Comparing SQL and NoSQL Database for IoT Applications

Project Proposal

Submitted To:

Dr. Ming Hwa Wang

Submitted By:

Venkatasai Rao Dheekonda

Sai Sankeerth Gangasani

Sajit Valiya Kizhakke

DATA 225- DB SYSTEMS FOR ANALYTICS

SPRING 2022

TABLE OF CONTENTS

ACKNOWLEDGEMENTS

First and foremost, we would like to express our gratitude to the founders of Planet Watch, authors of our reference papers, who generously shared their research findings and expertise with us, allowing us to gain a thorough understanding of the concepts and fundamentals, which was extremely helpful in completing this project.

We would also like to express our gratitude to Professor Dr. Wang, who taught us concepts and shared principles for us to complete the project.

We would also like to express our gratitude for the support we've received from our family and friends. We would not have been able to finish our job without their assistance.

Finally, we want to express our gratitude to everyone who assisted us in completing this project and encouraged us to do so.

LIST OF TABLES

LIST OF FIGURES

LIST OF ABBREVIATIONS

Abbreviation	Meaning
IoT	Internet of Things
Et al	And others
SQL	Structured Query Language
NoSQL	Not only SQL or Non-SQL
MySQL	My Structured Query Language
DB	Database
WHO	World Health Organization
MongoDB	A document NoSQL store, Mongo is a short name for the word humongous
YCSB	Yahoo Cloud Service Benchmarking
CO	Carbon Monoxide
NO2	Nitrogen Dioxide
CO2	Carbon Dioxide
H2S	Hydrogen Sulfide
SO2	Sulphur Dioxide
PM2.5	Fine inhalable particles with sizes of 2.5 micrometers or less are referred to as PM2.5
NO	Nitrogen Monoxide
ppm	Parts per million
ppb	Parts per billion
µg/m3	micrograms per cubic meter
ms	millisecond

1. ABSTRACT

The ceaseless generation of heterogeneous data from various platforms has produced numerous challenges to manage, store and more importantly transfer the data. With the advent of IoT, these challenges and many more like data security, data complexity, data volume, and challenges concerning the privacy of the data. Conventional databases that used the relational schema did fulfill a lot of user demands such as data integrity, database simplicity, and compatibility to name a few but such databases were found to lacking in several other aspects owing to the static design that these databases are built on. Incorporating this rigid schema renders the RDBMS to be unadaptable for IoT applications. This is where NoSQL databases, which are rapidly gaining prominence in the market, lend themselves an advantage. The dynamic and non-relational schema which a NoSQL database provides is suitable for hierarchical data storage and also the data generated by IoT applications which is largely unstructured. The high scalability, availability, and the high-performance features of NoSQL databases are what this paper will be focusing on. We aim to answer several questions such as whether a NoSQL database outperforms a traditional database in various scenarios? Is a NoSQL database superior to a SQL database in terms of the execution speed for multiple queries or actions? We will compare MySQL with MongoDB for SQL and NoSQL databases for an IoT application emphasizing air quality and examine the performances of the databases in handling the real-time data.

Keywords—NoSQL, SQL, IoT, MongoDB, MySQL, DBMS

2. INTRODUCTION

2.1 OBJECTIVE:

Analyze the performance of various database operations using SQL and NoSQL databases for IoT applications. We intend to focus on a single IoT application to capture the air quality data with multiple IoT devices from different locations. Our project idea is inspired by the Planet Watch, a startup that's aiming to set up devices around the globe by incentivizing people who set up these devices in their locality.

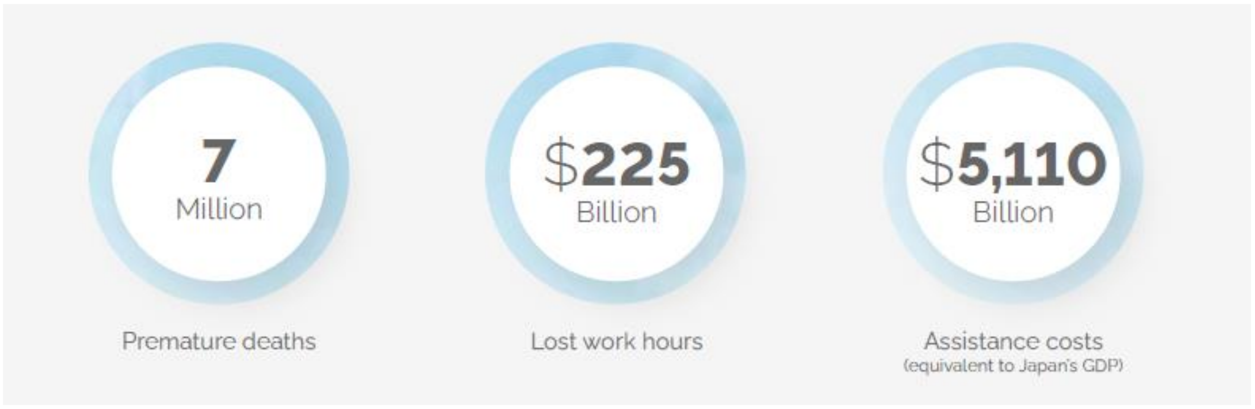
2.2 WHAT IS THE PROBLEM:

Bad air quality is one of the leading health hazards across the globe. World Health Organization's data reveals that nearly 90% of the world's population breathes unhealthy air, i.e. the air which exceeds WHO guidelines. Deaths caused by Air pollution annually are at a staggering rate of about seven million worldwide, with low and middle-income countries suffering the pollution's dire consequences.

WHO has classified air pollution into two major categories:

(i) **Outdoor air pollution:** Nearly 4.5 million people lose their lives per year due to respiratory-related illnesses, a few of them being lung cancer, stroke, and acute and chronic respiratory afflictions.

(ii) **Indoor air pollution:** In underdeveloped and developing nations, indoor air pollution is a root cause of higher and premature mortality. Even at low levels, it is harmful to one's physical and mental well-being and greatly cuts down their overall diminished productivity.



Source: World Health Organization

Figure 1: Resources wastage due to pollution

2.3 WHY THIS PROJECT IS RELATED TO THIS CLASS:

This project is related to this class because it aligns with the course objectives and learning outcomes. The project focuses on the database management systems, principally comparing SQL and NoSQL databases and finding out which of the DBS are compatible with our needs, which of them is more efficient than the other, and which one supports the IoT application or heterogeneous big data on the whole. It also traverses through the cloud computing aspect of the software development and will demonstrate how a device could be controlled and connected and how it could transmit data through the internet. This project also involves ETL and data warehouse which are very much related to the subject of the class.

2.4 WHY OTHER APPROACH IS NO GOOD:

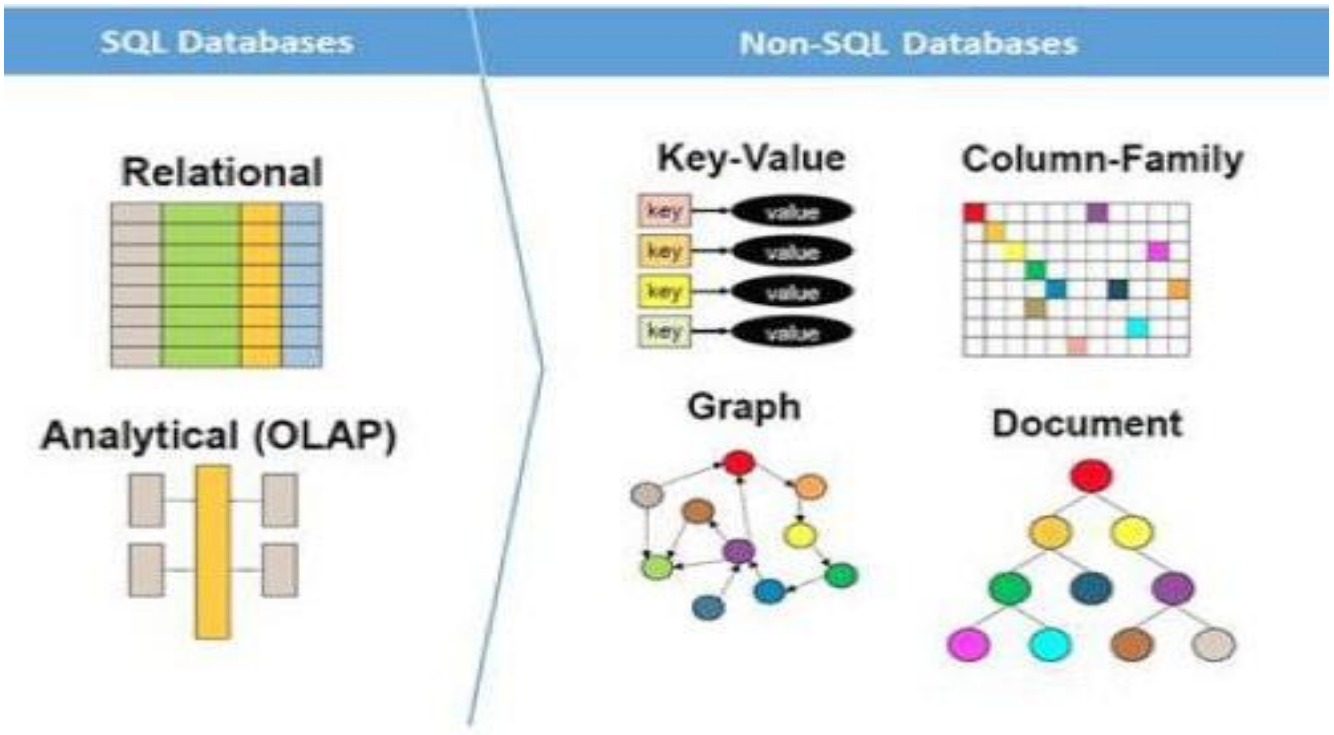
Relying on traditional DBs for big and heterogeneous data could end up being futile and calls for a better alternative to store, analyze and perform operations on the data as per our needs. It is very tricky and easy for someone to get lost while choosing a database for their use case. With the increase in IoT devices, we are talking about petabytes of data, as ACID restrictions of traditional databases impair performance.

2.5 WHY WE THINK OUR APPROACH IS BETTER:

While one prepares to store their data on any cloud platform, they encounter choices to do so. And the first action they will have to take a verdict on would be either to opt for a SQL database or a NoSQL database as their primary database or in many cases would they require the services of both these types of databases to accomplish requirements. More often than not we have to pay heed to the intricate aspects of our data. We have a vivid concept or a notion on how our data should resemble on multiple terrains such as viewing the data on a desktop will have a different look when compared to viewing the same data on a mobile device. We will have to pay very close attention while querying the data and most importantly how we will be querying it. The scalability aspects should also be considered as it’s the most imperative factor to contemplate if the business should thrive for a long time.

Considering certain types of data such as transactional data which are ideal on a SQL database as their data form scarcely alters, any SQL database would offer tailor-made perks for such sorts of data as it ensures a quick query processing for various data analytics. Also guarantees that the data not only remains consistent but also accurate throughout the life cycle. This also means the complete dearth of any sort of changes during any two update phases of the data. However, speaking of the NoSQL database, it would present the possibilities of extreme flexibility along with easy scalability, leading to swift development. Management of a NoSQL database can be done with ease as opposed to its SQL counterpart.

With the availability of numerous open-source relational databases as well as NoSQL databases, we regarded it best to test various databases and then pick one database, or even two if need be, that serves exceptionally to our requirements.



Source: Medium

Figure 2: SQL and NoSQL databases

Our goal is to set up experiments on a prototype system that employs real-time information streaming as well as data processing from IoT devices. These devices will incorporate the different data logistics forums such as Apache Kafka and Apache NiFi to compare which types of datasets are suitable for querying and storing the data and also to automate the data transfer.

2.6 STATEMENT OF THE PROBLEM:

There is a lot of scope for the growth of IoT devices as the world is becoming smarter and smarter with each passing day. It is estimated that there will be roughly 41 billion devices by 2025. It will be tricky and easy to get lost while choosing a database for their application as there are a lot of databases which are available in the market. So, one must be very careful while choosing a database for their applications. Without choosing a database by intuition, one has to compare the performance of multiple databases for their use case.

2.7 AREA OR SCOPE OF INVESTIGATION:

The data generated by the IoT application is utilized to correlate and compare the performances of SQL and NoSQL databases. The overall time consumed to carry out various types of query statements such as the Insert and Select queries based on diverse and several threads and records can be attributed to our area of investigation. With the gradual increase in the total size of the records within the table, it is apparent that the load obligation on the system will also raise equally. This will again lead to an increased latency i.e. the response time of the system will be delayed. We have to also consider the fact that every database will have its advantages and drawbacks. In some scenarios, a NoSQL DB may require less latency when compared to a traditional DB and vice versa.

As part of the investigation scope, we will keep increasing the number of devices that communicate over the network to attain workloads of more and more data thereby building a huge database that can be primarily utilized to test millions of records and eventually resulting in several supplemental test categories.

We also intend to continue this research work by including a study solely based on performance testing with various database management systems that will come into operation. Other scopes of this study can also consider blending the cloud technology with distributed databases which could lend this study to be applied to realistic problems and further improve and increase the attributes of SQL and NoSQL databases.

3 THEORETICAL BASES AND LITERATURE REVIEW

3.1 DEFINITION OF THE PROBLEM

As mentioned earlier, it is expected that there will be 41 billion IoT devices by 2025. As digitalization is getting adopted rapidly with the increase in the usage of electronic devices, finding an optimal database for each use case is the need of the hour and with the advancements in the field of NoSQL, it is quite tricky for someone to go ahead with a database for a particular use case. Instead, one can compare the efficiency of various databases both relational and NoSQL which will aid them to choose a database to best fulfill one's requirements.

3.2 THEORETICAL BACKGROUND OF THE PROBLEM

Millions and millions around the globe are killed due to pollution. An accurate system that captures the air quality data and conveys the key information to the users and alerts them on time to avoid places that are highly polluted is the need of the hour as both governments and various agencies are keeping a close watch on this area. When it comes to tracking air quality data of multiple locations, there will be millions of devices that are connected over the network, they collect and communicate data to the cloud storage system of various parameters presented in the air, below are a few of the common parameters which are tracked to analyze the air quality of a location:

a) Carbon dioxide (CO2)
b) Sulfur dioxide (SO2)
c) Hydrogen sulfide (H2S)
d) Nitrogen dioxide (NO2)
e) Nitric oxide (NO)
f) Carbon monoxide (CO)
g) Particulate matter (PM1-PM17)

Planet Watch is building a global air quality monitoring network to find out pollution hives and safeguard the well-being of humankind. As we have very few pollution detecting sensors on the ground, they are on a mission to set up a network that tracks the pollution levels of the majority of the places where humans are living. We as a team wanted to find out which database serves best to solve their problem. They provide planet watch tokens as perks to people who install their IoT devices to track various parameters which are responsible for the pollution.

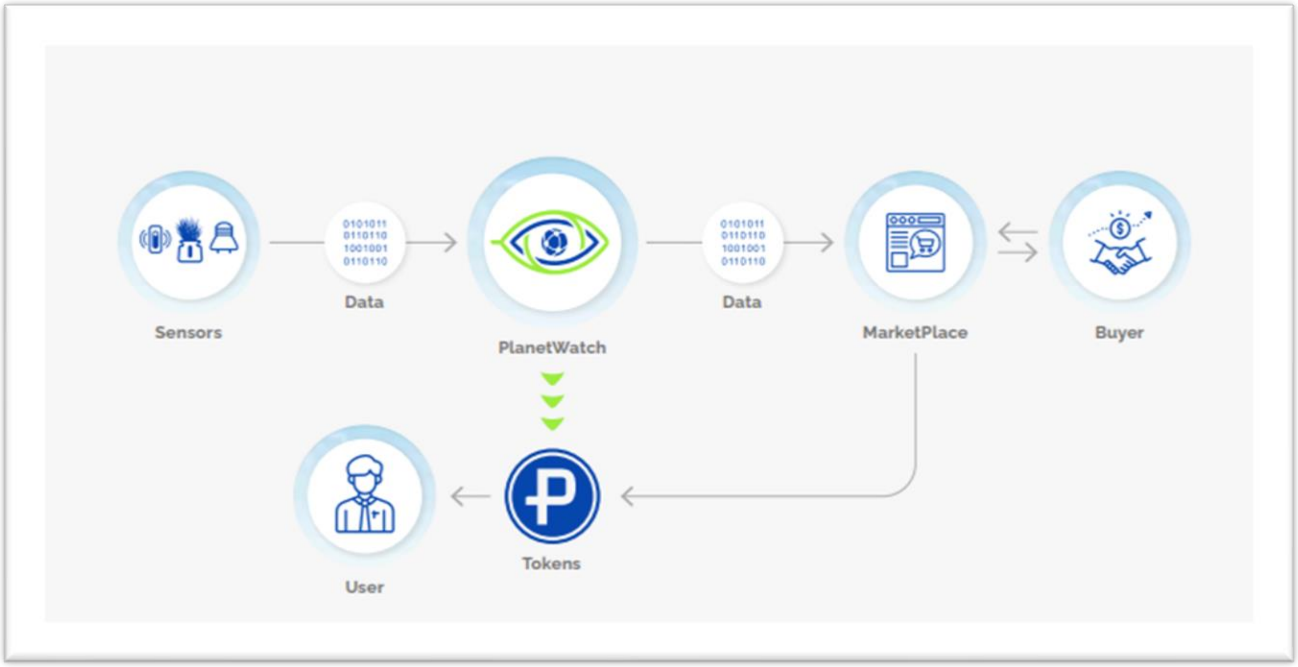


Figure 3: Planet Watch Flow Diagram

By looking at figure 3, we can get a brief overview of how every process is linked up in the Planet Watch ecosystem.

3.3 RELATED RESEARCH TO SOLVE THE PROBLEM

A comparison of MySQL and NoSQL databases for an IoT application related to soil moisture detection is performed by Sharvari et al [1]. The author used data related to the IoT application and fed it to MySQL and MongoDB to find out which best serves the use case. Once the data is uploaded to the database, select(read) and insert(write) operations are performed gauge to the best database concerning the response time of both MySQL and MongoDB.

Moreover, the number of threads is varied and the response times for reading and write operations with multiple thread combinations are calculated and plotted to analyze the performance of both DBs.

In addition to this, Nadia et al [2] used a mix of 6 workloads generated by YCSB to evaluate the performance of three NoSQL stores namely Redis, Cassandra, and MongoDB. Execution time to perform the workloads is the parameter used to study the performance of these three databases.

3.4 ADVANTAGE/DISADVANTAGE OF THOSE RESEARCH

As observed in [1], the data was collected from a single Arduino device and is fed to the database. Ideally, there will be a lot of IoT devices that gather data and communicate the data to the cloud storage. This method is best when there is only one endpoint that has to be tracked, but several sensors increase feeding data to Arduino, and then sending it to DB can become a cumbersome process. Hence, we cannot assume that the conclusion of the paper holds when there are a lot of IoT devices that communicate over the network.

In [2], six hybrid workloads generated by YCSB were used to understand the performance of the three NoSQL stores namely Redis, Cassandra, and MongoDB. In the real-life scenario, there are less likely chances for a hybrid workload, instead, there will be simultaneous read and write requests from various end-users which may increase the response time for the queries as the system has to handle multiple requests. For hybrid workloads, we can assume that the conclusion of the [2] paper holds, whereas, for multiple requests from multiple devices, we never know how things fold up. Moreover, the dataset used for evaluation was fabricated by using the YCSB tool and there is vague information related to the dataset generated by the YCSB. It could have been much better if a real-time dataset was used for analysis along with the data points of the dataset.

3.5 OUR SOLUTION TO SOLVE THIS PROBLEM

We will mimic the real-time data generated by the IoT devices and feed it to the database and then perform analysis using multiple read and write operations on various SQL and NoSQL data stores to gauge the performance in terms of execution time.

Furthermore, we are planning to generate real-time data instead of fabricated data that will be fed into the various databases and track the response time for reading or write operations which will aid us to determine the database for this particular IoT application.

Our approach is to mimic the nature of real-time data and then track the response time and execution time for reading and writing operations by each database to find the best suiting database for air quality detection using IoT devices. For a distributed network with hundreds of thousands of devices, one must always ensure to minimize the latency. An improvement in faster response time as small as 0.1 seconds or even smaller response time for hundreds of thousands of devices will improve the performance of the whole system.

3.6 WHERE OUR SOLUTION IS DIFFERENT FROM OTHERS

Many previous authors used system-generated datasets by directly loading them into the database and then compared the performance of the database. By using this method, they did not consider the real-time problems and there are higher chances that the conclusion of using a DB for a real-time situation might not hold.

We wanted to mimic the data flow of the real-time IoT devices to the storage system and find the optimal database for the use case by calculating the response time for reading and writing operations by the database. By following this method, we are considering real-time operational challenges and as the implementation is much inclined to the live problem, there are higher chances that conclusions from our hypotheses might hold while implementing it on a scale in the actual environment.

3.7 WHY OUR SOLUTION IS BETTER

We are trying to incorporate and include the real-time data collection behavior for air quality detection using IoT devices, as we are sending the real-time data to the storage, performing analyses using this data could mostly hold in real-time situations.

Moreover, with the comparison of the dataset for writing and reading operations for real-time data, we are coming up with the approach to perform analyses and compare databases for the respective use case and how one should go about it. All said and done, we are making sure that we are working on a dataset that is almost similar to the real-time environment.

4. HYPOTHESIS

4.1 SINGLE/MULTIPLE HYPOTHESIS:

1. Compare the scalability of SQL and NoSQL databases.

2. Measure multiple performance parameters on various databases.

3. Assess the flexibility.

4. Examine the system’s maturity.

5. Evaluate and compare the data retrieval durations.

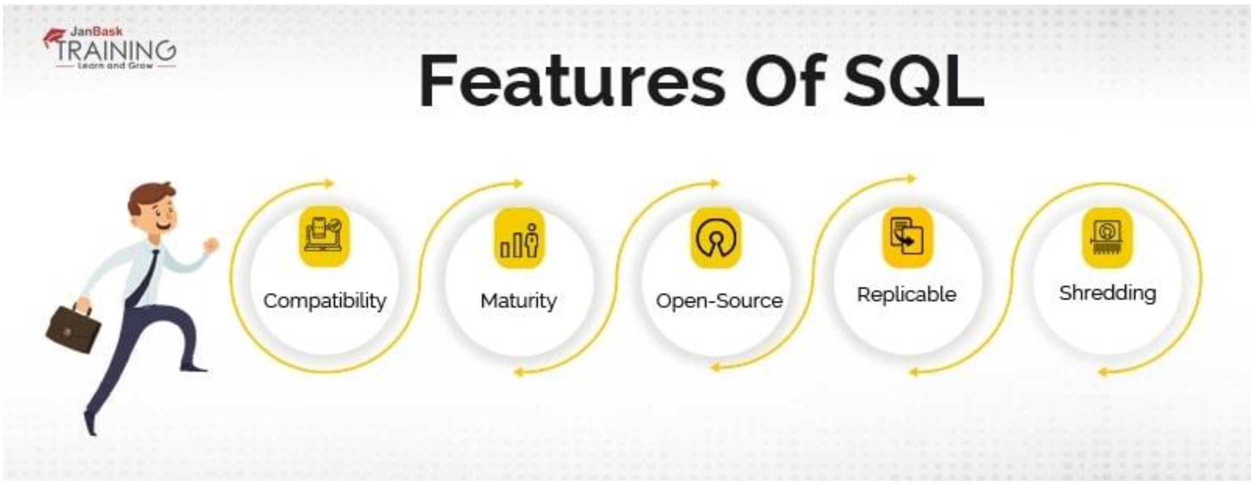
6. Compare Query performance for SQL and NoSQL.

7. Record which Databases are optimal for our use cases.

8. Compare read and write operations for the data we received.

4.2 POSITIVE OR NEGATIVE HYPOTHESIS:

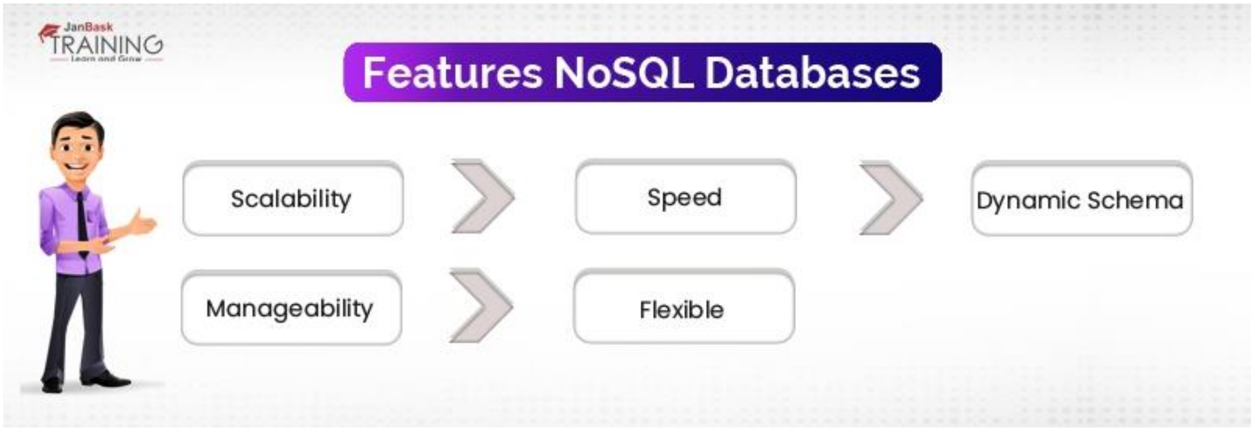
In general, NoSQL databases, like traditional relational databases, have advantages and disadvantages over one another. Regardless, switching from a SQL database to a NoSQL database might be difficult from a variety of perspectives. It is critical, for example, to provide a detailed examination of the two arrangements, their highlights, and their questioning options. However, the invention of NoSQL databases was not intended to destroy the relational database business; rather, it was intended to provide a solution to the shortcomings of both databases. Relational databases, on the other hand, are widely used because of their conventionalism, dependability, and stability. Its continued existence has shown its customers its undeniable excellence.



Source: Janbasktraining

Figure 4: Features of SQL

In comparison to traditional relational database clients, NoSQL clients are inherently fewer, which indicates that this could be a weak test for NoSQL databases in convincing new clients to adopt this new arrangement. Furthermore, the lack of a consistent query processing language may contribute to user apprehension about using NoSQL databases. Overall, while choosing the right database for a specific product application, we must consider some essential database criteria, such as query processing language, data availability, flexibility, data replication, performance, and scalability.



Source: Janbasktraining

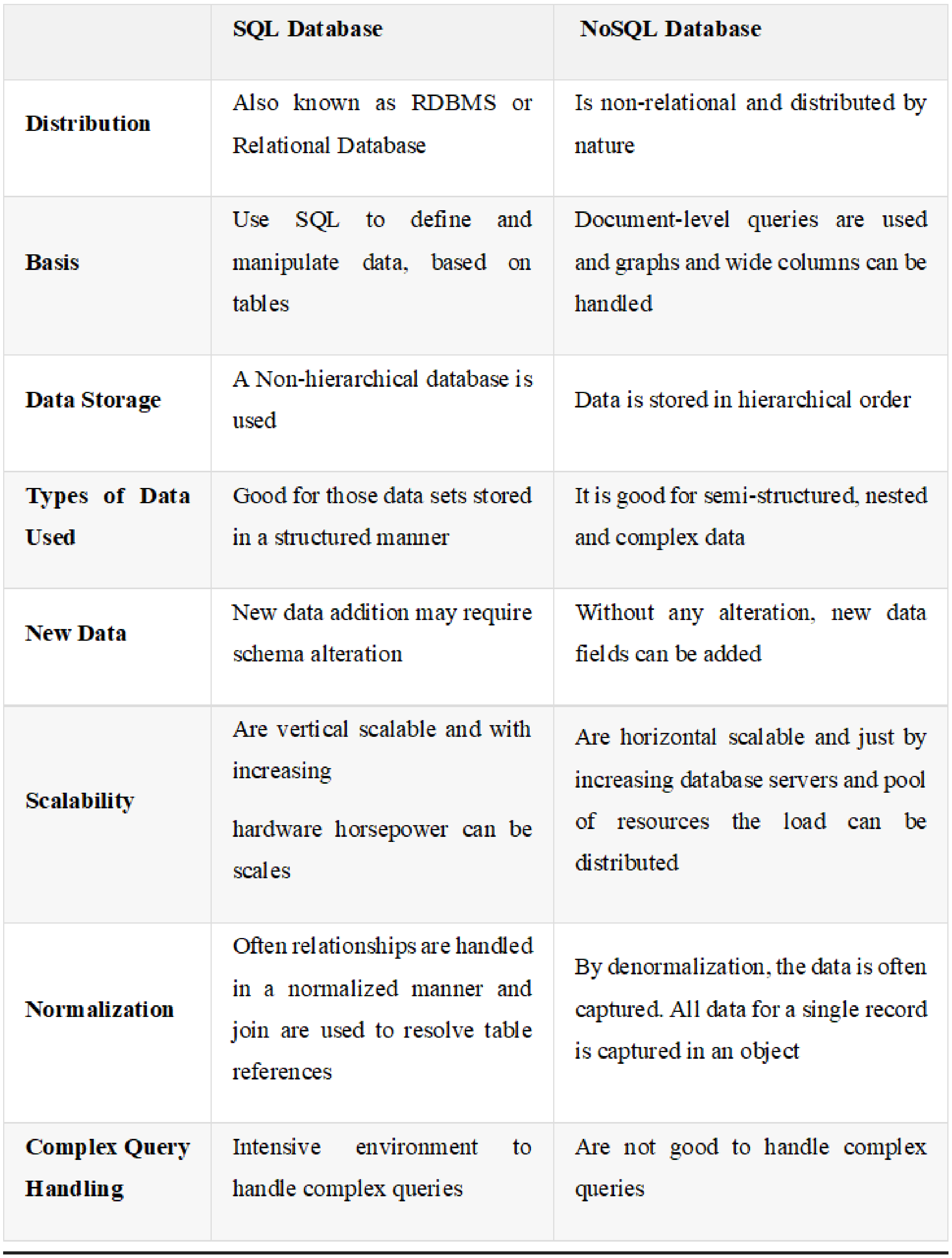
Figure 5: Features of NoSQL databases

When ACID properties are prioritized, the analysis has shown that using a relational database (SQL) is the better option. In contrast to relational databases, NoSQL databases feature a more adaptable paradigm, making it easier to search through a large amount of data with different arrangements and with adaptive increments over time. There is always the need to make modifications to the database schema when there are large datasets, and there is a necessity for both flexibility and performance, which makes NOSQL the appropriate choice.

Even though NoSQL databases are well-versed in large-scale data development and can easily handle large volumes of data, they lack security features and must rely on an external technique to perform and ensure database security. But then, given the expansion and development of NoSQL databases, the prospects are unquestionably bright.

Understanding the differences between SQL and NoSQL:

To reiterate the intricacies of NoSQL and SQL databases, we have divided and compared both the databases based on several attributes such as Distribution, Basis, Data Storage, Types of Data, Used, New Data, Scalability, Normalization, Complex Query Handling, Data Set Size, Support and Adoption, ACID Properties. These attributes provide broader and deeper insights into not only the differences but also the performances of each of the databases.



	SQL Database	NoSQL Database
Distribution	Also known as RDBMS or Relational Database	Is non-relational and distributed by nature
Basis	Use SQL to define and manipulate data, based on tables	Document-level queries are used and graphs and wide columns can be handled
Data Storage	A Non-hierarchical database is used	Data is stored in hierarchical order
Types of Data Used	Good for those data sets stored in a structured manner	It is good for semi-structured, nested and complex data
New Data	New data addition may require schema alteration	Without any alteration, new data fields can be added
Scalability	Are vertical scalable and with increasing hardware horsepower can be scales	Are horizontal scalable and just by increasing database servers and pool of resources the load can be distributed
Normalization	Often relationships are handled in a normalized manner and join are used to resolve table references	By denormalization, the data is often captured. All data for a single record is captured in an object
Complex Query Handling	Intensive environment to handle complex queries	Are not good to handle complex queries

Source: Janbasktraining

Table 2: Comparing SQL and NoSQL databases

5. METHODOLOGY

5.1 HOW TO GENERATE/COLLECT INPUT DATA

As mentioned earlier, our research work is inspired by Planet Watch, they are currently deploying four types of devices, namely Type 1, Type 2, Type 3, and Type 4 IoT devices. We are capturing the concentration of CO2, CO, SO2, NO2, NO, H2S, and PM2.5 in the air. To generate/ collect data for tracking air quality, we have come up with a data schema as shown below in table 2 and table 3.

Table 3: IoT real-time tracking data

Fields	Datatype	Explanation
CO	Integer	The concentration of Carbon Monoxide in the air (ppb)
NO2	Integer	The concentration of Nitrogen Dioxide in the air (µg/m3)
CO2	Integer	The concentration of Carbon Dioxide in the air (ppb)
H2S	Integer	The concentration of H2S in the air (ppb)
PM 2.5	Integer	The concentration of PM2.5 in the air (µg/m3)
SO2	Integer	The concentration of SO2 in the air (µg/m3)
NO	Integer	The concentration of Nitrogen Oxide in the air (µg/m3)
date_time	DateTime	GMT timestamp of each row, we are using GMT so that a standard date is being used from all IoT devices
pkey	Integer	The primary key of the data will be an increment for each row starting from 1.
Device ID	Varchar	The ID of the device which is communicating the data

Table 4: Device details

Column	Datatype	Information
Device Id	Varchar	The ID of the device which is communicating the data
Latitude	Decimal (10,8)	The latitude where the IoT device is located
Longitude	Decimal (11,8)	Longitude where the IoT device is located

Data in table 3 are communicated regularly and data in table 4 is subject to changes when a device is moved from its location, we are going with an assumption that the chance of changing the location of a device can change in very rare situations. Moreover, we are considering 8 decimal places for latitude and longitude which will give us more accuracy to track the air quality data. Up to 111 kilometers is given by the unit digit (decimal degree) (60 nautical miles, or 69 miles). It informs us of our approximate location in a country. The importance of each decimal position is explained in detail in table 5.

Table 5: Level of Accuracy v/s decimal places of latitude and longitude

DECIMAL PLACE	DISTANCE
Unit digit	111 km
First decimal place	11.1 km
Second decimal place	1.1 km
Third decimal place	110 m
Fourth decimal place	11 m
Fifth decimal place	1.1 m
Sixth decimal place	0.11 m
Seventh decimal place	11 mm
Eighth decimal place	1.1 mm

The data is collected by the IoT devices and then streamed to the cloud storage, Planet Watch has set the limit of streams as they have to reward users for each stream. The limit of the number of streams for each type of device is shown in figure 2.

As mentioned earlier, the number of streams is capped as Planet Watch has put a restriction on the number of coins that are to be released in a year and expected devices that are to be installed according to the roadmap.

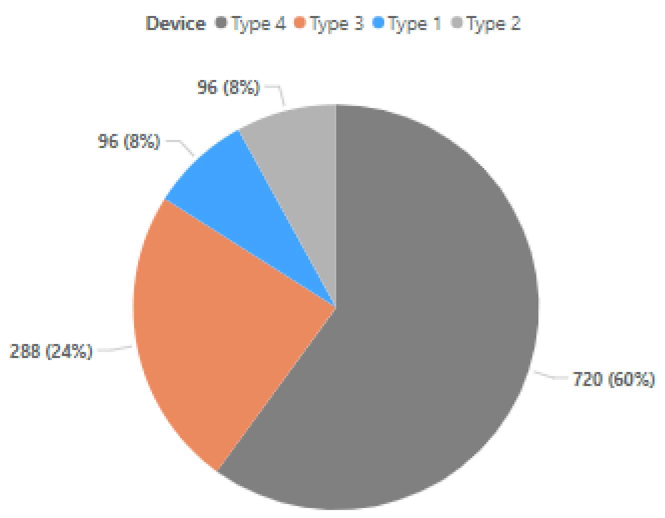


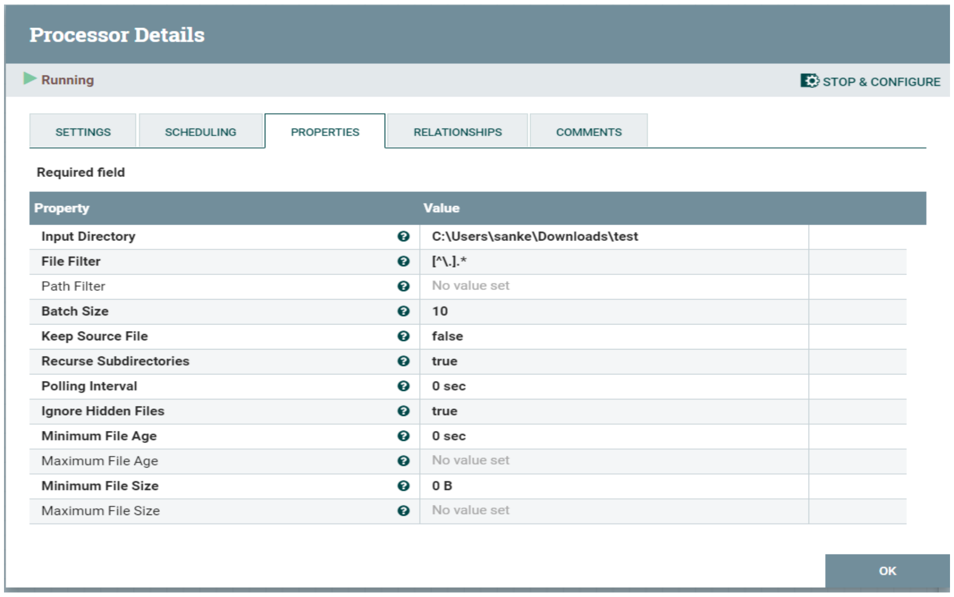
Figure 6: Number of Streams for each type of IoT device

Our interest is to study the IoT device of Type 1. We are focusing on the data in the table1 device and with the help of a python script, we are continuously generating the data using NiFi we are picking up the data and then pushing the data to the storage system. The data generated is randomly selected from a data point in the typical concentration range of each pollutant. The focus here is to mimic the real-time behavior by continuously generating the data and pushing it to the storage system using NiFi.

5.2 HOW TO SOLVE THE PROBLEM

A python script continuously generates data in the format mentioned above and stores the generated data in the form of a .json file, Once the output file has been generated, NIFI fetches the file and removes the file in the specified path.

The processor used in NIFI to fetch the file is the Get file. We have used the following properties in the getfile processor to accomplish the above-mentioned task.



Processor Details

Running | STOP & CONFIGURE

SETTINGS | SCHEDULING | PROPERTIES | RELATIONSHIPS | COMMENTS

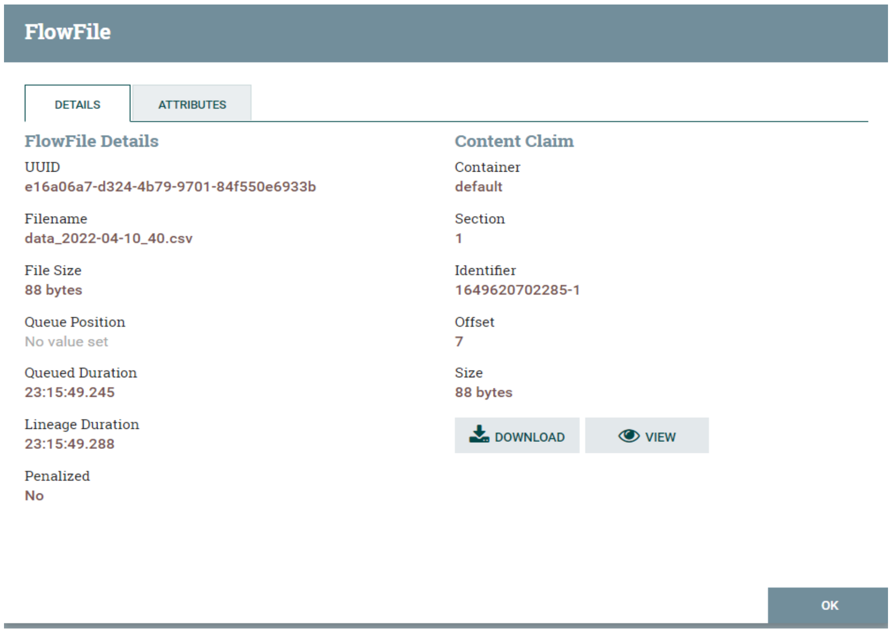
Required field

Property	Value
Input Directory	C:\Users\sanke\Downloads\test
File Filter	[^\.].*
Path Filter	No value set
Batch Size	10
Keep Source File	false
Recurse Subdirectories	true
Polling Interval	0 sec
Ignore Hidden Files	true
Minimum File Age	0 sec
Maximum File Age	No value set
Minimum File Size	0 B
Maximum File Size	No value set

OK

Figure 7: getfile processor properties

Once the file has been successfully fetched by NiFi, File flow details will be generated as shown in figure 8. We can check for the file attributes by clicking on the attributes tab as shown in figure 8.



FlowFile

DETAILS | ATTRIBUTES

FlowFile Details

UUID
e16a06a7-d324-4b79-9701-84f550e6933b

Filename
data_2022-04-10_40.csv

File Size
88 bytes

Queue Position
No value set

Queued Duration
23:15:49.245

Lineage Duration
23:15:49.288

Penalized
No

Content Claim

Container
default

Section
1

Identifier
1649620702285-1

Offset
7

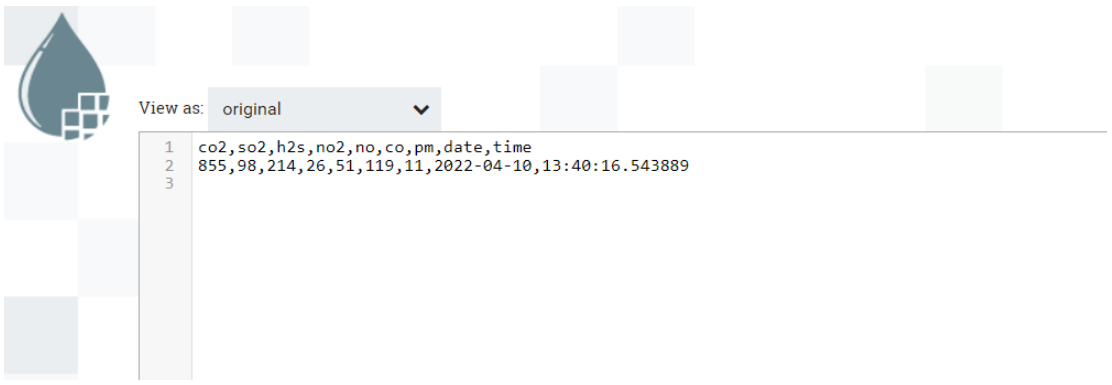
Size
88 bytes

DOWNLOAD | VIEW

OK

Figure 8: File Flow details and attributes

Once the data is successfully fetched, we can preview the data by clicking on the view option which is at the bottom right of the file flow details as mentioned in figure 8. A preview of the fetched data is shown in figure 9.



```
co2,so2,h2s,no2,no,co,pm,date,time
855,98,214,26,51,119,11,2022-04-10,13:40:16.543889
```

Figure 9: Preview of fetched data

5.2.1 ALGORITHM DESIGN

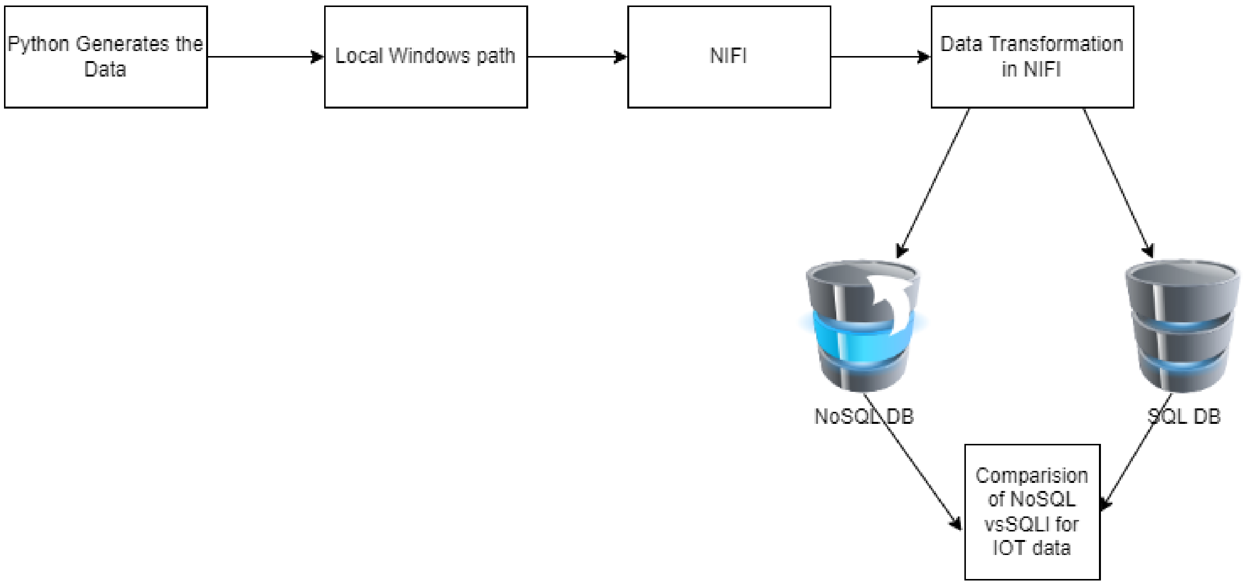

Python Generates the Data
Local Windows path
NIFI
Data Transformation in NIFI
NoSQL DB
SQL DB
Comparision of NoSQL vsSQLl for IOT data

Figure 10: Algorithm

5.2.2 LANGUAGES USED:

- Python
- SQL

5.2.3 TOOLS USED:

- NIFI
- Jupyter Notebook
- MySQL Workbench
- Tableau

6. IMPLEMENTATION

We have collected Sensor data and replicated the data in python which resembles the actual Sensor Data. Python will generate the data every 5 seconds and store it in our local machine.

NiFi which is a lightweight ETL tool will fetch the files from the local. Once NiFi fetches the file, the file will be removed from the local machine. The data will contain many unnecessary characters. So, we will transform the data into JSON.

Once the data is transformed we will store the data in databases i.e. MySQL and Mongo DB. We have utilized the JDBC gateway to achieve this. Irrespective of success or failure we will receive a response from the databases and if it is a success we will calculate the time taken from the moment it executes a database query to the moment we have received a response. Once we have calculated the time we will store the time is taken and the unique primary key of the data in the database using a gateway.

We have used Tableau to analyze and compare the performance of MySQL and MongoDB for write, delete and update operations. In addition to this, we have also compared the performance for bulk insertion and read operations on both databases.

6.1 DESIGN DOCUMENT AND FLOW CHART:

When we first started this project, instead of just comparing the SQL and NoSQL for IoT applications, we wanted to mimic real-life scenarios. So, we wanted to check how an external tool interacts with databases and which can receive the data from IoT applications.

Using NIFI was our safest bet, as it can connect to MQTT Directly, this will align with the real-time data extraction from IoT devices. We can connect to the databases by utilizing gateways because of its user-friendly interface, we can utilize the drag and drop option to perform many inbuilt operations.

FLOW CHART:

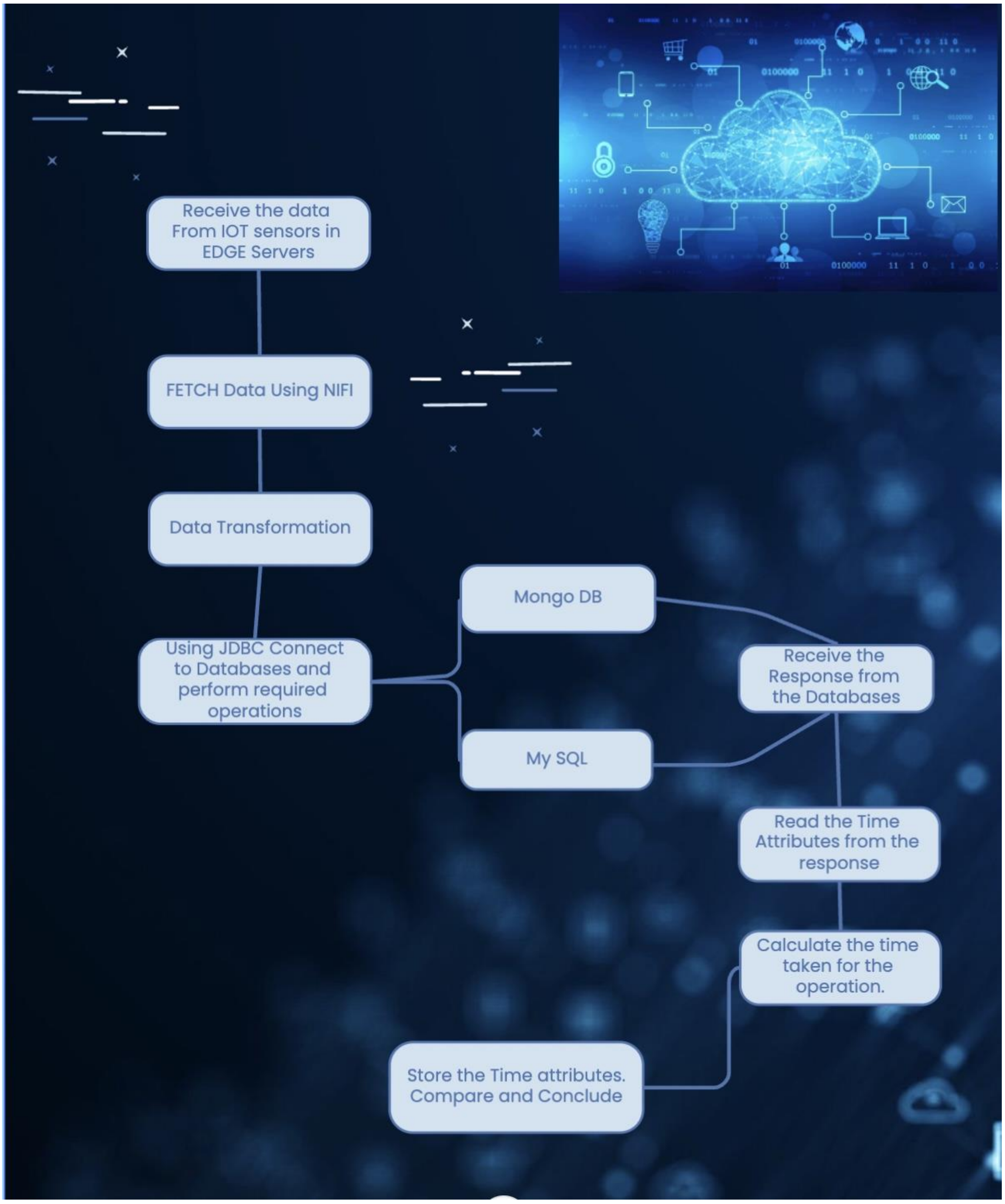

Receive the data From IOT sensors in EDGE Servers
FETCH Data Using NIFI
Data Transformation
Using JDBC Connect to Databases and perform required operations
Mongo DB
My SQL
Receive the Response from the Databases
Read the Time Attributes from the response
Calculate the time taken for the operation.
Store the Time attributes. Compare and Conclude

Figure 11: Flowchart

Once we have stored the time taken for the databases, we can compare the time taken for different databases and can choose the best database as per our requirements.

As we have used the real-life scenario which just resembles the real-time streaming of IoT data, this mechanism will help us to choose the best database and is more likely to align with the real-time challenges we face when sending the data to any database.

6.2 DEVICE CONFIGURATION:

Hardware	**Software**
CPU: 8 core CPU	Python - 3.9.11
GPU: 14-Core	NIFI - 1.16.0
Processor: Apple M1 Pro	MySQL - 8.0.29
RAM: 16GB	Mac version - 12.3.1
SSD: 512GB	MongoDB - 4.4.13
	Tableau – 2022.1

7. DATA ANALYSIS AND DISCUSSION

7.1 OUTPUT GENERATION:

The analysis commences with the data generated from various sensors on the EDGE server. The data is then retrieved using NiFi and necessary data wrangling is done. We then make use of the JDBC server to connect and transfer the data to MySQL and MongoDB databases. All the required operations are performed on each of the databases and the intended data based on which we can conclude our views is studied. We are principally focusing on the time comparison of each of the databases and validating our study accordingly.

7.2 OUTPUT ANALYSIS:

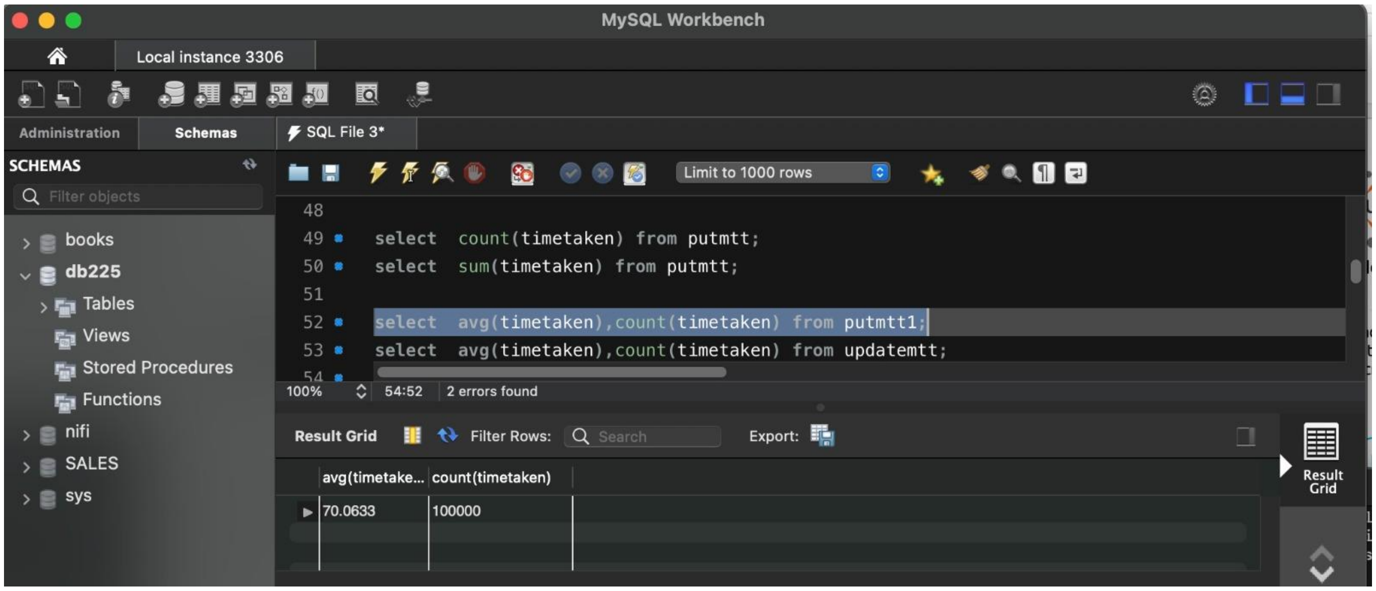
As mentioned in the previous sections, we will be comparing the time taken for each operation in both the databases i.e. MySQL and MongoDB. The operations that we implement in both the databases are:

(i) Row-by-row Insert
(ii) Row-by-row Update
(iii) Row-by-row Delete
(iv) Bulk Insertion
(v) Bulk Read

The analysis for each of the operations can be seen in the below figures:

(i) **<u>Insert Operation:</u>**

(a) MySQL: The time taken to insert 100000 rows of data into a MySQL database is around 70 ms.

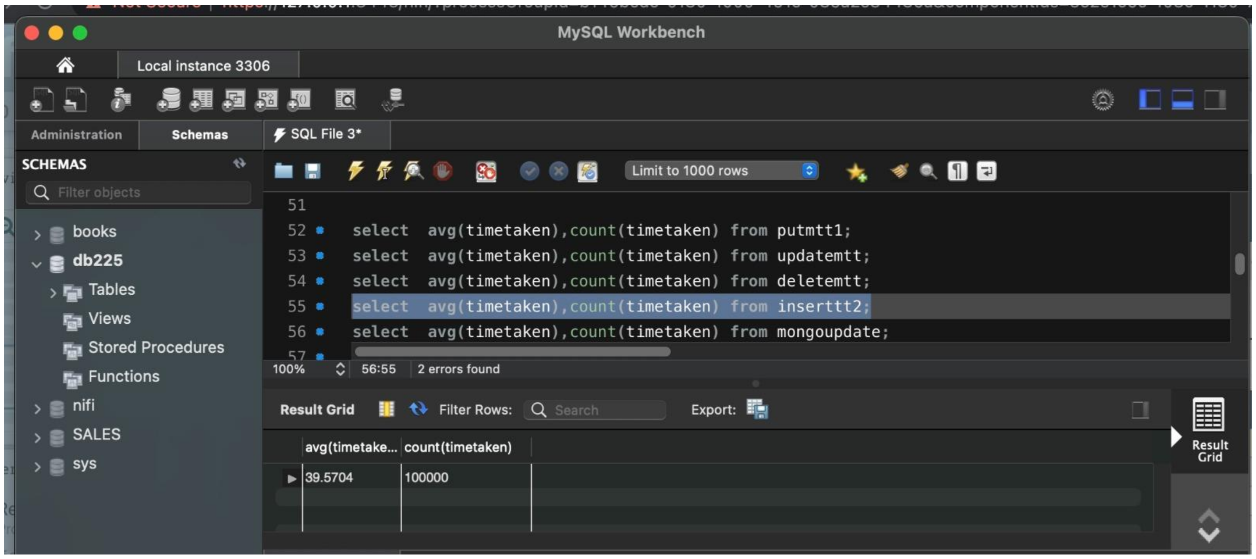


```
select  count(timetaken) from putmtt;
select  sum(timetaken) from putmtt;

select  avg(timetaken),count(timetaken) from putmtt1;
select  avg(timetaken),count(timetaken) from updatemtt;
```

```
avg(timetake...  count(timetaken)
70.0633          100000
```

Figure 12: Insert operation in MySQL

(b) MongoDB: The time taken to insert 100000 rows of data into a Mongo database is around 39.5 ms which is just a little more than half of what it took for a MySQL database.



```
select  avg(timetaken),count(timetaken) from putmtt1;
select  avg(timetaken),count(timetaken) from updatemtt;
select  avg(timetaken),count(timetaken) from deletemtt;
select  avg(timetaken),count(timetaken) from inserttt2;
select  avg(timetaken),count(timetaken) from mongoupdate;
```

```
avg(timetake...  count(timetaken)
39.5704          100000
```

Figure 13: Insert Operation in MongoDB

- MongoDB performs faster than a relational database for insert operations.

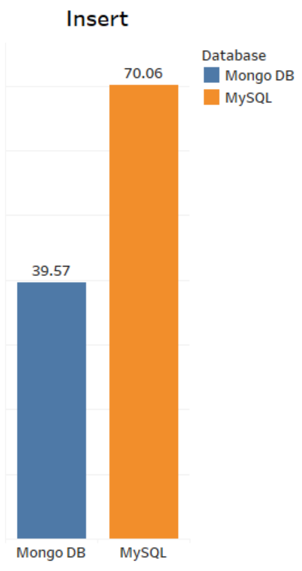
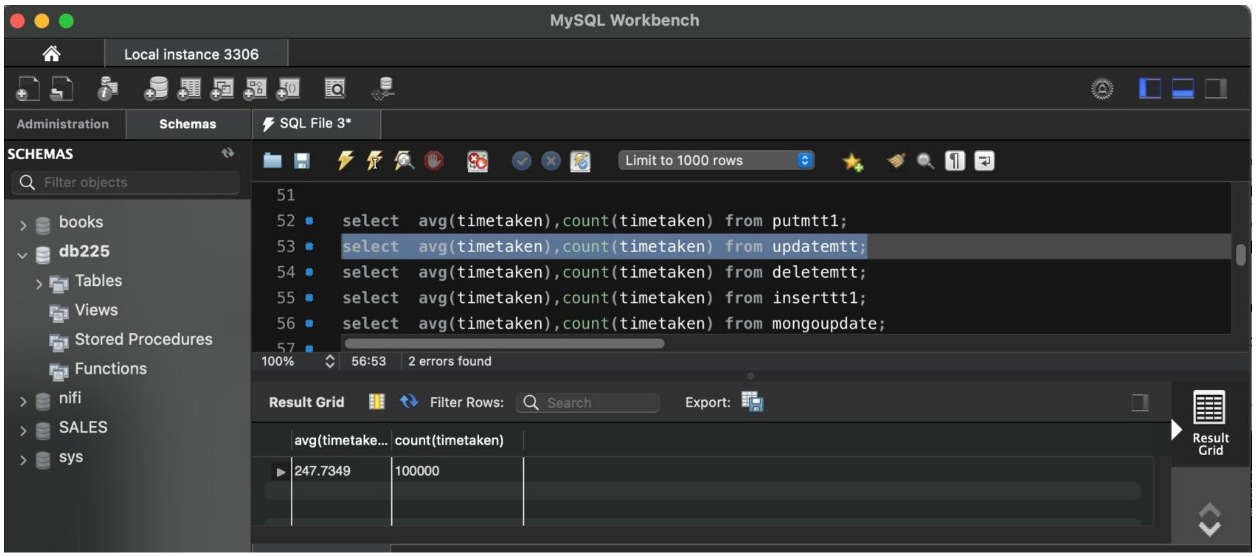


Figure 14: Comparison of the Insert operation

(ii) **<u>Update Operation:</u>**

(a) MySQL: The time taken to update 10000 rows of data within a MySQL database is about 247.7 ms.

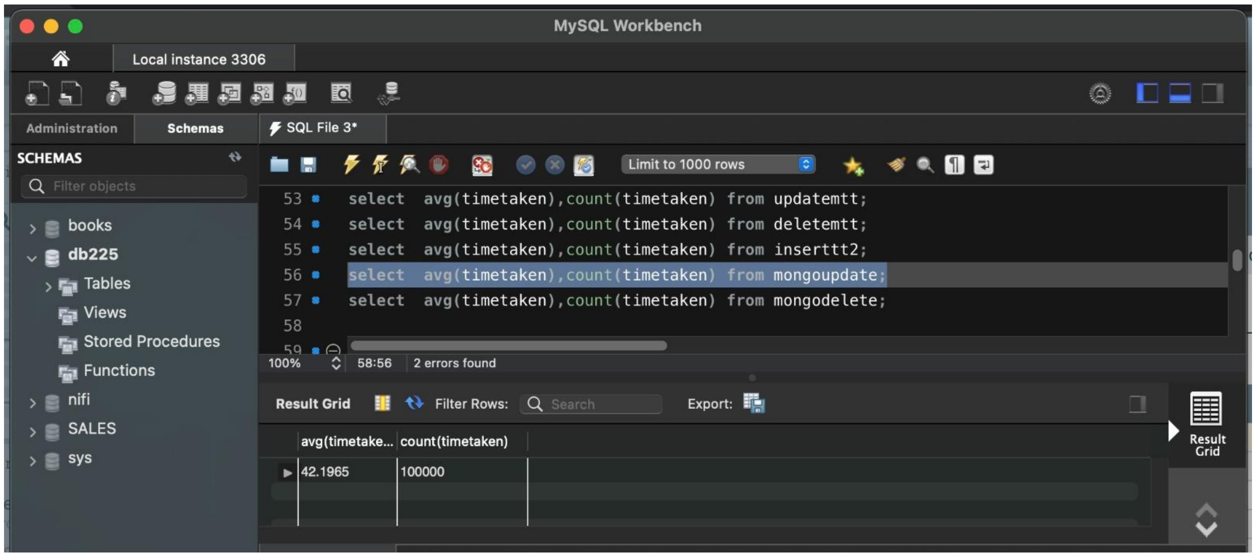


```
select  avg(timetaken),count(timetaken) from putmtt1;
select  avg(timetaken),count(timetaken) from updatemtt;
select  avg(timetaken),count(timetaken) from deletemtt;
select  avg(timetaken),count(timetaken) from inserttt1;
select  avg(timetaken),count(timetaken) from mongoupdate;
```

avg(timetake...	count(timetaken)
247.7349	100000

Figure 15: Update Operation in MySQL

(b) MongoDB: The time taken to update 10000 rows of data within a Mongo database is about 42 ms.



```
select  avg(timetaken),count(timetaken) from updatemtt;
select  avg(timetaken),count(timetaken) from deletemtt;
select  avg(timetaken),count(timetaken) from inserttt2;
select  avg(timetaken),count(timetaken) from mongoupdate;
select  avg(timetaken),count(timetaken) from mongodelete;
```

avg(timetake...	count(timetaken)
42.1965	100000

Figure 16: Update Operation in MongoDB

- Update operation in MongoDB is executed at least 5 times faster than that of MySQL database rendering MongoDB to be far superior for update operations.

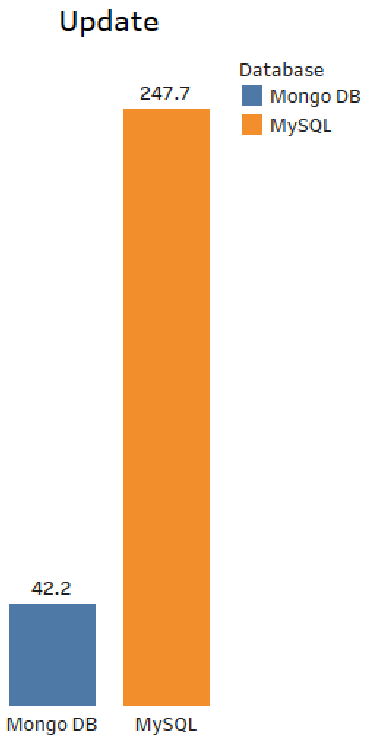
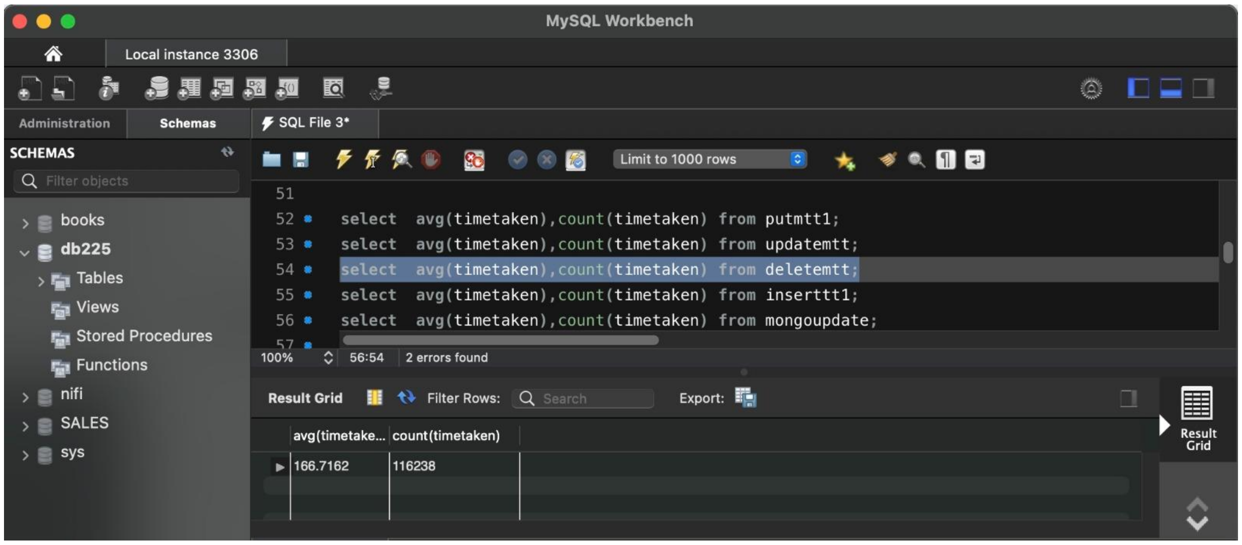


Figure 17: Comparison of Update Operation

(iii) <u>Delete Operation:</u>

(a) MySQL: The time taken to delete data within a MySQL database is about 166.7 ms.

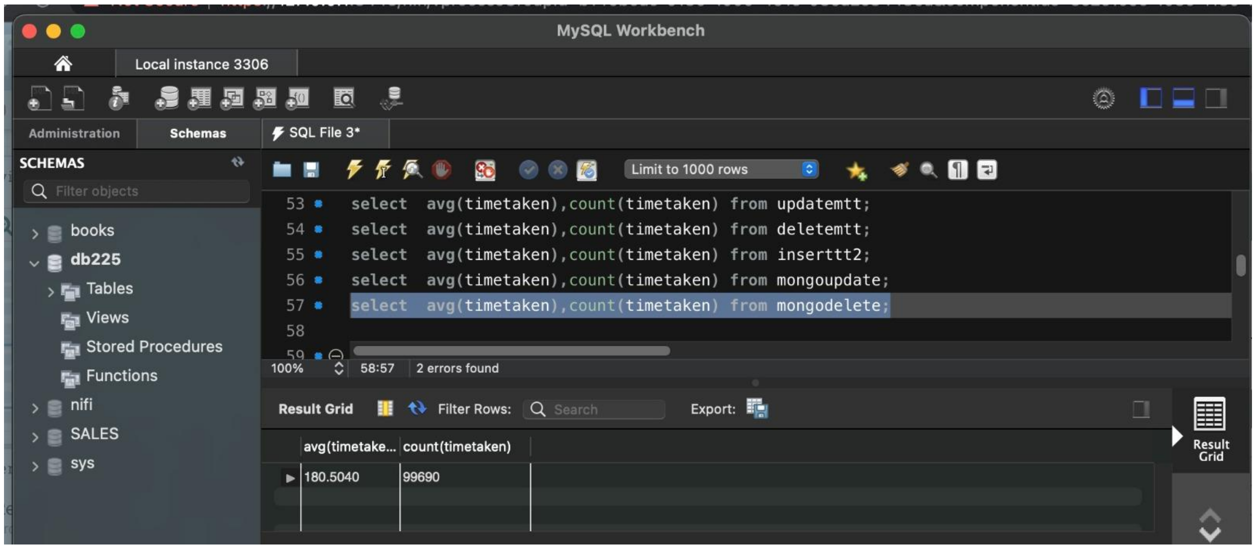


```
select  avg(timetaken),count(timetaken) from putmtt1;
select  avg(timetaken),count(timetaken) from updatemtt;
select  avg(timetaken),count(timetaken) from deletemtt;
select  avg(timetaken),count(timetaken) from inserttt1;
select  avg(timetaken),count(timetaken) from mongoupdate;
```

avg(timetake...	count(timetaken)
166.7162	116238

Figure 18: Delete Operation in MySQL

(b) MongoDB: The time taken to delete data within a Mongo database is about 180.5 ms.



```
select  avg(timetaken),count(timetaken) from updatemtt;
select  avg(timetaken),count(timetaken) from deletemtt;
select  avg(timetaken),count(timetaken) from inserttt2;
select  avg(timetaken),count(timetaken) from mongoupdate;
select  avg(timetaken),count(timetaken) from mongodelete;
```

avg(timetake...	count(timetaken)
180.5040	99690

Figure 19: Delete Operation in MongoDB

- MongoDB falls behind MySQL for delete operations.

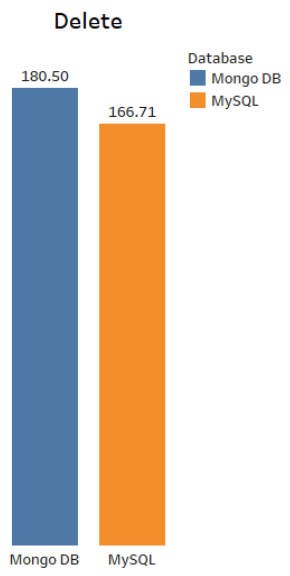
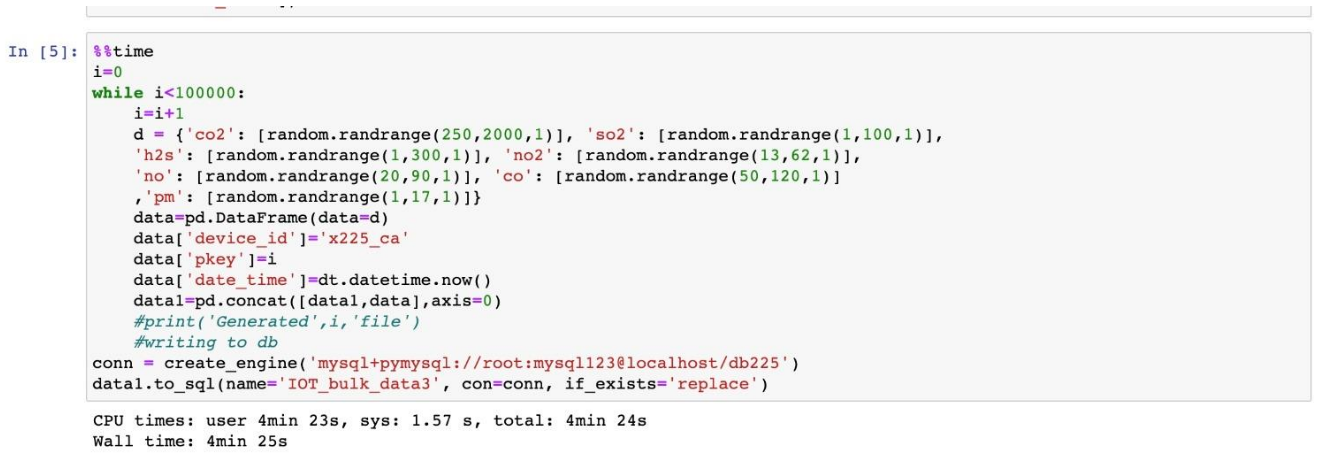


Figure 20: Comparison of Delete Operation

(iv) **Bulk Insertion Operation:**

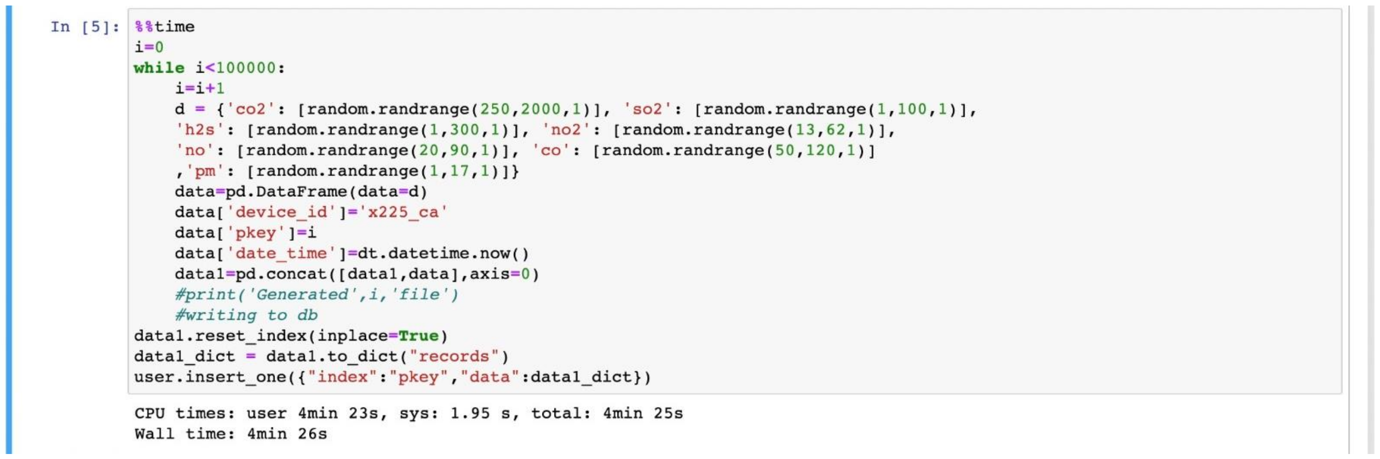
(a) **MySQL:** The time taken for bulk insertion of data into a MySQL is about 4 mins and 25 sec.



```
In [5]: %%time
        i=0
        while i<100000:
            i=i+1
            d = {'co2': [random.randrange(250,2000,1)], 'so2': [random.randrange(1,100,1)],
            'h2s': [random.randrange(1,300,1)], 'no2': [random.randrange(13,62,1)],
            'no': [random.randrange(20,90,1)], 'co': [random.randrange(50,120,1)]
            ,'pm': [random.randrange(1,17,1)]}
            data=pd.DataFrame(data=d)
            data['device_id']='x225_ca'
            data['pkey']=i
            data['date_time']=dt.datetime.now()
            data1=pd.concat([data1,data],axis=0)
            #print('Generated',i,'file')
            #writing to db
        conn = create_engine('mysql+pymysql://root:mysql123@localhost/db225')
        data1.to_sql(name='IOT_bulk_data3', con=conn, if_exists='replace')

        CPU times: user 4min 23s, sys: 1.57 s, total: 4min 24s
        Wall time: 4min 25s
```

Figure 21: Bulk Insertion in MySQL

(b) **MongoDB:** The time taken for bulk insertion of data into a MySQL is about 4 mins and 26 sec.



```
In [5]: %%time
        i=0
        while i<100000:
            i=i+1
            d = {'co2': [random.randrange(250,2000,1)], 'so2': [random.randrange(1,100,1)],
            'h2s': [random.randrange(1,300,1)], 'no2': [random.randrange(13,62,1)],
            'no': [random.randrange(20,90,1)], 'co': [random.randrange(50,120,1)]
            ,'pm': [random.randrange(1,17,1)]}
            data=pd.DataFrame(data=d)
            data['device_id']='x225_ca'
            data['pkey']=i
            data['date_time']=dt.datetime.now()
            data1=pd.concat([data1,data],axis=0)
            #print('Generated',i,'file')
            #writing to db
        data1.reset_index(inplace=True)
        data1_dict = data1.to_dict("records")
        user.insert_one({"index":"pkey","data":data1_dict})

        CPU times: user 4min 23s, sys: 1.95 s, total: 4min 25s
        Wall time: 4min 26s
```

Figure 22: Bulk Insertion in MongoDB

- We can see similar time consumed for bulk insertion in both the databases but MySQL has a slightly better edge as it took a second less than MongoDB.

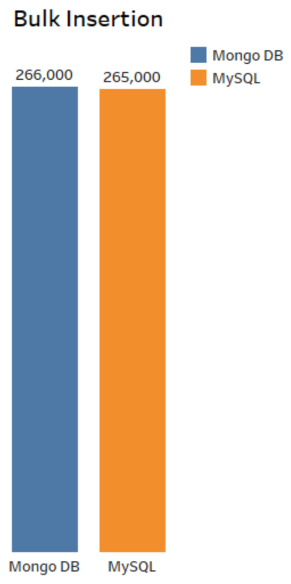
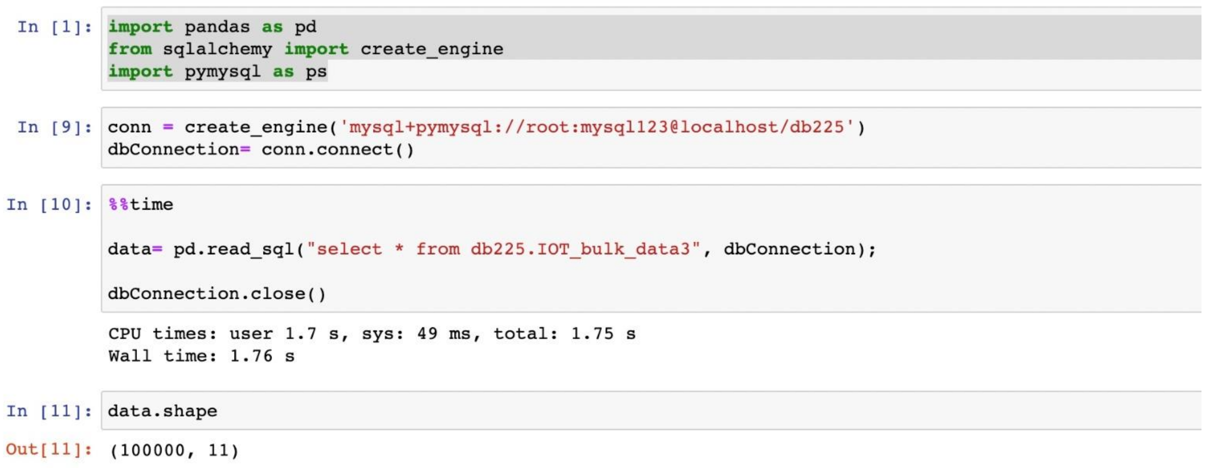


Figure 23: Comparison of Bulk Insertion Operation

(v) **Bulk Read Operation:**

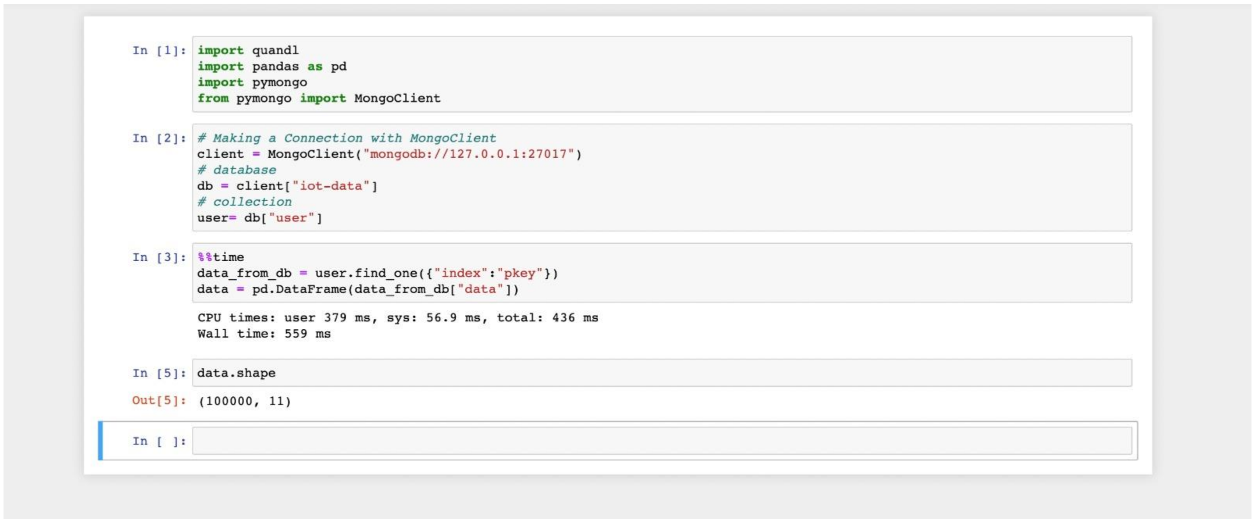
(a) MySQL: The time taken for the bulk read of data into a MySQL database is about 1.76 sec.



```
In [1]:  import pandas as pd
         from sqlalchemy import create_engine
         import pymysql as ps

In [9]:  conn = create_engine('mysql+pymysql://root:mysql123@localhost/db225')
         dbConnection= conn.connect()

In [10]: %%time

         data= pd.read_sql("select * from db225.IOT_bulk_data3", dbConnection);

         dbConnection.close()

         CPU times: user 1.7 s, sys: 49 ms, total: 1.75 s
         Wall time: 1.76 s

In [11]: data.shape

Out[11]: (100000, 11)
```

Figure 24: Bulk Read in MySQL

(b) **MongoDB:** The time taken for the bulk read of data into a Mongo database is about 559 ms.



```
In [1]: import quandl
        import pandas as pd
        import pymongo
        from pymongo import MongoClient

In [2]: # Making a Connection with MongoClient
        client = MongoClient("mongodb://127.0.0.1:27017")
        # database
        db = client["iot-data"]
        # collection
        user= db["user"]

In [3]: %%time
        data_from_db = user.find_one({"index":"pkey"})
        data = pd.DataFrame(data_from_db["data"])

        CPU times: user 379 ms, sys: 56.9 ms, total: 436 ms
        Wall time: 559 ms

In [5]: data.shape

Out[5]: (100000, 11)

In [ ]:
```

Figure 25: Bulk Read in MongoDB

- MongoDB turns out to be more than twice as fast as MySQL for bulk reading operations.

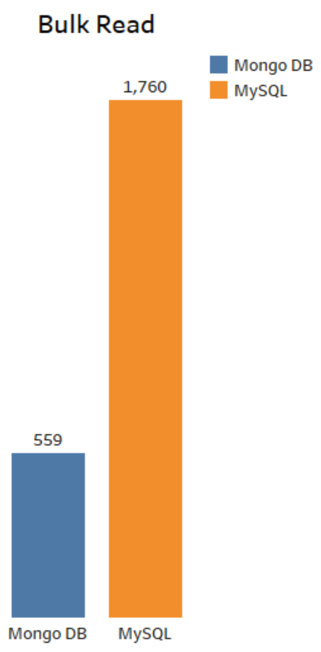
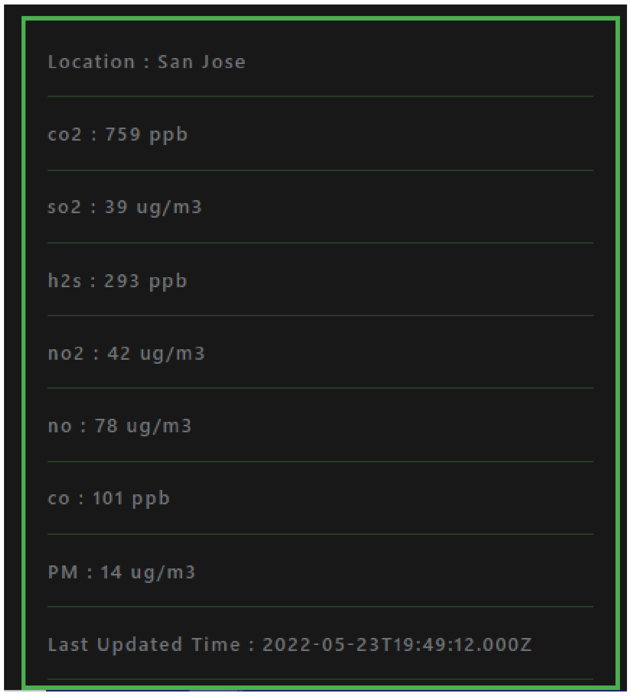


Figure 26: Comparison of Bulk Read Operation

In addition to this, we have developed a website and an interactive real-time dashboard to show the latest reading from the DB along with the updated time as shown in Figure 27 and Figure 28.



Location : San Jose

co2 : 759 ppb

so2 : 39 ug/m3

h2s : 293 ppb

no2 : 42 ug/m3

no : 78 ug/m3

co : 101 ppb

PM : 14 ug/m3

Last Updated Time : 2022-05-23T19:49:12.000Z

Figure 27: Data Published on the website

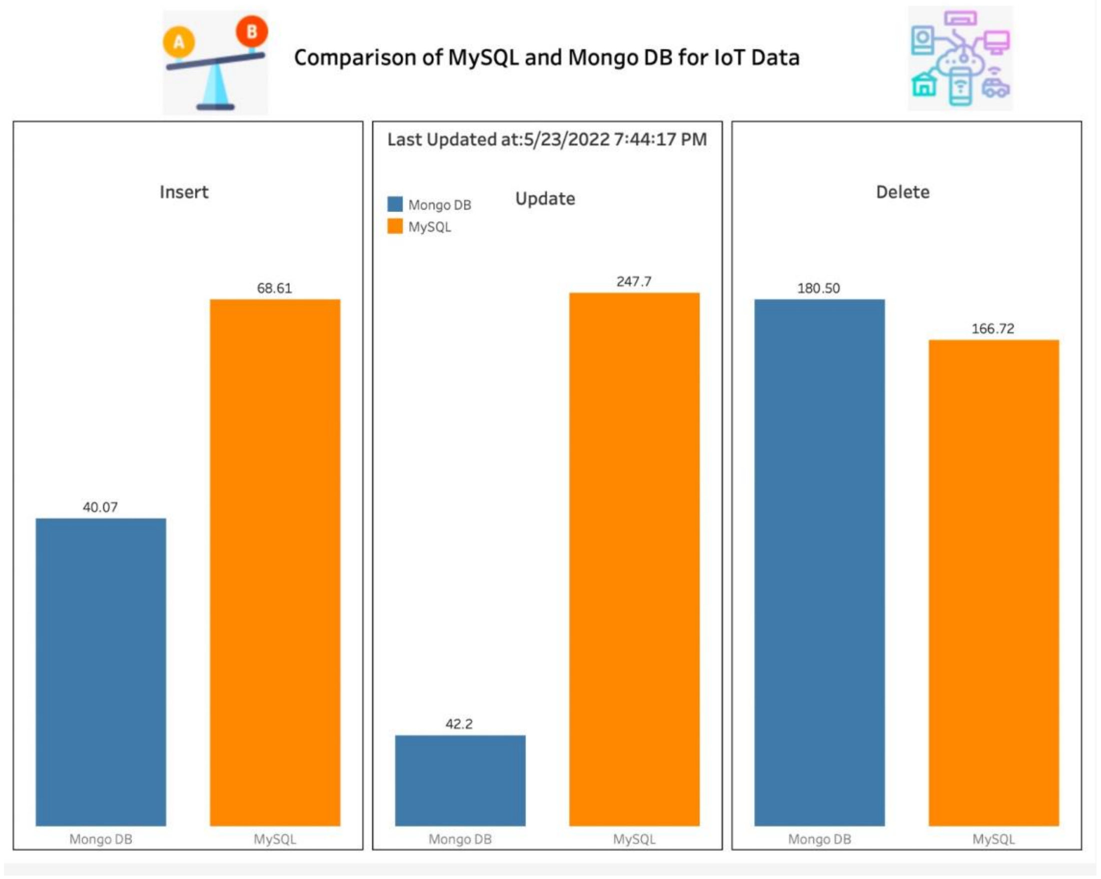


Figure 28: Real-time Tableau Dashboard

The data in the dashboard will be refreshed as the operations are performed on MySQL and MongoDB.

7.3 COMPARING OUTPUT AGAINST HYPOTHESIS:

Given the features of a NoSQL database, we believed that MongoDB would trump a relational database such as MySQL in various aspects. As the data generated from the IoT application is largely unstructured, we came up with the hypothesis that MongoDB would be the most appropriate platform to store and run the data for analysis. The same was found to be true after our study with several operations that we compared between both the databases. While the output stood true to the hypothesis on most occasions, it was only the delete operation with which the MySQL had a quicker performance. MongoDB had better performances, nearly 5 times better than MySQL when it came to the operation speed to update the data. Further proving the hypothesis that MongoDB would be ideal for real-time analytics in comparison to MySQL.

7.4 ABNORMAL CASE OBSERVATION:

While inserting row by row, MongoDB outperformed MySQL whereas when we pushed bulk data into the database, MySQL performed better when compared to Mongo DB.

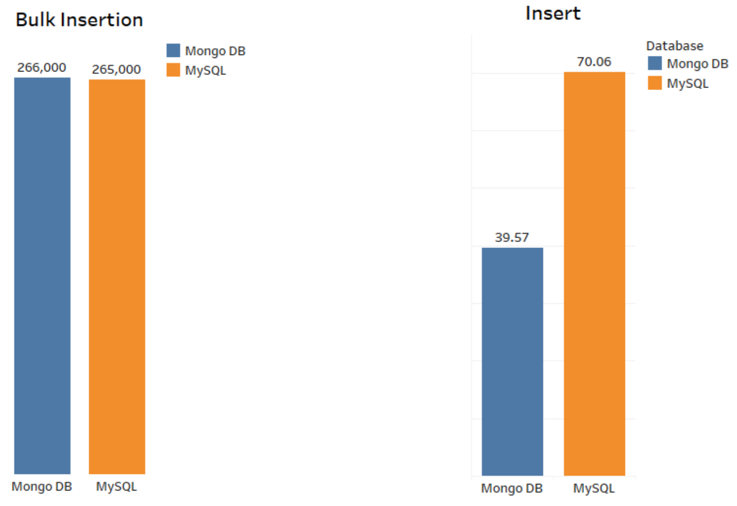


Figure 29: Abnormal case observation

7.5 DISCUSSION:

We followed a multilayered (three-tier) architecture for the data warehousing. For the bottom layer or warehouse data server, where the data is extracted via the gateway, we have utilized Java Database Connection (JDBC) as the gateway. As the middle layer or OLAP server a multidimensional OLAP (MOLAP) and the top of the front-end client layer which is a query and reporting/analysis/mining tool to know the trend analysis and predictions. We have made use of Tableau to develop the charts to compare both the databases' performances for multiple operations.

When we look at some of the high-level query characteristics of the two database systems, MySQL tends to have an upper hand in selecting a large chunk of records while MongoDB turned out to be much quicker at inserting or updating numerous records.

With the added features that MongoDB holds, such as flexibility, rapid scalability, user-friendliness, and the advantages to manage cloud data, MongoDB is seen as a winner. Whereas one could opt for a MySQL database if data security is the utmost priority.

8. CONCLUSIONS AND RECOMMENDATIONS

8.1 SUMMARY AND CONCLUSIONS

In summary, we have generated sample data using python to simulate the IoT nature. We have tested for the bulk insertion and bulk read operations, in addition, we performed row by row insert, update, and delete operations for both MySQL and MongoDB. The comparison study shows that MongoDB performed better compared to MySQL for bulk read, row by row insert, and update operations. For bulk insertion, both databases performed similarly, MySQL took 1000 milliseconds i.e., 1 second less than MongoDB. So, it is preferred to use MongoDB for IoT data. We wanted to touch base on all the concepts covered in the class as we have implemented various ways to consume data, be it a website or Tableau dashboard once it is published into the database.

8.2 FUTURE SCOPE

In the future, we would like to use several other NoSQL databases such as Cassandra, Neo4J, HBase, etc., for the comparison analysis. We would also like to test the algorithm in a realistic environment where data is generated from IoT devices. We would also like to host a website, one can upload the dataset and select the databases for the comparison analysis, it would automatically calculate the performance of various parameters and display the comparison analysis.

9. BIBLIOGRAPHY

1) Sharvari Rautmare, Dr. D. M. Bhalerao, “MySQL and NoSQL database comparison for IoT application,” March 2017, Available: https://ieeexplore.ieee.org/document/7887957
2) Nadia Ben Seghier, Okba Kazar, “Performance Benchmarking and Comparison of NoSQL Databases: Redis vs MongoDB vs Cassandra Using YCSB Tool,” September 2021, Available: https://ieeexplore.ieee.org/document/9585956
3) Yosra Hajjaji, