog_sales,
                       date_dim
                WHERE  cs_sold_date_sk = d_date_sk
                       AND d_year BETWEEN 1999 AND 1999 + 2
                UNION ALL
                SELECT ws_quantity   quantity,
                       ws_list_price list_price
                FROM   web_sales,
                       date_dim
                WHERE  ws_sold_date_sk = d_date_sk
                       AND d_year BETWEEN 1999 AND 1999 + 2) x)
```

```
SELECT  *
FROM    (SELECT 'store'                            channel,
                i_brand_id,
                i_class_id,
                i_category_id,
                Sum(ss_quantity * ss_list_price) sales,
                Count(*)                         number_sales
         FROM   store_sales,
                item,
                date_dim
         WHERE  ss_item_sk IN (SELECT ss_item_sk
                               FROM   cross_items)
                AND ss_item_sk = i_item_sk
                AND ss_sold_date_sk = d_date_sk
                AND d_week_seq = (SELECT d_week_seq
                                  FROM   date_dim
                                  WHERE  d_year = 1999 + 1
                                         AND d_moy = 12
                                         AND d_dom = 25)
         GROUP  BY i_brand_id,
                   i_class_id,
                   i_category_id
         HAVING Sum(ss_quantity * ss_list_price) > (SELECT average_sales
                                                    FROM   avg_sales))
    this_year,
        (SELECT 'store'                            channel,
                i_brand_id,
                i_class_id,
                i_category_id,
                Sum(ss_quantity * ss_list_price) sales,
                Count(*)                         number_sales
         FROM   store_sales,
                item,
                date_dim
         WHERE  ss_item_sk IN (SELECT ss_item_sk
                               FROM   cross_items)
                AND ss_item_sk = i_item_sk
                AND ss_sold_date_sk = d_date_sk
                AND d_week_seq = (SELECT d_week_seq
                                  FROM   date_dim
                                  WHERE  d_year = 1999
                                         AND d_moy = 12
                                         AND d_dom = 25)
         GROUP  BY i_brand_id,
                   i_class_id,
                   i_category_id
         HAVING Sum(ss_quantity * ss_list_price) > (SELECT average_sales
                                                    FROM   avg_sales))
    last_year
WHERE   this_year.i_brand_id = last_year.i_brand_id
        AND this_year.i_class_id = last_year.i_class_id
        AND this_year.i_category_id = last_year.i_category_id
ORDER   BY this_year.channel,
```

```
353           this_year.i_brand_id,
354           this_year.i_class_id,
355           this_year.i_category_id
356 LIMIT 100;
```

Listing A.3: TPC-DS MySQL Queries 10 - 14

```
1
2  -- 10
3  -- Start query 10 in stream 0 using template query10.tpl
4
5  -- Main Query
6  SELECT
7      cd_gender,
8      cd_marital_status,
9      cd_education_status,
10     COUNT(*) AS cnt1,
11     cd_purchase_estimate,
12     COUNT(*) AS cnt2,
13     cd_credit_rating,
14     COUNT(*) AS cnt3,
15     cd_dep_count,
16     COUNT(*) AS cnt4,
17     cd_dep_employed_count,
18     COUNT(*) AS cnt5,
19     cd_dep_college_count,
20     COUNT(*) AS cnt6
21 FROM
22     customer c
23 JOIN
24     customer_address ca ON c.c_current_addr_sk = ca.ca_address_sk
25 JOIN
26     customer_demographics cd ON cd.cd_demo_sk = c.c_current_cdemo_sk
27 WHERE
28     ca.ca_county IN ('Lycoming County', 'Sheridan County', 'Kandiyohi
       County', 'Pike County', 'Greene County')
29     AND EXISTS (
30         SELECT *
31         FROM store_sales ss
32         JOIN date_dim d ON ss.ss_sold_date_sk = d.d_date_sk
33         WHERE c.c_customer_sk = ss.ss_customer_sk
34         AND d.d_year = 2002
35         AND d.d_moy BETWEEN 4 AND 4 + 3
36     )
37     AND (
38         EXISTS (
39             SELECT *
40             FROM web_sales ws
41             JOIN date_dim d ON ws.ws_sold_date_sk = d.d_date_sk
42             WHERE c.c_customer_sk = ws.ws_bill_customer_sk
43             AND d.d_year = 2002
44             AND d.d_moy BETWEEN 4 AND 4 + 3
45         )
46         OR EXISTS (
```

