UMEÅ UNIVERSITY

COMPARISON BETWEEN NOSQL AND SQL DATABASE SOLUTIONS FOR DICOM DATA

Hanna Holmgren

Bachelor, 15 hp/credits

Bachelor of Science in Computing Science

2025

Abstract

Internal imaging is a crucial part of healthcare to diagnose different conditions. DICOM has become the international standard for storing and transmitting medical images obtained by different internal imaging devices. These images are stored in DICOM files that cannot be opened and read as a regular text file. The files require special software to be viewed. Finding a specific file for a patient can therefore become a difficult task. A database management system can be used to easier locate the desired file. Traditionally, this is done with the use of a relational database, but there has been an increase in the use of NoSQL models in recent years. The benefit of using a document-based NoSQL model is its flexible schema and avoidance of join operations. Related information can be stored together in nested documents. This thesis compare the relational MySQL with the document-based MongoDB in terms of retrieval time for different queries to determine which model is better suited for storing DICOM data. The result of the thesis shows that the decision between the two models for storing DICOM data is largely dependent on the queries performed on the database. Although MySQL performed better for some queries, MongoDB was faster at executing simple queries that do not need aggregation.

Acknowledgements

I would like to thank my family, friends, and partner for their support and encouragement during my studies.

Data used in this thesis were generated by the National Cancer Institute Cancer Moonshot Biobank.

Contents

1 Introduction

Internal imaging is a crucial part of the medical industry, both in terms of diagnostic procedures and in research. Some common techniques for obtaining internal images include computed tomography (CT), magnetic resonance imaging (MRI), and positron emission tomography (PET). An important aspect of medial research and healthcare is the possibility of transferring data obtained from a medical imaging device between workstations and research institutes. Digital Imaging Communications in Medicine (DICOM)[1] is the international standard for medical images and defines a process for transmitting, storing, and displaying medical imaging data. DICOM is incorporated into most medical imaging devices and other methods for handling medical images usually relies on the conversion of DICOM to other formats [7].

DICOM is not only the standard for communication, but also a file format with the extension `.dcm`. A DICOM file contains metadata related to the patient and the procedure performed, as well as the image obtained from the imaging device. A CT scan of a patient can produce hundreds of 2D images that together can make a 3D image of the internal body. Each 2D image is saved in their own DICOM file, which may result in an extensive amount of data when multiple scans are performed on a patient.

DICOM files cannot be opened or read like a regular text file. The file requires special software to read and view the content. There are several DICOM viewer software's on the market, including applications enabling medical image analysis. The applications used for imaging analysis can often be used to compare the progression of a tumour or to compare two patients' tumours. Since DICOM files cannot be read like a regular file, traversing the files in order to find specific studies or images can become a time-consuming task. With the use of a Database Management System (DBMS), the search for patient studies and series can be made easier and faster. When developing software for medical image analysis, it is important that the software uses a reliable DBMS to ensure an easy search of information. Picturing Archiving and Communication Systems (PACS) are used by hospitals and researchers to store and transmit large amounts of DICOM data. These systems traditionally use relational database systems [14].

However, in recent years, there has been an increase in the use of NoSQL (Not Only SQL) database systems [8]. There is little research comparing these two types of DBMS and their underlying structure for storing DICOM data. Unlike a relational model where there is a fixed design of how data is stored, NoSQL databases are known for their flexibility, allowing data to be stored without a predefined structure [9, 14].

A DICOM file follows strict rules in terms of how the data is structured to be a valid DICOM file. However, the files can contain different information depending on the type of device used to obtain an image and is not always the same. This variety within the DICOM files and the NoSQL databases ability to store data without a predefined structure, makes a NoSQL solution a possible alternative to the relational model when storing DICOM files.

This thesis will investigate which type of database system, SQL or NoSQL, is best suited regarding retrieval time for different types of queries when storing DICOM-data in software for medical image analysis. The thesis will compare the retrieval time for two different DBMSs,

[1]DICOM: https://www.dicomstandard.org

one based on the relational model, MySQL, and one using NoSQL, MongoDB, and their underlying structure to determine which DBMS is better suited.

2 Background

DICOM is a communication protocol to transmit, store, retrieve, print, and display medical imaging information between different devices. The protocol is described in an extensive standard consisting of 22 parts[2]. The following section will give a brief overview of the DICOM structure and its file format.

2.1 The DICOM Structure

A DICOM file contains data related to the procedure performed with a medical imaging device. The DICOM standard consists of abstract definitions of different procedures, which are described through entity models and modules, as well as real data objects called Data Elements that carry the actual data in the file. The DICOM hierarchy consists of the levels patient, study, series, and instance. The instance refers to the image obtained by a medical imaging device. A patient can have multiple studies where each study can have several series. Each series can, in turn, contain one or more instances [11]. Figure 1 shows the DICOM hierarchy.

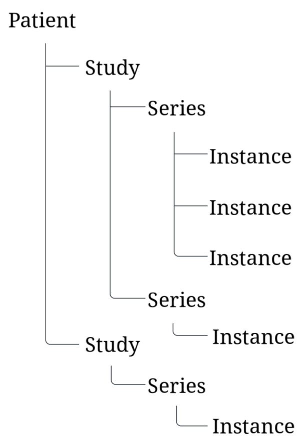

Figure 1: Shows the DICOM structure.

