IOThings MongoDB Report and Implementation

Jaspreet singh

Bachelor of Science with Honours Computer and Data Science

School of Computing and Digital Technology

Faculty of Computing, Engineering and the Built Environment

Birmingham City University

Submitted May 2022

BIRMINGHAM CITY
University

Table of Contents

1 Part A

1.1 Introduction

Due to the various restrictions of a relational database and the new requirements for faster and more diverse utilization of massive data sets became increasingly vital as time progressed. SQL databases were not flexible enough for programmers that end up finding the perfect alternative in the NoSQL extension of SQL.

1.2 NoSQL vs Relational databases

NoSQL, often known as not only SQL, is a database design technique that allows data to be stored and queried outside of the standard structures found in relational databases. NoSQL databases store data in the perfect data format that is adaptable, making it ideal for storing unstructured data, rather than the conventional tabular structure of a relational database that is only appropriate for managing relational data. Because this non-relational database architecture does not require a schema, it can handle massive, often unstructured data sets quickly. (Kunda, D. and Phiri, H., 2017)

NoSQL provides a variety of data storage and retrieval options such as, key-value data, wide-column databases, graph structures, and JSON documents are all options for storing data in NoSQL.(Sharma, V. and Dave, M., 2012) This allows for the storage of data in various structures or unstructured data where the JSON documents option, in particular, reveals to be quite useful as JSON objects have become the standard for transferring data across apps. NoSQL databases have a changeable schema and allow you to work with unstructured data so applications and databases can use the same data formats. This implies you may start developing your app without first defining the schema other than cutting down on the number of transformations needed to move data between apps and databases. However, to store and handle records in a relational database, you must first specify your schema and relationships before you can begin adding data. Structured queries are made easier by defining data into relational schemas. As a result, the performance and flexibility of SQL queries are determined by the design thus changing the schema structure of a relational database is costly, time-consuming, and frequently involves downtime or service disruptions.

Given its nature, NoSQL databases can handle massive volumes of data at breakneck rates and scale-up architecture implementations benefit from the same flexibility. In fact, scaling a NoSQL database is substantially less expensive since capacity can be added horizontally across cheap, commodity servers, especially with the introduction of cloud computing which has become even more pronounced nowadays. When vertically scalable SQL databases are utilized, often results as more costly. Because they only need a single server to host the complete database and scaling would require purchasing a larger and more expensive server. (Kunda, D. and Phiri, H., 2017) Being NoSQL a distributed database, the data is duplicated and stored on a number of servers, both remote and local. This ensures data availability and consistency and even if some of the data is unavailable, the database as a whole can continue to function. In order to accommodate the developer's preferences and security when changing multiple data formats during the development and deployment stages, NoSQL's nature helps with problems associated with repurposing data into new formats or when working with data stored into not accessible servers, giving it flexibility and enhancing the development process.

NoSQL is not perfect, it has certain disadvantages as well. The lack of uniformity is the main disadvantage of NoSQL.(Leavitt, N., 2010) Several NoSQL database systems exist, however, they lack the same defined protocols that regulate relational database systems' usage, deployment, and data exchange. Support for NoSQL systems and queries is made more difficult by the fact that most vendors' systems have their own set of properties. The future of NoSQL databases is jeopardized by the lack of standardization, only incompatible and specialized platform systems are present and they might quickly become obsolete as even the lack of compatibility with SQL instructions makes it hard to transfer queries between them other than the absence of any Cross-platform support between operating systems. The fact that NoSQL stores redundant entries of the same data, it jeopardizes the data's integrity and atomicity. When using NoSQL to store data, no checks are performed to assure data integrity. Unlike SQL databases, which use a primary key constraint to assure data atomicity, this is because NoSQL databases prioritize scalability and performance over data consistency and integrity.

1.3 Types of NoSQL Databases

Document and graph databases as well as column-oriented and key-value stores databases are some common examples of NoSQL (Lemahieu et al., 2018). These databases store unstructured data by using dynamic schemas. Furthermore, to store and manipulate data, they do not require a structured query language and by being without a preset structure, these NoSQL systems are adaptable and horizontally scalable.

Key-value stores are a common database type put into practice in NoSQL systems. This is because it allows scalability and flexibility by using a name-value combinations that can be utilized to store or retrieve data elements as it provides identification for a record whose value is stored against a name. For this to be retrieved, a key or value is supplied to access the records, making this method minimally complex other than flexible as well as scalable, in fact, there are several key-value sets that users can adopt to store any type of data.

Storing data in document formats such as Extensible Markup Language, JavaScript Object Notation and Binary JSON is what document databases allow as well as the possibility to store and retrieve data similar to data objects processed by applications. When data is shared between the application and database, there is little modification required, this makes it flexible to adjust the structure of the data in order to make it significant to the application. For the horizontal scaling architecture and the need for extensive data processing, it is crucial that the format for both, the application and the database, is alike. The most recurrent NoSQL database system put into effect for various applications is the JSON document database, while an illustrious system that implements it is MongoDB as it requires to processes a large amount of data and requires flexibility.

Graph database allows data to be stored in the form of edges and nodes. Data stored as nodes is information about real objects whereas data sorted as edges are data that links the nodes together. The relationships between data is the focal point for this type of database, and the primary elements are the relationships, which is not the case for relational databases. When it comes to different data elements' relationships, high optimization is required for graph databases to ensure effectiveness and exceed performance in queries (Winkler et al., 2019). However despite this, the usage of graph databases in a stand-lone business-orientated database is rare and they are more often used along with relational databases for real-time recommendation engines, master data management, fraud detection systems, network, and IT operations, Identity and access management.

The main similarity between relational and column-oriented databases is that data is stored in a tabular form. However, an additional factor of the column-oriented database is that it has inflexible columns, giving it more flexibility as data can be organized into columns, unlike relational databases. (Khosrow-Pour, 2017) Relational databases are optimized for storing rows of data, typically for transaction-based applications, while columnar databases are optimized for quickly retrieving columns of data, typically for analytics applications and business intelligence. Column storage for database tables is a very important factor when it comes to analytical query performance, as it can dramatically reduce the overall I / O requirements, as well as decrease the amount of data to load, thus improving performance (Lemahieu et al., 2018). A disadvantage is that it can be extremely time-consuming as the database requires information in each column, (Mejia et al., 2019) while the scan queries can be complicated as the columns are required to be stitched back together for a row to be formed within the database (Selamat et al., 2013).

1.4 Different NoSQL Databases

NoSQL databases are implemented in a variety of ways. Some are cross-platform, proprietary, and open-source. MongoDB, Cassandra, Neo4J, Amazon DynamoDB, and HBase are some of these possibilities. Many businesses provide solutions for specific reasons and as a result, the architectural designs, schema designs, and query models of these systems differ from one another.

MongoDB

It's an open-source and cross-platform document database that is based on collections and documents and it saves and retrieves data in JSON format while the document storage format utilised is BSON. To guarantee elasticity, flexibility, and performance, data has replica sets that are also indexed using main and secondary indices. Shard keys provide horizontal scalability and they control how data is dispersed across various storage systems while the grid file system is used for load balancing and data replication. MongoDB provides several commands for storing and retrieving data objects, while executing these commands, the object skeleton described by the schema must be followed by the objects inserted or queried. (Gupta, A., Tyagi, S., Panwar, N., Sachdeva, S. and Saxena, U., 2017)

Cassandra

It is a free and open-source NoSQL database system implemented with a column-oriented database and a language syntax, Cassandra Language Query, comparable to regular SQL that enables the creation of a keyspace, a namespace for identifying data replicated across several nodes.(Abramova, V. and Bernardino, J., 2013) As a result, keyspaces may be thought of as relational databases with tables. It's built on a distributed architecture with nodes that are masterless that grant dependable availability with no single point of failure. As the data is spread evenly and copied across all nodes, it can withstand the loss of an entire data center and ensure no data is lost during a failure. Low latency is also ensured by the data being automatically duplicated across all defined data centers.