```
            SELECT *
            FROM catalog_sales cs
            JOIN date_dim d ON cs.cs_sold_date_sk = d.d_date_sk
            WHERE c.c_customer_sk = cs.cs_ship_customer_sk
            AND d.d_year = 2002
            AND d.d_moy BETWEEN 4 AND 4 + 3
        )
    )
GROUP BY
    cd_gender,
    cd_marital_status,
    cd_education_status,
    cd_purchase_estimate,
    cd_credit_rating,
    cd_dep_count,
    cd_dep_employed_count,
    cd_dep_college_count
ORDER BY
    cd_gender,
    cd_marital_status,
    cd_education_status,
    cd_purchase_estimate,
    cd_credit_rating,
    cd_dep_count,
    cd_dep_employed_count,
    cd_dep_college_count
LIMIT 100;

-- 11
-- Start query 11 in stream 0 using template query11.tpl

-- Common Table Expression (CTE) to calculate yearly totals
WITH year_total AS (
    SELECT
        c_customer_id AS customer_id,
        c_first_name AS customer_first_name,
        c_last_name AS customer_last_name,
        c_preferred_cust_flag AS customer_preferred_cust_flag,
        c_birth_country AS customer_birth_country,
        c_login AS customer_login,
        c_email_address AS customer_email_address,
        d_year AS dyear,
        SUM(ss_ext_list_price - ss_ext_discount_amt) AS year_total,
        's' AS sale_type
    FROM
        customer
    JOIN
        store_sales ON c_customer_sk = ss_customer_sk
    JOIN
        date_dim ON ss_sold_date_sk = d_date_sk
    GROUP BY
        c_customer_id,
        c_first_name,
```

```
100         c_last_name,
101         c_preferred_cust_flag,
102         c_birth_country,
103         c_login,
104         c_email_address,
105         d_year
106     UNION ALL
107     SELECT
108         c_customer_id AS customer_id,
109         c_first_name AS customer_first_name,
110         c_last_name AS customer_last_name,
111         c_preferred_cust_flag AS customer_preferred_cust_flag,
112         c_birth_country AS customer_birth_country,
113         c_login AS customer_login,
114         c_email_address AS customer_email_address,
115         d_year AS dyear,
116         SUM(ws_ext_list_price - ws_ext_discount_amt) AS year_total,
117         'w' AS sale_type
118     FROM
119         customer
120     JOIN
121         web_sales ON c_customer_sk = ws_bill_customer_sk
122     JOIN
123         date_dim ON ws_sold_date_sk = d_date_sk
124     GROUP BY
125         c_customer_id,
126         c_first_name,
127         c_last_name,
128         c_preferred_cust_flag,
129         c_birth_country,
130         c_login,
131         c_email_address,
132         d_year
133 )
134
135 -- Main Query
136 SELECT
137     t_s_secyear.customer_id,
138     t_s_secyear.customer_first_name,
139     t_s_secyear.customer_last_name,
140     t_s_secyear.customer_birth_country
141 FROM
142     year_total t_s_firstyear
143 JOIN
144     year_total t_s_secyear ON t_s_secyear.customer_id = t_s_firstyear.
    customer_id
145 JOIN
146     year_total t_w_firstyear ON t_s_firstyear.customer_id = t_w_firstyear
    .customer_id
147 JOIN
148     year_total t_w_secyear ON t_s_firstyear.customer_id = t_w_secyear.
    customer_id
149 WHERE
```

```
    t_s_firstyear.sale_type = 's'
    AND t_w_firstyear.sale_type = 'w'
    AND t_s_secyear.sale_type = 's'
    AND t_w_secyear.sale_type = 'w'
    AND t_s_firstyear.dyear = 2001
    AND t_s_secyear.dyear = 2001 + 1
    AND t_w_firstyear.dyear = 2001
    AND t_w_secyear.dyear = 2001 + 1
    AND t_s_firstyear.year_total > 0
    AND t_w_firstyear.year_total > 0
    AND (
        CASE
            WHEN t_w_firstyear.year_total > 0 THEN t_w_secyear.year_total / t_w_firstyear.year_total
            ELSE 0.0
        END > CASE
            WHEN t_s_firstyear.year_total > 0 THEN t_s_secyear.year_total / t_s_firstyear.year_total
            ELSE 0.0
        END
    )
ORDER BY
    t_s_secyear.customer_id,
    t_s_secyear.customer_first_name,
    t_s_secyear.customer_last_name,
    t_s_secyear.customer_birth_country
LIMIT 100;

--12
-- Start query 12 in stream 0 using template query12.tpl

SELECT
    i_item_id,
    i_item_desc,
    i_category,
    i_class,
    i_current_price,
    SUM(ws_ext_sales_price) AS itemrevenue,
    SUM(ws_ext_sales_price) * 100 / SUM(SUM(ws_ext_sales_price)) OVER (PARTITION BY i_class) AS revenueratio
FROM
    web_sales
JOIN
    item ON ws_item_sk = i_item_sk
JOIN
    date_dim ON ws_sold_date_sk = d_date_sk
WHERE
    i_category IN ('Home', 'Men', 'Women')
    AND d_date BETWEEN CAST('2000-05-11' AS DATE) AND (CAST('2000-05-11' AS DATE) + INTERVAL '30' DAY)
GROUP BY
    i_item_id,
    i_item_desc,
```

```
199     i_category,
200     i_class,
201     i_current_price
202 ORDER BY
203     i_category,
204     i_class,
205     i_item_id,
206     i_item_desc,
207     revenueratio
208 LIMIT 100;
209 
210 --13
211 -- Start query 13 in stream 0 using template query13.tpl
212 
213 SELECT
214     AVG(ss_quantity),
215     AVG(ss_ext_sales_price),
216     AVG(ss_ext_wholesale_cost),
217     SUM(ss_ext_wholesale_cost)
218 FROM
219     store_sales
220 JOIN
221     store ON s_store_sk = ss_store_sk
222 JOIN
223     customer_demographics ON cd_demo_sk = ss_cdemo_sk
224 JOIN
225     household_demographics ON hd_demo_sk = ss_hdemo_sk
226 JOIN
227     customer_address ON ss_addr_sk = ca_address_sk
228 JOIN
229     date_dim ON ss_sold_date_sk = d_date_sk
230 WHERE
231     d_year = 2001
232     AND (
233         (ss_hdemo_sk = hd_demo_sk
234         AND cd_marital_status = 'U'
235         AND cd_education_status = 'Advanced Degree'
236         AND ss_sales_price BETWEEN 100.00 AND 150.00
237         AND hd_dep_count = 3)
238         OR
239         (ss_hdemo_sk = hd_demo_sk
240         AND cd_marital_status = 'M'
241         AND cd_education_status = 'Primary'
242         AND ss_sales_price BETWEEN 50.00 AND 100.00
243         AND hd_dep_count = 1)
244         OR
245         (ss_hdemo_sk = hd_demo_sk
246         AND cd_marital_status = 'D'
247         AND cd_education_status = 'Secondary'
248         AND ss_sales_price BETWEEN 150.00 AND 200.00
249         AND hd_dep_count = 1)
250     )
251     AND (
```

```
        (ss_addr_sk = ca_address_sk
        AND ca_country = 'United States'
        AND ca_state IN ('AZ', 'NE', 'IA')
        AND ss_net_profit BETWEEN 100 AND 200)
        OR
        (ss_addr_sk = ca_address_sk
        AND ca_country = 'United States'
        AND ca_state IN ('MS', 'CA', 'NV')
        AND ss_net_profit BETWEEN 150 AND 300)
        OR
        (ss_addr_sk = ca_address_sk
        AND ca_country = 'United States'
        AND ca_state IN ('GA', 'TX', 'NJ')
        AND ss_net_profit BETWEEN 50 AND 250)
    );

--14
-- Start query 14 in stream 0 using template query14.tpl
WITH cross_items AS (
    SELECT DISTINCT i_item_sk AS ss_item_sk
    FROM item
    WHERE i_brand_id IN (
        SELECT iss.i_brand_id
        FROM store_sales
        JOIN item iss ON store_sales.ss_item_sk = iss.i_item_sk
        JOIN date_dim d1 ON store_sales.ss_sold_date_sk = d1.d_date_sk
        WHERE d1.d_year BETWEEN 1999 AND 2001
        UNION ALL
        SELECT ics.i_brand_id
        FROM catalog_sales
        JOIN item ics ON catalog_sales.cs_item_sk = ics.i_item_sk
        JOIN date_dim d2 ON catalog_sales.cs_sold_date_sk = d2.d_date_sk
        WHERE d2.d_year BETWEEN 1999 AND 2001
        UNION ALL
        SELECT iws.i_brand_id
        FROM web_sales
        JOIN item iws ON web_sales.ws_item_sk = iws.i_item_sk
        JOIN date_dim d3 ON web_sales.ws_sold_date_sk = d3.d_date_sk
        WHERE d3.d_year BETWEEN 1999 AND 2001
    )
), avg_sales AS (
    SELECT AVG(quantity * list_price) AS average_sales
    FROM (
        SELECT ss_quantity AS quantity, ss_list_price AS list_price
        FROM store_sales
        JOIN date_dim ON store_sales.ss_sold_date_sk = date_dim.d_date_sk
        WHERE date_dim.d_year BETWEEN 1999 AND 2001
        UNION ALL
        SELECT cs_quantity AS quantity, cs_list_price AS list_price
        FROM catalog_sales
        JOIN date_dim ON catalog_sales.cs_sold_date_sk = date_dim.
    d_date_sk
        WHERE date_dim.d_year BETWEEN 1999 AND 2001
```