2.1.1 Information Object Definitions

The largest building block of the DICOM structure is the Information Object Definition (IOD) which defines a DICOM object. IODs are abstractions and describe real-world objects pertaining to different medical imaging procedures. The IOD lists the data required to be a valid DICOM object which can be interpreted by different modalities and workstations. The DICOM standard has an extensive list of the IODs, used to describe real-world objects. The IOD is constructed of Information Entities (IE) which describes the category of attributes needed to build a valid IOD. The IOD for a CT scan consists of the IEs `Patient`, `Study`, `Series`, `Equipment`, `Acquisition`, and `Frame of Reference`. The IEs in turn con-

[2]DICOM standard: https://www.dicomstandard.org/current

sist of Information Modules (IM). The IMs provide a way of organizing the smallest components of the DICOM structure, the Data Elements. Table 1 shows the IOD for a CT scan with its IEs, IMs and Data Elements. The IOD shown in Table 1 is not exhaustive and some Data Elements have been omitted for simplicity.

Table 1 Shows the IOD for a CT scan

CT image IOD		
Information Entity	**Information Module**	**Data Elements**
Patient	Patient	Patient's name Patient ID Patient's Birth Date Patient's Sex
Study	General Study	Study Instance UID Study Date Study Time Referring Physician's Name Study ID Accession Number
Series	General Series	Modality Series Instance UID Series Number Study Instance IUD Patient Position
Frame of Reference	Frame of Reference	Frame of Reference UID Position Reference Indicator
Equipment	General Equipment	Manufacturer
Acquisition	General Acquisition	Acquisition UID Acquisition Number
Image	General Image	Instance Number
	Image Plane	Pixel Spacing Image Orientation (Patient) Image Position (Patient) Slice Thickness
	Image Pixel	Pixel Data
	CT image	Image Type Samples Per Picture Photometric Interpretation Bits Allocated Bits Stored High Bits Rescale Intercept Rescale Slope Acquisition Number
	SOP Common	SOP Class UID SOP Instance UID

2.1.2 Data Elements and Information Modules

Data elements are the actual attributes needed to describe a real-world object. Different IODs can require the same IEs but different IMs and Data Elements. IMs can be mandatory, optional, or conditional. The IOD for a CT scan requires the IE `Patient` and the `Patient Module`. The Patient Module contains Data Elements such as name, patient ID, birth date, and sex, while other modules related to patients such as the Patient Medical Module contain Data Elements such as allergies, smoking status, pregnancy status, and other attributes. It is not possible to combine the Data Elements at random. Most DICOM devices will reject such arbitrary combinations of elements or will not know how to interpret them [11]. Therefore, the DICOM standard, which defines thousands of different Data Elements, combines them into Information Modules [11].

A Data Element consists partly of a Tag and a Value. Because of the large number of Data Elements defined in the standard, the elements themselves are divided into groups. These groups are loosely based on the similarity of the attributes [11]. Attributes relating to a patient such as name, smoking status, and date of birth has group number 10. The group is then organized by element number. The Tag in the Data Element describes the attributes group and element number and has the format (GGGG, EEEE) where the first part describes the group number, and the second part the element number within the group. The Tag is a unique identifier and is used in the DICOM file to identify an attribute. Figure 2 shows the structure of a Data Element with its Tag.

Tag	Value
(0010, 0010)	Jane Doe

Figure 2: Shows the Data Element for Patient's Name with its Tag and Value.

2.1.3 Pixel Data

Most IODs require an Image IE with a module that includes the Pixel Data attribute. The Pixel Data attribute contains the actual image obtained by the medical imaging device. Other attributes in modules pertaining to the Image IE describes how the pixel data should be interpreted. The attributes used to interpret the pixels include colour scale, size, and other relevant information used to reconstruct the image accurately.

2.2 Database Models

Since DICOM files cannot be opened and read without special software, building an application for medical imaging analysis requires the data to be stored in a database for easy access and structure. The most common database model is the relational model which uses Structured Query Language (SQL) to store, manipulate, and retrieve data [4]. In recent years, the NoSQL database model has gained traction due to its flexible nature and has become increasingly popular.

2.2.1 Relational Database Model

A relational database model consists of tables called relations with attributes that describe an object [4]. A relation with its attributes is called a schema and is well suited for storing structured data [12]. Structured data is characterized by its adherence to a well-defined data model and predefined data types. It is typically stored in tables, where each attribute has a set format [5]. The relational model uses an entity-relationship (E/R) model to describe how data is structured in the database [4]. The E/R-model has three elements, entities, attributes, and relationships [4]. Entities can be described as an abstract object and is something we store data about through attributes. For example, an entity could be a customer with the attributes name, customer ID, phone number, and address. Entities can be connected to each other through relationships. The customer entity could be connected to an item entity. The relationship between the customer and the entity of an item can describe the items a customer purchased. Table 2 shows an example of a table that contains information relating to a customer.

Table 2 Shows table for a customer.

Name	Customer ID	Phone Number	Address
Jane Doe	1	123456	Address 1
John Doe	2	987654	Address 2

One of the benefits of using a relation model is the possibility to divide data into small logical tables. This is called normalization and prevents data from being duplicated across multiple tables [12]. In order to access related data from different tables, a `JOIN` operation has to be performed. This results in a combined table where related information can be extracted [4].

Relational databases are static, which means that the schema of the tables and relationships are defined before any data is inserted [12]. Modifications to the schema after data insertion is possible, but such actions require consideration since it can lead to service failure and decreased performance if not done correctly [12].

2.2.2 NoSQL

The NoSQL database can be seen as a compliment to the relational model [9]. Unlike the schemas in the relational model, NoSQL uses a dynamic schema, which means that the schema does not have a fixed structure. The use of dynamic schemas gives the model greater flexibility and can change more easily over time as needs change [12]. Another advantage of the flexible schema is the possibility to store structured, semi-structured and unstructured data, something a relational model can struggle with [12]. Unstructured data does not have a set format and cannot be represented with a schema as structured data can [10]. Semi-structured data falls between the two other categories. The data may follow a schema but are subject to continuous change [5]. Different models of a NoSQL database store data differently, and their underlying architecture varies. The models are categorized according to the way they store and retrieve data. This thesis focuses on investigating the document-based model.

Document-based NoSQL Model

The document-based model stores data in the form of documents, as implied by the name. The model offers great flexibility when it comes to the format in which the data is stored. Different NoSQL models support different formats of documents, but formats such as XML, PDF, and JSON are usually supported [6]. The data can be stored hierarchically with related data embedded in one document, or it can be stored in different documents with references between them, much like a relational model. One of the most popular document-based models is MongoDB.

2.2.3 MongoDB

The documentation provided by MongoDB describes the components and structure of the database. A document is made up of field and value pairs. Figure 3 shows the basic structure of a document in MongoDB. The value can have a BSON data type such as double, string, object, null, etc., or consist of other documents, arrays, and arrays of documents. Figure 4 shows how documents can be embedded within another document or as an array of strings. Documents are gathered in collections that can be compared to tables in a relational database. The schema in MongoDB is flexible, and documents in a collection are not required to have the same set of fields. The schema in MongoDB can also be changed over time as the needs change. Due to the possibility of embedding related data in a document, there is no need to join documents, which can improve performance and reduce workload [12].

```
{
    field: value,
    field: value,
    field: value
}
```

Figure 3: Shows the structure of the document in MongoDB.

```
{
    name: {first: "Jane", last: "Doe"},
    pets: ["Milo", "Luna", "Nala"],
}
```

Figure 4: Shows how embedded documents and an array within a document.

Aggregation and Unwinding

A document containing other embedded documents is considered one single document. The `find()` operation is used to find documents that meet a condition. When the `find()` operation is performed on a collection of documents with a condition, the whole document that contains a value that matches the condition will be returned with all its embedded documents. In order to retrieve only the embedded document that meets the condition, an aggregation pipeline has to be used. An aggregation pipeline is defined by a sequence of operations called stages. Each stage performs an operation on the document, and the result can be processed by a new operation in the next stage [2]. To return only the embedded document, the array that holds the embedded documents has to be unwound with the help of the `unwind` operation. The unwind operation outputs a new document for each element in the array. This means that if a document has an array of five embedded documents and an unwind operation is performed, each element in the array gets its own document. This can be an expensive operation if the document contains a large array. After unwinding an array, a `match` operation can be performed to find the individual documents that meet the condition.

2.3 DICOM as Structured and Semi-structured Data

The DICOM files can be seen as both structured and semi-structured data. On the Data Element level, the content can be regarded as structured. Each Tag has a value of a certain data type. However, multiple DICOM files can contain different Data Elements depending on the medical imaging procedure, making them semi-structured at the file level [13]. It is important that the data storage system can accommodate both of these levels and support the adding of new Tags and still maintain good query performance.

3 Related Work

PACS used to store and transmit DICOM files usually uses relational database management systems [1]. Previous studies have investigated the use of NoSQL for storing DICOM data, both in PACS and other DICOM management systems. One of the biggest advantages of a NoSQL solution is the hierarchical structure and flexible schema, as well as the avoidance of `JOIN` operations, which can make querying faster [12].

[1] investigated MongoDB as an alternative to the relational database model. The study proposed a MongoDB architecture in which the workload was on one node or distributed over multiple nodes and compared their performance. The queries tested were "search all exams of patients by ID in which the patient exists" and "search for all exams of a patient by ID in which the patient does not exist". The results showed that the single node architecture was faster in querying the database than the distributed architecture. The slower query time was explained as a result of network overhead in the distributed architecture. The study still concluded that the NoSQL solution is a good solution to solve the problem of an increasing volume of medical imaging data. However, [1] did not investigate the two proposed architectures to a relational model and how they compare in terms of retrieval time, which is an important factor in deciding which database model is best suited for storing DICOM data.

Another study [13] introduced a NoSQL, row-store database, using Appache Cassandra. The row-store solution stores the data at tag-level, meaning that one tag from a DICOM file represents one row in the table. The row and tag are then identified with the use of different attributes such as `studyInstanceUID`, `seriesInstanceUID` or `patientID`. For all DICOM files, all tags from the files are saved in their own row. The study evaluated the row-store approach to a relational model. Overall, the relational approach performed better and was faster than the NoSQL solution, although the authors do argue that the flexibility of the row-based model makes it a possible option for storing DICOM data at the tag level. The authors did conclude that their experimental setup and the chosen NoSQL approach, as well as the data model is an unfeasible option for storing DICOM data at the tag level. By redefining the data model, the results may be different.

Other studies have also compared the general performance of SQL and NoSQL databases for storing different types of data.

[8] compared read, write, and instantiate operations on several NoSQL solutions with a SQL solution. The results showed that not all NoSQL solutions performed better than the SQL one. The different NoSQL databases were both column-based and document-based, but there was no observable correlation between the data model and performance. In the case of MongoDB and MS SQL Express, MongoDB outperformed the SQL version in terms of time for reading, 23 vs. 46 ms for 100 operations. The SQL database had better read performance than some of the NoSQL but not all of them.

When comparing MongoDB with SQL Server in [10] with a modest-sized, structured database, the results showed that MongoDB had a better runtime performance for inserting, updating, and simple queries, while the SQL solution performed better for aggregated queries.

A final study [3] compared the retrieval of data from a Covid-19 genome data set and showed that MongoDB performed well compared to the SQL solution. For all queries, the SQL solution took longer time to query the database, possibly due to the need for expensive JOIN operations on the relational model.

The related work indicates that a NoSQL solution often outperforms the SQL model in terms of data retrieval execution time, both for structured and unstructured data. The attempts made to store and retrieve DICOM data using a NoSQL model also show potential to challenge the relational model used in PACS.

However, previous studies regarding DICOM data have not compared a document-based NoSQL with SQL solutions in regard to retrieval time. Previous studies investigating NoSQL for DICOM data have mainly compared different NoSQL models to each other [1] [14]. Therefore, there is a gap in understanding how the underlying data structure of these databases affects the storing of DICOM data. The natural hierarchy of DICOM data can easily be replicated in the document-based NoSQL models but previous studies have indicated that aggregated operations on document-based based models have a higher retrieval time. This thesis aims to investigate how the data structure in the different models differs when storing DICOM data and how this affects retrieval time.

4 Methodology

In order to investigate which database model is best suited in terms of retrieval time for storing DICOM data when developing software for medical image analysis, a comparison between MySQL and MongoDB is made. MongoDB is tested using the Community Edition that allows work with local databases but does not support cloud storage. The Community Edition is free and open-source. Previous studies [14] have compared the performance of different document-based NoSQL models and have observed similar performance in storing and retrieving DICOM files. [14] concluded that the differences in architecture between document-based models are insignificant and their performances are similar. This thesis uses the MongoDB database, but other document-based models would probably have yielded similar results. MySQL was chosen as the relational database due to their popularity and common usage in many industries.

4.1 Data

The dataset used to compare the two database models is from the Cancer Imaging Archive, which has a wide range of publicly available DICOM datasets. The dataset[3] contains images from PET, MRI, and CT scans performed on five different patients during 22 studies and 124 series, totalling 22 321 images.

4.2 Pydicom

Pydicom is a Python package that can be used to read the content of DICOM file. Pydicom is a tool developed to read, modify, and write DICOM data. By combining Pydicom with a MySQL connector and a MongoDB connector, specific tags in the DICOM file can be extracted and entered into the two databases.

4.3 MySQL

For the relational database model, MySQL, is used with MySQL Workbench to obtain a graphical interface of the database. The database consists of the four tables Patient, Study, Series, and Instance. The schema for the different tables can be seen in Figure 5. The division of tables follows the standard for how DICOM objects are separated and organized. The first level of organization is the patient, then comes the study, then series, and finally instances. The tables are connected to each other through relationships. The attributes in the tables correspond to the Data Elements in the DICOM file. Since there are thousands of possible Data Elements in the standard, only a few and the most common ones have been chosen to be a part of the schema. These attributes are chosen with the work process in mind, where a user might want

[3] Cancer Moonshot Biobank. (2022). Cancer Moonshot Biobank – Gastroesophageal Cancer Collection (CMB-GEC) (Version 6) [dataset]. The Cancer Imaging Archive. https://doi.org/10.7937/e7kh-r486

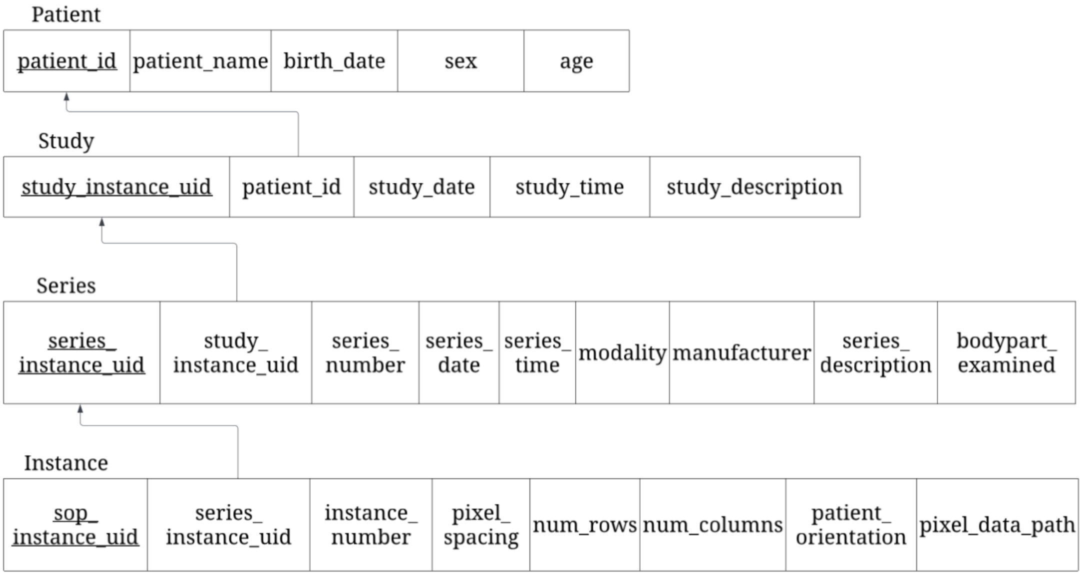

Figure 5: The schema for the relational data base.

to view images of all patients receiving a CT scan between two dates. The purpose of having a database to store DICOM information is to simplify the search for different patients, studies, or series without having to go through thousands of files. Depending on the purpose of the software for medical imaging analysis, different attributes could be chosen or added to support the needs. If other attributes are needed from a DICOM file that is not part of the database, the instance table has the attribute `pixel data path` that contains the file path to the storage place for the complete DICOM file. The image obtained by a device is not stored directly in the database due to the size limitation of the database.

4.4 MongoDB

For the NoSQL database, MongoDB Community Edition is used with Compass as the graphical interface. MongoDB supports both embedded data models and references between documents as a relational model. However, one of the benefits of using a document-based NoSQL model is the option to store related data in embedded documents, avoiding the use of joins and speeding up the query. The data model used in the study is embedded documents, which means that all studies, series, and instances are embedded in a patient document, stored in arrays. The data model follows the same organization as the standard and the relational model where a patient has one or more studies with series and images. Unlike the relational model, the document holding the information does not have to be created beforehand, MongoDB handles the creation of documents automatically when the script for inserting the data is run. The database is created with the same attributes as the relational model, making it possible to search for patients and DICOM files based on certain criteria. Listing 4.1 shows the structure of the document in MongoDB.

Listing 4.1: Shows the structure of the DICOM data in MongoDB


```
{
    "_id": {
        "$oid": "68172536a46788dc4ecea810"
    },
    "patient_id": "MSB-01594",
    "patient_name": "MSB-01594",
    "birth_date": "",
    "sex": "O",
    "age": "000Y",
    "studies": [
    {
        "study_instance_uid": "1.3.6.1.4....40891658750872",
        "study_date": "19590712",
        "study_time": "071517",
        "study_description": "CT_Chest",
        "series": [
        {
            "series_instance_uid": "1.3.6.1.4.1...288018627",
            "series_number": 601,
            "series_date": null,
            "series_time": null,
            "modality": "CT",
            "manufacturer": "GE MEDICAL SYSTEMS",
            "series_description": "COR 2X2",
            "bodypart_examined": null,
            "instances": [
             {
                "sop_instance_uid": "1.3.6.1.4...8997772592",
                "instance_number": 14,
                "pixel_spacing": "[0.716797, 0.716797]",
                "num_rows": 512,
                "num_columns": 512,
                "patient_position": "FFS",
                "path_to_instance": "/Users/hannaholmgren/
                  Desktop/manifest-1724932897048/CMB-GEC/
                  MSB-01594/07-12-1959-NA-CTChest-50872/
                  601.000000-COR 2X2-18627/1-013.dcm"
             },
             {
                "sop_instance_uid":
                     "1.3.6.1.4.1.14519.5....6242609464524530",
                "instance_number": 8,
                "pixel_spacing": "[0.716797, 0.716797]",
                ...
             }]
        }]
    }]
}
```

4.5 Queries

Six different queries were performed on each database. The different queries were designed to catch different operations on the database, both simple ones and more complex ones using aggregation, such as group and sort. The queries are meant to reflect a real-world scenario where the user might want to search or filter the result, or only retrieve certain information. The operations performed on the databases are as follows:

1. Retrieve all metadata for CT scans performed on a specific patient by ID.
2. Retrieve the path to the last image for a specific series for one patient.
3. Retrieve the path to the file for all images in a series for a specific patient.
4. Retrieve the path to all CT images for a specific patient.
5. List all patients who have undergone an MRI.
6. List all patients who underwent a CT scan in 1959.
7. Retrieve all pixel data paths in the database.

The full queries performed on each database can be found in the Appendix.

4.6 Experiment

Two Python scripts are used to read the content of the DICOM files and enter the values of specific DICOM tags into the databases. Once the databases have the data, the queries are performed ten times each for both databases. The execution time is measured through a Python script that queries the database and measures the time through the `time.time()` module. Once all queries have been performed, the average execution time is calculated.

5 Result

The average execution time in milliseconds for each query can be seen in Figure 6. The results show that neither database is superior to the other. For queries 2, 3, and 4, MySQL performs better and has a lower retrieval time than MongoDB. For queries 1, 5, 6, and 7, MongoDB has better performance. MySQL retrieved the results in half the time of MongoDB for Query 2, while MySQL also took more than double the time to retrieve the result for Query 5. Both databases show a variety in the retrieval time, depending on the query performed. The highest execution times for both databases can be seen for Query 7 where all 22 321 paths to files in the databases are retrieved.

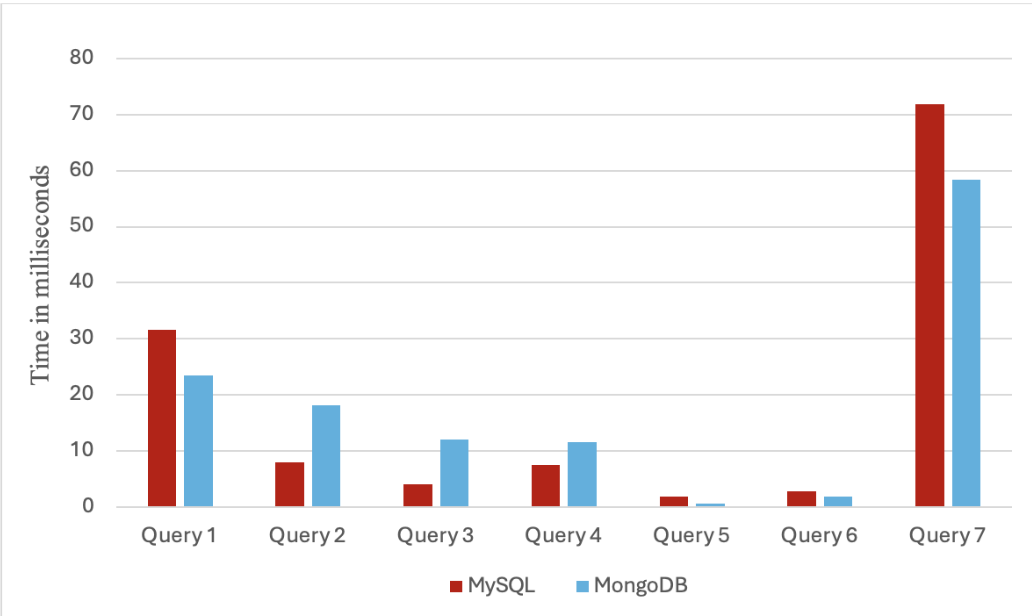

Figure 6: Shows the execution time for the different queries ran on MySQL and MongoDB.

6 Discussion

The result of the study, illustrated in Figure 6, shows that the average execution time for all queries (see Section 4.5) varies for both databases. However, neither database is superior to the other. Although MySQL uses at least one `JOIN` operation for most queries (see Appendix, Listings A.1–A.5), which is typically considered an expensive operation [12], it still outperforms MongoDB in three out of seven queries (see Figure 6).

The difference in retrieval time can be explained by the underlying architecture of the two databases (see Section 2.2). MySQL, the relational model, stores DICOM data in normalized tables (as illustrated in Figure 5), requiring `JOIN` operation to retrieve related information. In contrast, MongoDB, the document-based model, stores data in hierarchical documents, where related data is embedded within the patient document (see Listing 4.1 in Section 4.4).

These structural differences can have an impact on query performance. Different architectures and data models may excel in different types of queries. The result in Figure 6 indicates that MongoDB often performs better in hierarchical queries, as in the case of Query 1, where all the metadata for CT scans performed on a specific patient is retrieved. MySQL has to join all four tables and MongoDB has to unwind all nested documents to retrieve all the metadata (see Appendix, Listings A.1 and A.8). MongoDB also has better performance for simple queries without the need for an aggregation pipeline (see Section 2.2.3) as in the case of queries 5 and 6 (see Appendix, Listings A.12 and A.13) where Query 5 lists all patients who have undergone an MRI and Query 6 lists all patients who underwent a CT scan in 1959. For the same queries, MySQL uses two JOIN operations (see Appendix, Listings A.5 and A.6).

However, Figure 6 shows that MySQL outperforms MongoDB for queries 2, 3, and 4, where not all tables have to be joined, but MongoDB has to unwind all levels of embedded documents. MySQL does not have to join any tables for Query 2 (see Appendix, Listing A.2) which retrieves the path to the last file in a series. Instead, the query applies directly to the `Instance` table and returns the result.

Query 3 and 4 retrieve the path to the files for all images in a specific series and the path to all CT images for a patient. For these queries, MySQL uses two `JOIN` operations (see Appendix, Listings A.3 and A.4) and still performs faster than MongoDB. Although both of these queries use two joins, there is a difference in execution time between the two of them, as seen in Figure 6. This is likely due to the use of condition clauses. Query 3 in MySQL only uses a `WHERE` clause while Query 4 uses both a `WHERE` and an `AND` clause. Based on the result, there also seems to be a threshold at which performing more than two joins, as in the case for Query 1 in MySQL (see Appendix, Listing A.1), becomes slower than unwinding multiple levels of embedded documents.

The query with the highest execution time and number of returned results is Query 7, where all 22 321 paths to files are returned. The high execution time is likely due to the volume of data. Although MySQL does not have to join any tables for the query, MongoDB has to unwind all levels of embedded documents (see Appendix, Listings A.7 and A.14). The result indicates that the unwinding of embedded documents is not a detrimental factor here. Instead, the difference in execution time might come down to each database read performance where MongoDB excels.

Worth noticing though is the order in which MongoDB performs unwind and match operations in the queries (see Section 2.2.3). Although there is no clear indication that the order in which MongoDB performs these operations affects the retrieval time, it is still possible that this could have an impact, since this dictates the number of documents in each stage. Query 3 and 4 in MongoDB have similar retrieval time (as seen in Figure 6) even though they have different query structures (see Appendix, Listings A.10 and A.11). Query 3 unwinds studies and series before performing a match and finishes of by unwinding the instances, resulting in 744 documents returned. However, Query 4 starts with performing a match operation before unwinding the studies and series. Then another match is performed before unwinding the instances resulting in 1034 documents. The first two unwind operations for Query 3 result in 22 and 124 unwound documents, while the same operations for Query 4 after the match operation result in 4 and 29 unwound documents. Although the resulting number of documents returned from the query differs, the order in which unwind and match operations are performed might have an impact on the execution time.

6.1 Limitations

It is important to consider the limitations of the study that could affect the result. One of these limitations is the chosen data, which only consists of five patients and 22 321 images. This can be considered a relatively small dataset. The PACS used to store DICOM files can have hundreds of thousands of patients, but when developing software for image analysis, the need to store such large amounts of data might not be possible due to the limitations of the hardware on the computer. Such software used for image analysis on a local computer might use smaller datasets. Another limitation which might affect the performance of the MongoDB database is the use of the Community Edition, which only uses a single node for processing. One of the benefits of using MongoDB is the possibility of using multiple nodes to distribute the workload, which might increase performance. However, [1] showed that using a distributed node architecture did not improve the retrieval time for queries tested in the study. A third limitation of the study is the omission of indexes in the databases. The use of indexes can make data retrieval faster and applications using a database should in most cases use indexes. Although indexes are recommended, this thesis aims at investigating the underlying structures of different database models and how their architecture affects retrieval time when storing DICOM data.

It is also important to consider the structure of the queries performed (see Appendix). Some queries could have been written differently and possibly optimized the performance. However, for the sake of comparing their performance, the filtering should be made with the same conditions. The queries in MongoDB were designed to return a single document for each result (see Section 2.2.3), the same way a relational model returns one row for each result. For the case of Query 7, where all the paths to the files are returned, the paths are split into their own document. It would be possible to return all the paths still embedded in one document. However, this would result in the need to later traverse the arrays in order to access the actual file path.

6.2 Comparison With Previous Work

Previous studies comparing SQL and NoSQL solutions show varying results. In the case of [8], MongoDB outperformed the SQL solution in terms of reading, which is also confirmed in this thesis, where MongoDB could return all files in the database faster than MySQL. In [10], MongoDB also had better performance for inserting, updating, and simple queries than the NoSQL solution. The findings of [10] are also shown in this thesis, where MongoDB performed better for simple queries, 5 and 6, while MySQL performed better for aggregated queries. It is important to note that the data models used are not shown in the related work, making it difficult to draw any conclusions between the results. However, [3] showed that all SQL queries took longer to execute than MongoDB which was possibly explained by the need for join operations. Although the tables in MySQL had to be joined, MySQL performed better than MongoDB for some queries. Other studies investigating whether or not a NoSQL solution might be a good alternative to the relational approach to store DICOM data do not compare the two models except for in [13], which concluded that a row-based NoSQL model did not perform better than the SQL solution. The other studies focuses on comparing different NoSQL solutions in order to find the best suited one out of the models, which leaves a gap in the understanding of if a NoSQL model is even a good alternative to SQL when it comes to storing DICOM data. This thesis shows that the NoSQL solution does not outperform SQL in all cases in terms of retrieval time.

7 Conclusion

The result (see Figure 6) of the thesis indicates that the decision between a SQL and NoSQL database in the case of storing DICOM data largely depends on the type of queries (see Section 4.5) that will be performed on the database. For simple queries, without the need for an aggregation pipeline (see Appendix, Listings A.12 and A.13), MongoDB performed slightly better than MySQL. For queries that needed aggregation, MySQL performed better in most cases. However, there does appear to be a breakpoint in which multiple `JOIN` operations becomes a more expensive operations than unwinding multiple levels.

One of the appeals of using MongoDB to store the DICOM data is the possibility to model the database according to the DICOM hierarchy (see Figure 1) with studies, series, and instances embedded in one document (see Section 4.4). The same hierarchy cannot be replicated in the same way in MySQL (see Figure 5). Through normalization, a table for each level, that is, patient, study, series, and instance, can be created, but they have to be joined in order for the complete information to be retrieved. Although embedded documents can come with benefits as related information is stored close to each other, this thesis shows that embedding related documents also can slow down retrieval time when unwinding is needed. While MySQL can access an instance directly from the `Instance` table (see Appendix, Listings A.2–A.4), MongoDB has to unwind the studies, series, and instances array in order to retrieve the same information (see Appendix, Listings A.9–A.11). The possibility of omitting certain tables in MySQL makes it faster in some cases as seen in Figure 6 for queries 2, 3, and 4.

Although the decision between the two databases for storing DICOM data is largely dependent on the needs of the database and the operations that will be performed, MongoDB provides a more flexible schema (see Section 2.2.2) that would be beneficial if more attributes are needed in the database but not present in all files. Since the documents in a collection do not have to follow the same schema, MongoDB would prove beneficial in those cases. Adapting the tables in MySQL to contain attributes not present in all files would lead to empty attributes. Other solutions where additional tags are stored in a separate table could possibly be used but would lead to more joins having to be made.

This thesis focuses on investigating the underlying structure of the two database models. In the study, MongoDB uses a document-based model with nested documents, which is the preferred and recommended way to store related data in the model. However, it is also possible to store the data as separate documents, mimicking the relational model with connections between the documents. Future studies could investigate a relational approach in MongoDB or use the embedded document model used in this thesis with distributed nodes to investigate retrieval time.

When choosing which database model is best suited for an application, it is important to consider all relevant factors which might affect the application. The database has to be chosen with the user in mind. Although MongoDB had a slower execution time for some queries (as illustrated in Figure 6) in which the embedded document had to be unwound, the database shows potential as the data volumes increase. The flexible schema of MongoDB and the possibility of adding specific tags to specific documents make the argument for the document-based solution stronger.

References

[1] A. Almeida, F. Oliveira, R. Lebre, and C. Costa. NoSQL distributed database for DICOM objects. In *2020 IEEE International Conference on Bioinformatics and Biomedicine (BIBM)*, pages 1882–1885, 2020.

[2] D. Bierer. *MongoDB 4 Quick Start Guide : Learn the Skills You Need to Work with the World's Most Popular NoSQL Database.* Packt Publishing, 2018.

[3] S. Chakraborty, S. Paul, and H. Azharul Hasan. Performance comparison for data retrieval from NoSQL and SQL databases: A case study for COVID-19 genome sequence dataset. In *2021 2nd International Conference on Robotics, Electrical and Signal Processing Techniques (ICREST)*, pages 324–328, 2021.

[4] H. Garcia-Molina, J. D. Ullman, and J. Widom. *Database Systems: The Complete Book.* Prentice Hall, 01 2002.

[5] M. Kaufmann and A. Meier. *SQL and NoSQL Databases.* Springer, 2019.

[6] K.B.S Kumar, Srividya, and S. Mohanavalli. A performance comparison of document oriented NoSQL databases. In *2017 International Conference on Computer, Communication and Signal Processing (ICCCSP)*, pages 1–6, 2017.

[7] X. Li, P. Morgan, J. Ashburner, J. Smith, and C. Rorden. The first step for neuroimaging data analysis: DICOM to NIfTI conversion. *Journal of Neuroscience Methods*, 264:47–56, 2016.

[8] Y. Li and S. Manoharan. A performance comparison of SQL and NoSQL databases. In *2013 IEEE Pacific Rim Conference on Communications, Computers and Signal Processing (PACRIM)*, pages 15–19, 2013.

[9] A. Nayak, A. Poriya, and D. Poojary. Type of NoSQL databases and its comparison with relational databases. *International Journal of Applied Information Systems*, 5:16–19, 01 2013.

[10] Z. Parker, S. Poe, and S. Vrbsky. Comparing NoSQL MongoDB to an SQL DB. In *Proceedings of the 51st Annual ACM Southeast Conference*, ACMSE '13, New York, NY, USA, 2013. Association for Computing Machinery.

[11] O. S. Pianykh. *Digital Imaging and Communications in Medicine (DICOM).* Springer, 2012.

[12] K. Sahatqija, J. Ajdari, X. Zenuni, and B. Raufi F. Ismaili. Comparison between relational and NoSQL databases. In *2018 41st International Convention on Information and Communication Technology, Electronics and Microelectronics (MIPRO)*, pages 0216–0221, 2018.

[13] A. Savaris, T. Härder, and A. Wangenheim. Evaluating a row-store data model for full-content DICOM management. In *2014 IEEE 27th International Symposium on Computer-Based Medical Systems*, pages 193–198, 2014.

[14] L.A.B. Silva, L. Beroud, C. Costa, and J. Oliveira. Medical imaging archiving: A comparison between several NoSQL solutions. In *IEEE-EMBS International Conference on Biomedical and Health Informatics (BHI)*, pages 65–68, 2014.

A Appendix

Listing A.1: Query 1 in SQL

```
SELECT * FROM Patient
JOIN Study ON Study.patient_id = Patient.patient_id
JOIN Series ON Series.study_instance_uid =
Study.study_instance_uid
JOIN Instance ON Instance.series_instance_uid =
Series.series_instance_uid
WHERE Patient.patient_id = 'MSB-01493'
AND Series.modality = 'CT';
```

Listing A.2: Query 2 in SQL

```
SELECT pixel_data_path FROM Instance
WHERE series_instance_uid = '1.3.6.1.4.1.14519.5.2.1.2
87400551341593291066680000743170423025'
ORDER BY instance_number DESC
LIMIT 1;
```

Listing A.3: Query 3 in SQL

```
SELECT pixel_data_path FROM Instance
JOIN Series ON Instance.series_instance_uid =
Series.series_instance_uid
JOIN Study ON Series.study_instance_uid =
Study.study_instance_uid
WHERE Series.series_instance_uid = '1.3.6.1.4.1.14519.5.2.1.2
87400551341593291066680000743170423025';
```

Listing A.4: Query 4 in SQL

```
SELECT pixel_data_path FROM Instance
JOIN Series ON Instance.series_instance_uid =
Series.series_instance_uid
JOIN Study ON Series.study_instance_uid =
Study.study_instance_uid
WHERE Series.modality = 'MR'
AND Study.patient_id = 'MSB-04437';
```

Listing A.5: Query 5 in SQL

```
SELECT DISTINCT patient_name FROM Patient
JOIN Study ON Patient.patient_id = Study.patient_id
JOIN Series ON Study.study_instance_uid =
Series.study_instance_uid
WHERE Series.modality = 'CT';
```

Listing A.6: Query 6 in SQL

```
SELECT DISTINCT patient_name FROM Patient
JOIN Study ON Patient.patient_id = Study.patient_id
JOIN Series ON Study.study_instance_uid =
Series.study_instance_uid
WHERE Series.modality = 'MR'
AND Study.study_date BETWEEN '1959-01-01' AND '1959-12-31';
```

Listing A.7: Query 7 in SQL

```
SELECT pixel_data_path FROM Instance;
```

Listing A.8: Query 1 in MongoDB

```
db.getCollection('patient').aggregate(
  [
    { $match: { patient_id: 'MSB-01493' } },
    { $unwind: { path: '$studies' } },
    { $unwind: { path: '$studies.series' } },
    {
      $match: { 'studies.series.modality': 'CT' }
    },
    {
      $unwind: {
        path_to_instance: '$studies.series.instances'
      }
    }
  ],
  { maxTimeMS: 60000, allowDiskUse: true }
);
```

Listing A.9: Query 2 in MongoDB

```
      db.getCollection('patient').aggregate(
  [
    { $unwind: { path: '$studies' } },
    { $unwind: { path: '$studies.series' } },
    {
      $match: {
        'studies.series.series_instance_uid':
          '1.3.6.1.4.1.14519.5.2.1.2874005513415932910
          666800007431704230255'
      }
    },
    {
      $unwind: {
        path_to_instance: '$studies.series.instances'
      }
    },
    {
      $sort: {
        'studies.series.instances.instance_number':
```

```
          -1
        }
      },
      { $limit: 1 },
      {
        $project: {
          path_to_instance:
            '$studies.series.instances.path_to_instance'
        }
      }
    ],
    { maxTimeMS: 60000, allowDiskUse: true }
);
```

Listing A.10: Query 3 in MongoDB

```
db.getCollection('patient').aggregate(
    [
      { $unwind: { path: '$studies' } },
      { $unwind: { path: '$studies.series' } },
      {
        $match: {
          'studies.series.series_instance_uid':
            '1.3.6.1.4.1.14519.5.2.1.28740055134
            159329106668000074317042302 5'
        }
      },
      {
        $unwind: {
          path: '$studies.series.instances'
        }
      },
      {
        $project: {
          path_to_instance:
            '$studies.series.instances.path_to_instance'
        }
      }
    ],
    { maxTimeMS: 60000, allowDiskUse: true }
);
```

Listing A.11: Query 4 in MongoDB

```
db.getCollection('patient').aggregate(
    [
      { $match: { patient_id: 'MSB-04437' } },
      { $unwind: { path: '$studies' } },
      { $unwind: { path: '$studies.series' } },
      {
        $match: { 'studies.series.modality': 'MR' }
      },
```

```
    {
      $unwind: {
        path_to_instance: '$studies.series.instances'
      }
    },
    {
      $project: {
        path_to_instance:
          '$studies.series.instances.path_to_instance'
      }
    }
  ],
  { maxTimeMS: 60000, allowDiskUse: true }
);
```

Listing A.12: Query 5 in MongoDB

```
db.getCollection('patient').find({"studies.series.modality": "CT"},
{patient_name:1});
```

Listing A.13: Query 6 in MongoDB

```
db.getCollection('patient').find({'studies.series.modality': 'MR',
'studies.study_date': { $regex: '^1959' },{"patient_name":1});
```

Listing A.14: Query 7 in MongoDB

```
db.getCollection('patient').aggregate(
  [
    { $unwind: { path: '$studies' } },
    { $unwind: { path: '$studies.series' } },
    {
      $unwind: {
        path: '$studies.series.instances'
      }
    },
    {
      $project: {
        path_to_instance:
          '$studies.series.instances.path_to_instance'
      }
    }
  ],
  { maxTimeMS: 60000, allowDiskUse: true }
);
```

UMEÅ UNIVERSITY