Neo4j

It’s a highly scalable and efficient graph database with dispersed clusters that collaborate closely and expand as data grows, lowering operational expenses. It's made to handle data in a linked state, data items are kept as nodes and edges with highly optimized relationships for speed and scalability. (Sharma, Monika & Deep, Vishal & Bundele, Mahesh., 2020) Neo4j provides a fine-grained degree of security thanks first to the Schema-based security that protects nodes and their interactions, and second to the employment of role-based access control that guarantees that a user's access to database attributes and objects is limited to its allocated role. Finally, its query language is called Cipher and it uses an ASCII-art type of syntax. Edges have only one direction and are logically represented as lines joining nodes that express relationships, while nodes can be source or destination and are logically represented as circles.

Amazon DynamoDB

This table-based database is made by a key-value store plus a document database. Its tables are made up of attributes, which are the values of the recorded data, that is obtained and saved using JSON syntax. To make data retrieval and storage easier, partition keys and sort keys are employed as indices. The partition key is used to hash the several partitions spread over the storage and a write action is received straight from the user and the request is authenticated before the execution. Finally, to provide high availability and performance, data replication across several data centers is synchronized.

HBase

It’s a column-oriented open-source NoSQL database system, which architecture provides for speedy data retrieval, making it ideal for dealing with large amounts of sparse data. As a result, data is processed as columns, indexed with unique column keys. Its data is organized into tables with rows and columns, where the data in the rows are divided into column families by IDs. The HBase database system is made up of distributed clusters that provide data replication and that handle load balancing and automated scalability, this type also provides automatic failure support, consistent read and writes, and an easy java API for clients and it integrates with Hadoop, both as a source and at the destination.

1.5 Conclusion

With rising demands on database systems, a new paradigm named NoSQL for storing and manipulating data differently from the standard SQL language was adopted. This model is able to process enormous amounts of unstructured data with flexibility, scalability, and developer friendliness otherwise impossible with the traditional relational database. Different database types have been created to satisfy the aims of NoSQL databases, such as Column-oriented databases, graph databases, key-value databases, and document databases. These types define their data structure and have been created to accommodate various sorts of data, in fact, the design architecture, schema design, and query model of the systems are all different. However, NoSQL has a disadvantage, that is the lack of standardization that makes it difficult to design cross-platform solutions that are interoperable.

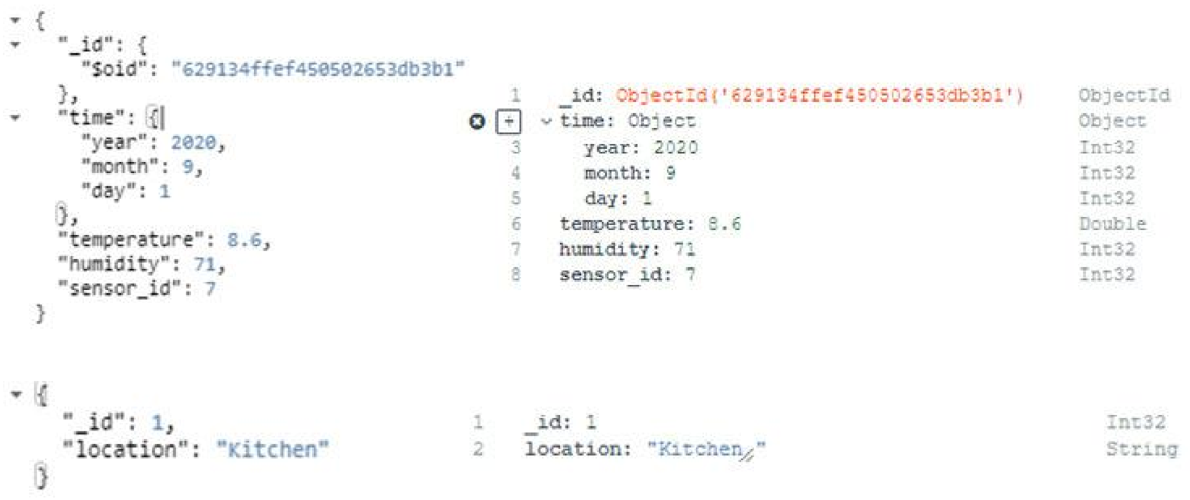
2 Part B

2.1 IoT, problem statement

In regards of big data technology combined with IoT (Internet of Things), a variety of instances have been born, as such, the smart temperature and humidity monitoring systems provide significant social, financial, and environmental advantages to homes and businesses. Some of these include the warranty on temperature or humidity sensible food and goods as well as people’s ideal values for health, security, and convenience. This project's purpose is to utilize a web application to measure how the values of humidity and temperature in the household change over time. People and businesses will be able to better comprehend and operate changes in the humidity and temperature for their own goals and objectives through their actions or other IoT instruments.

2.2 Data Model, Design Process and Architecture

There will be two collections, sensors and readings, the first with only the id value (as int) and the location (as string) of the sensor while the second will consists of the temperature (double) and humidity (int) values in a certain day (int) of a certain month (int) of the year 2020. Furthermore, the readings collection will also include a reference from the collection of the sensor in a one-to-many relationship regarding the sensor id. (Figure 1)



```
{
  "_id": {
    "$oid": "629134ffef450502653db3b1"
  },
  "time": {
    "year": 2020,
    "month": 9,
    "day": 1
  },
  "temperature": 8.6,
  "humidity": 71,
  "sensor_id": 7
}
```

```
_id: ObjectId('629134ffef450502653db3b1')    ObjectId
time: Object                                 Object
  year: 2020                                 Int32
  month: 9                                   Int32
  day: 1                                     Int32
temperature: 8.6                             Double
humidity: 71                                 Int32
sensor_id: 7                                 Int32
```

```
{
  "_id": 1,
  "location": "Kitchen"
}
```

```
_id: 1                  Int32
location: "Kitchen"     String
```

Figure 1 (Fields, types, and relationships present in the collections.)

Python was used to construct the web application since it is one of the most used programming languages nowadays, its also known for its simple code, low cost, and adaptability to every situation. It would not be wise to run phyton codes inside an HTML page but it’s still possible to use it with the use of a python framework. Some of these include Django, Tornado, Bottle, Flask, TurboGears, and jam.py. In this project, the Flask framework will be chosen as it comes with a variety of useful libraries such as render_template. The creation of a web application it’s usually distinguishable between the front end and back end, all the user’s requests are compiled in the back end thanks to python, flask, and the MongoDB database that execute the lines of code after a POST or GET method coming from the front end composed of HTML and CSS files in order to send back a response to it so it can be seen by the user.
In this artifact the data regarding the temperature and humidity are stored in the MongoDB database in the localhost:27017, then the data can be edited through phyton and flask that show a simple front end visible at localhost:5000 with an admin edit page where a possible administrator can execute changes to the database. While other possible information fetched from the database can be seen as tables on the show data page.

2.3 Tools selections

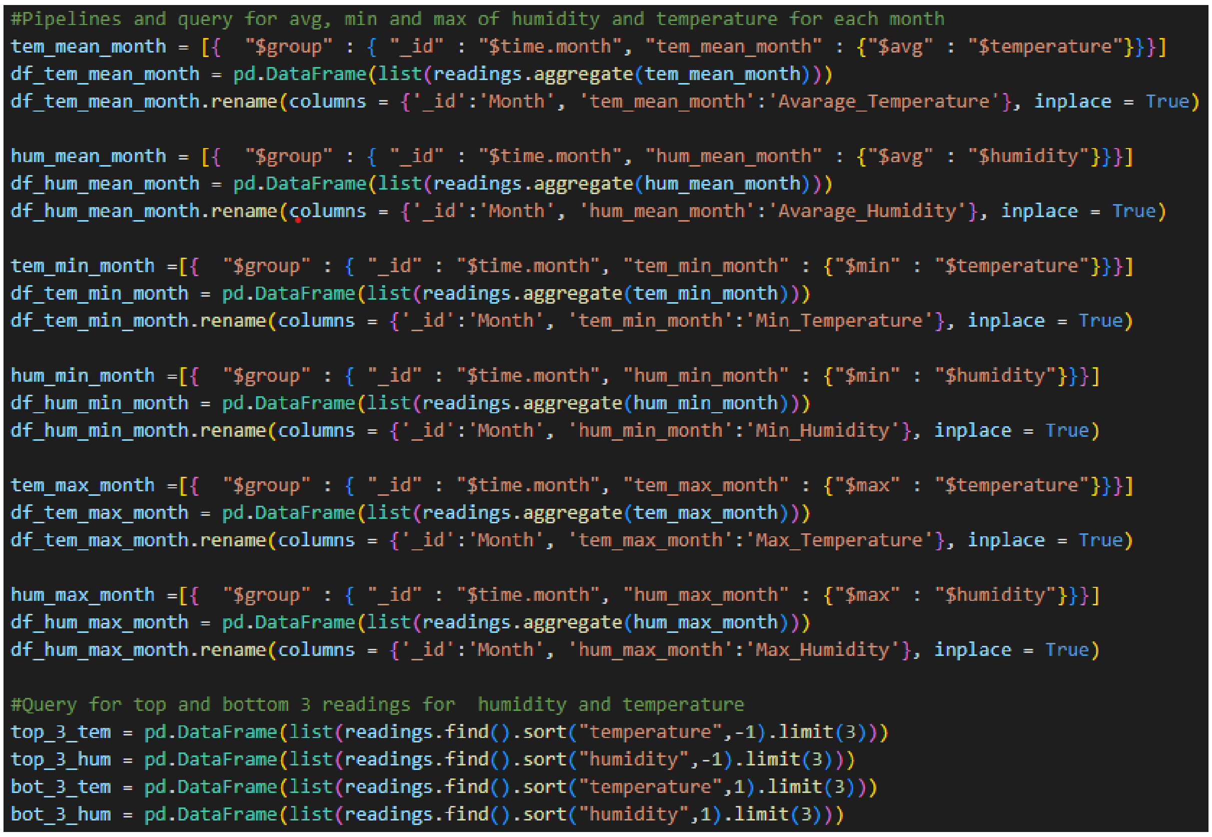
The various tool, frameworks, and Python libraries used for this project are the Python programming language for the code, flask as a web framework, MongoDB, a Document based database, Html, and CSS for the web language and styling. The libraries used are Pymongo, a library for interacting with the MongoDB database, NumPy used for working with arrays and finally Pandas, used for working with data frames and sets.

2.4 Data Characteristics

At the start, the database results are empty but it can be quickly filled up with data for both collections thanks to a randomly generated instance creation present on the admin edit page. The creation inserts the following possible id as well as a random location in the area of the house for the sensor while it inserts a random date and values for the temperature and humidity, in regard to the one-to-many relationship the program gives the id of one of the existing sensors. In the case of a user with admin access, it's possible to enter the edit page to add a new sensor or 5 new readings to populate the database and possibly check the last insert instance or all of them. Once data has been added to a certain extent it's possible to see the highest, lowest, average, min and max of each month’s temperature and humidity values in tables and have a better insight on the data in the system.

2.5 Queries

The queries written down for the creation of the project are present below in the figure 2.

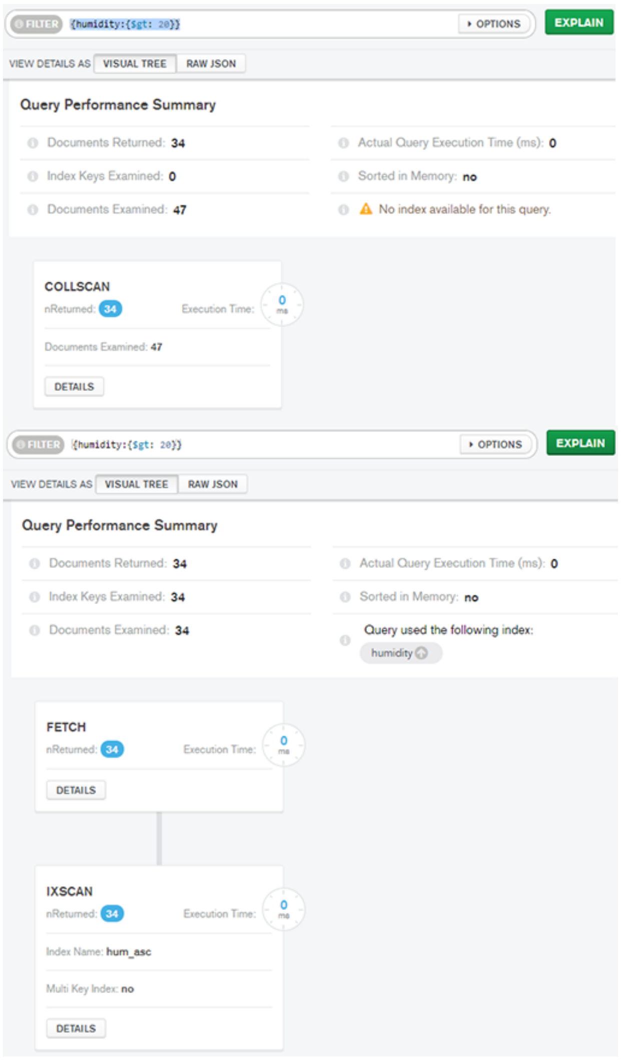


```
#Pipelines and query for avg, min and max of humidity and temperature for each month
tem_mean_month = [{  "$group" : { "_id" : "$time.month", "tem_mean_month" : {"$avg" : "$temperature"}}}]
df_tem_mean_month = pd.DataFrame(list(readings.aggregate(tem_mean_month)))
df_tem_mean_month.rename(columns = {'_id':'Month', 'tem_mean_month':'Avarage_Temperature'}, inplace = True)

hum_mean_month = [{  "$group" : { "_id" : "$time.month", "hum_mean_month" : {"$avg" : "$humidity"}}}]
df_hum_mean_month = pd.DataFrame(list(readings.aggregate(hum_mean_month)))
df_hum_mean_month.rename(columns = {'_id':'Month', 'hum_mean_month':'Avarage_Humidity'}, inplace = True)

tem_min_month =[{  "$group" : { "_id" : "$time.month", "tem_min_month" : {"$min" : "$temperature"}}}]
df_tem_min_month = pd.DataFrame(list(readings.aggregate(tem_min_month)))
df_tem_min_month.rename(columns = {'_id':'Month', 'tem_min_month':'Min_Temperature'}, inplace = True)

hum_min_month =[{  "$group" : { "_id" : "$time.month", "hum_min_month" : {"$min" : "$humidity"}}}]
df_hum_min_month = pd.DataFrame(list(readings.aggregate(hum_min_month)))
df_hum_min_month.rename(columns = {'_id':'Month', 'hum_min_month':'Min_Humidity'}, inplace = True)

tem_max_month =[{  "$group" : { "_id" : "$time.month", "tem_max_month" : {"$max" : "$temperature"}}}]
df_tem_max_month = pd.DataFrame(list(readings.aggregate(tem_max_month)))
df_tem_max_month.rename(columns = {'_id':'Month', 'tem_max_month':'Max_Temperature'}, inplace = True)

hum_max_month =[{  "$group" : { "_id" : "$time.month", "hum_max_month" : {"$max" : "$humidity"}}}]
df_hum_max_month = pd.DataFrame(list(readings.aggregate(hum_max_month)))
df_hum_max_month.rename(columns = {'_id':'Month', 'hum_max_month':'Max_Humidity'}, inplace = True)

#Query for top and bottom 3 readings for  humidity and temperature
top_3_tem = pd.DataFrame(list(readings.find().sort("temperature",-1).limit(3)))
top_3_hum = pd.DataFrame(list(readings.find().sort("humidity",-1).limit(3)))
bot_3_tem = pd.DataFrame(list(readings.find().sort("temperature",1).limit(3)))
bot_3_hum = pd.DataFrame(list(readings.find().sort("humidity",1).limit(3)))
```

Figure 2 (Queries and pipelines for the project.)

2.6 MongoDB Query Optimization

As for the query, optimization is possible to use an indexing mechanism in order to speed up MongoDB queries as it could drastically decrease the operational time for each operation. That means MongoDB will not proceed to scan every element that could match the query in the collection thus requiring a large amount of computational power and time but will instead provide an efficient and fast query resolution. Indexes are special data structures that store some information related to the documents such that it becomes easy for MongoDB to find the right data file. The indexes are ordered by the value of the field specified in the index. (GeeksforGeeks, 2020) In the artifact two indexes have been created, one for temperature and one for humidity, and both rank in an ascending order their field`s values. Figure 3 below its shows their difference in performance even with a relatively small amount of documents.



```
{humidity:{$gt: 20}}
```

Query Performance Summary

Documents Returned: 34
Index Keys Examined: 0
Documents Examined: 47
Actual Query Execution Time (ms): 0
Sorted in Memory: no
No index available for this query.

COLLSCAN
nReturned: 34
Execution Time: 0 ms
Documents Examined: 47
DETAILS

```
{humidity:{$gt: 20}}
```

Query Performance Summary

Documents Returned: 34
Index Keys Examined: 34
Documents Examined: 34
Actual Query Execution Time (ms): 0
Sorted in Memory: no
Query used the following index:
humidity

FETCH
nReturned: 34
Execution Time: 0 ms
DETAILS

IXSCAN
nReturned: 34
Execution Time: 0 ms
Index Name: hum_asc
Multi Key Index: no
DETAILS

Figure 3 (Difference in performance with and without indexes.)

2.7 Sharding and replication

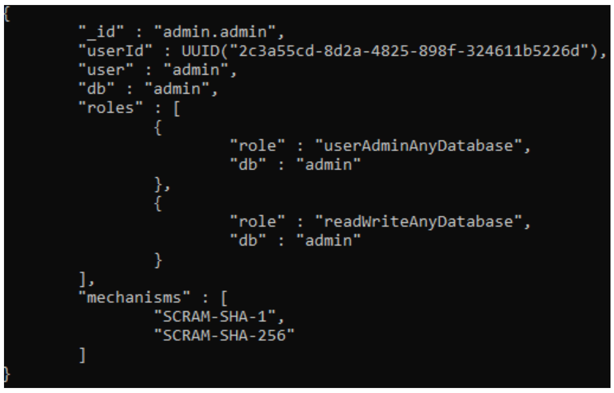
As the database isn't too big the sharding of tables into smaller ones was not necessary, but in order to protect the database from a potential failure, replication can be used to synchronize data across servers to provide redundancy and increases data availability. This method has been used and a replica set of 3 has been constructed.

```
mongod --port 27017 --dbpath C:\Users\jaspr\data\rs1 --replSet replicaSet1
mongod --port 27027 --dbpath C:\Users\jaspr\data\rs2 --replSet replicaSet2
mongod --port 27037 --dbpath C:\Users\jaspr\data\rs3 --replSet replicaSet3
```

primary cluster port – 27017 secondary cluster port – 27027, 27037

2.8 Database administration

MongoDB Administration is in charge of a multitude of operations and its major Task includes Installation, Configuration, Database Design, Monitoring, Troubleshooting, Backup, and recovery. (EDUCBA, 2020) Users and administrators are two different types of roles in a database and both have different types of privileges to read and write inside certain databases that can be specifically modified. In this project, an admin with privileges is created and its details are needed to access the database.



```
{
        "_id" : "admin.admin",
        "userId" : UUID("2c3a55cd-8d2a-4825-898f-324611b5226d"),
        "user" : "admin",
        "db" : "admin",
        "roles" : [
                {
                        "role" : "userAdminAnyDatabase",
                        "db" : "admin"
                },
                {
                        "role" : "readWriteAnyDatabase",
                        "db" : "admin"
                }
        ],
        "mechanisms" : [
                "SCRAM-SHA-1",
                "SCRAM-SHA-256"
        ]
}
```

Figure 4 (Admin user used to access database.)

2.9 Conclusion and future work

While the artifact used basic levels of each listed tool its still able to provide a general understanding of the applied data and methods. Unfortunately as a result of not using real-world data but instead randomly generated one the system does not properly represent information that could be taken fully into account, thus the necessity in the future to use real-world data. Another improvement could be the visualization of the data in plotted graphs with matplotlib or similar libraries that could not make it in time for this build.

3 Bibliography

Kunda, D. and Phiri, H. (2017) “A Comparative Study of NoSQL and Relational Database”, Zambia ICT Journal, 1(1), pp. 1-4. doi: 10.33260/zictjournal.v1i1.8.

Sharma, V. and Dave, M. (2012). SQL and NoSQL Databases. International Journal of Advanced Research in Computer Science and Software Engineering, [online] 2(8), pp.1–8. Available at: https://www.researchgate.net/profile/Meenu-Dave/publication/303856633_SQL_and_NoSQL_Databases/links/5758557f08ae9a9c954a7573/SQL-and-NoSQL-Databases.pdf ISSN: 2277 128X

Leavitt, N. (2010). Will NoSQL Databases Live Up to Their Promise? Computer, 43(2), pp.12–14. doi:10.1109/mc.2010.58.

Abramova, V. and Bernardino, J. (2013). NoSQL Databases: MongoDB vs Cassandra. [online] Available at: https://web.cs.wpi.edu/~cs585/s17/StudentsPresentations/This%20Year/Week14/mongodb_vs_cassandra.pdf.

Gupta, A., Tyagi, S., Panwar, N., Sachdeva, S. and Saxena, U. (2017). NoSQL databases: Critical analysis and comparison. [online] IEEE Xplore. doi:10.1109/IC3TSN.2017.8284494.

Sharma, Monika & Deep, Vishal & Bundele, Mahesh. (2020). Performance Analysis of RDBMS and No SQL Databases: PostgreSQL, MongoDB and Neo4j.

Mehdi Khosrow-Pour (2019). Advanced methodologies and technologies in network architecture, mobile computing, and data analytics. Hershey, Pa: Engineering Science Reference.

Winkler, D., Biffl, S. and Johannes Bergsmann (2019). Software Quality: The Complexity and Challenges of Software Engineering and Software Quality in the Cloud 11th International Conference, SWQD 2019, Vienna, Austria, January 15–18, 2019, Proceedings. Cham Springer International Publishing.

Selamat, A., Thanh, N. and Habibollah Haron (2013). Intelligent information and database systems : 5th Asian Conference, ACIIDS 2013, Kuala Lumpur, Malaysia, March 18-20, 2013, proceedings. Part I / edited by Ali Selamat, Ngoc Thanh Nguyen and Habibollah Haron. Berlin: Springer.

Jezreel Mejía Miranda, MuñozM., RochaA. and Antonio, J. (2020). Trends and applications in software engineering : proceedings of the 8th International Conference on Software Process Improvement (CIMPS 2019). Cham, Switzerland Springer.

Lemahieu, W., Broucke, S. vanden and Baesens, B. (2018). Principles of Database Management: The Practical Guide to Storing, Managing and Analyzing Big and Small Data. [online] Cambridge University Press, pp.1–770. Available at: https://books.google.co.uk/books?id=gAJiDwAAQBAJ&hl=it&source=gbs_navlinks_s.

EDUCBA. (2020). MongoDB Administration | Why Do We Need MongoDB Administration? [online] Available at: https://www.educba.com/mongodb-administration/

GeeksforGeeks. (2020). Indexing in MongoDB. [online] Available at: https://www.geeksforgeeks.org/indexing-in-mongodb/#:~:text=Indexes%20are%20special%20data%20structures.