```
304         UNION ALL
305         SELECT ws_quantity AS quantity, ws_list_price AS list_price
306         FROM web_sales
307         JOIN date_dim ON web_sales.ws_sold_date_sk = date_dim.d_date_sk
308         WHERE date_dim.d_year BETWEEN 1999 AND 2001
309     ) x
310 )
311 SELECT
312     channel,
313     i_brand_id,
314     i_class_id,
315     i_category_id,
316     SUM(sales),
317     SUM(number_sales)
318 FROM (
319     SELECT
320         'store' AS channel,
321         item.i_brand_id,
322         item.i_class_id,
323         item.i_category_id,
324         SUM(ss_quantity * ss_list_price) AS sales,
325         COUNT(*) AS number_sales
326     FROM store_sales
327     JOIN item ON store_sales.ss_item_sk = item.i_item_sk
328     JOIN date_dim ON store_sales.ss_sold_date_sk = date_dim.d_date_sk
329     WHERE item.i_item_sk IN (SELECT ss_item_sk FROM cross_items)
330         AND date_dim.d_year = 2001
331         AND date_dim.d_moy = 11
332     GROUP BY item.i_brand_id, item.i_class_id, item.i_category_id
333     HAVING SUM(ss_quantity * ss_list_price) > (SELECT average_sales FROM
    avg_sales)
334     UNION ALL
335     SELECT
336         'catalog' AS channel,
337         item.i_brand_id,
338         item.i_class_id,
339         item.i_category_id,
340         SUM(cs_quantity * cs_list_price) AS sales,
341         COUNT(*) AS number_sales
342     FROM catalog_sales
343     JOIN item ON catalog_sales.cs_item_sk = item.i_item_sk
344     JOIN date_dim ON catalog_sales.cs_sold_date_sk = date_dim.d_date_sk
345     WHERE item.i_item_sk IN (SELECT ss_item_sk FROM cross_items)
346         AND date_dim.d_year = 2001
347         AND date_dim.d_moy = 11
348     GROUP BY item.i_brand_id, item.i_class_id, item.i_category_id
349     HAVING SUM(cs_quantity * cs_list_price) > (SELECT average_sales FROM
    avg_sales)
350     UNION ALL
351     SELECT
352         'web' AS channel,
353         item.i_brand_id,
354         item.i_class_id,
```

```
        item.i_category_id,
        SUM(ws_quantity * ws_list_price) AS sales,
        COUNT(*) AS number_sales
    FROM web_sales
    JOIN item ON web_sales.ws_item_sk = item.i_item_sk
    JOIN date_dim ON web_sales.ws_sold_date_sk = date_dim.d_date_sk
    WHERE item.i_item_sk IN (SELECT ss_item_sk FROM cross_items)
        AND date_dim.d_year = 2001
        AND date_dim.d_moy = 11
    GROUP BY item.i_brand_id, item.i_class_id, item.i_category_id
    HAVING SUM(ws_quantity * ws_list_price) > (SELECT average_sales FROM
    avg_sales)
) y
GROUP BY ROLLUP(channel, i_brand_id, i_class_id, i_category_id)
ORDER BY channel, i_brand_id, i_class_id, i_category_id
LIMIT 100;
```

Listing A.4: TPC-DS Hive Queries 10 - 14

```
#!/bin/bash

mysql -h mafuehles.mysql.database.azure.com -u t_fuehles -
    p12Compexpert1994- -D ma -e "SELECT * FROM household_demographics" --
    batch --raw --skip-column-names | sed 's/\t/,/g' | iconv -f ISO-8859-1
     -t UTF-8 > household_demographics.csv
```

Listing A.5: Bash Script MySQL Table to UTF-8 csv

```
#!/bin/bash

# Define the filename variable
file="$1"

# Check if the filename argument is provided
if [ -z "$file" ]; then
    echo "Usage: $0 <filename>"
    exit 1
fi

# Upload the CSV file to HDFS, replacing if it already exists
hdfs dfs -put -f "$file" /test/test/

# List the files in the destination directory
hdfs dfs -ls /test/test

# Change the permissions of the uploaded file
chmod 777 "$file"
```

Listing A.6: Bash Script Move

```

#!/bin/bash

# Iterate over all CSV files in the current directory
```

```
for file in *.csv; do
    # Check if the file exists and is a regular file
    if [ -f "$file" ]; then
        # Execute the move.sh script for each CSV file
        ./move.sh "$file"
    fi
done
```

Listing A.7: Bash Script Iterate

```
USE ma;
-- 1

drop table customer_address;
CREATE TABLE customer_address (
    ca_address_sk             INTEGER                NOT NULL,
    ca_address_id             STRING                 NOT NULL,
    ca_street_number          STRING,
    ca_street_name            STRING,
    ca_street_type            STRING,
    ca_suite_number           STRING,
    ca_city                   STRING,
    ca_county                 STRING,
    ca_state                  STRING,
    ca_zip                    STRING,
    ca_country                STRING,
    ca_gmt_offset             DECIMAL(5,2),
    ca_location_type          STRING,
    PRIMARY KEY (ca_address_sk) DISABLE NOVALIDATE
)
ROW FORMAT DELIMITED
FIELDS TERMINATED BY ','
LINES TERMINATED BY '\n'
STORED AS TEXTFILE;

LOAD DATA INPATH '/test/test/customer_address.csv'
INTO TABLE customer_address;

-- 2
drop table customer_demographics ;
CREATE TABLE customer_demographics (
    cd_demo_sk                INTEGER                NOT NULL,
    cd_gender                 STRING,
    cd_marital_status         STRING,
    cd_education_status       STRING,
    cd_purchase_estimate      INTEGER,
    cd_credit_rating          STRING,
    cd_dep_count              INTEGER,
    cd_dep_employed_count     INTEGER,
    cd_dep_college_count      INTEGER,
    PRIMARY KEY (cd_demo_sk) DISABLE NOVALIDATE
)
ROW FORMAT DELIMITED
FIELDS TERMINATED BY ','
```

```
LINES TERMINATED BY '\n'
STORED AS TEXTFILE;

LOAD DATA INPATH '/test/test/customer_demographics.csv'
INTO TABLE customer_demographics;

-- 3
drop table date_dim;
CREATE TABLE date_dim (
    d_date_sk                 INTEGER               NOT NULL,
    d_date_id                 STRING                NOT NULL,
    d_date                    DATE,
    d_month_seq               INTEGER,
    d_week_seq                INTEGER,
    d_quarter_seq             INTEGER,
    d_year                    INTEGER,
    d_dow                     INTEGER,
    d_moy                     INTEGER,
    d_dom                     INTEGER,
    d_qoy                     INTEGER,
    d_fy_year                 INTEGER,
    d_fy_quarter_seq          INTEGER,
    d_fy_week_seq             INTEGER,
    d_day_name                STRING,
    d_quarter_name            STRING,
    d_holiday                 STRING,
    d_weekend                 STRING,
    d_following_holiday       STRING,
    d_first_dom               INTEGER,
    d_last_dom                INTEGER,
    d_same_day_ly             INTEGER,
    d_same_day_lq             INTEGER,
    d_current_day             STRING,
    d_current_week            STRING,
    d_current_month           STRING,
    d_current_quarter         STRING,
    d_current_year            STRING,
    PRIMARY KEY (d_date_sk) DISABLE NOVALIDATE
)
ROW FORMAT DELIMITED
FIELDS TERMINATED BY ','
LINES TERMINATED BY '\n'
STORED AS TEXTFILE;


LOAD DATA INPATH '/test/test/date_dim.csv'
INTO TABLE date_dim;


-- 4
drop table warehouse;
CREATE TABLE warehouse (
    w_warehouse_sk            INTEGER               NOT NULL,
```

```
98      w_warehouse_id              STRING                  NOT NULL,
99      w_warehouse_name            STRING,
100     w_warehouse_sq_ft           INTEGER,
101     w_street_number             STRING,
102     w_street_name               STRING,
103     w_street_type               STRING,
104     w_suite_number              STRING,
105     w_city                      STRING,
106     w_county                    STRING,
107     w_state                     STRING,
108     w_zip                       STRING,
109     w_country                   STRING,
110     w_gmt_offset                DECIMAL(5,2),
111     PRIMARY KEY (w_warehouse_sk) DISABLE NOVALIDATE
112 )
113 ROW FORMAT DELIMITED
114 FIELDS TERMINATED BY ','
115 LINES TERMINATED BY '\n'
116 STORED AS TEXTFILE;
117
118 LOAD DATA INPATH '/test/test/warehouse.csv'
119 INTO TABLE warehouse;
```

Listing A.8: Hive Table Generation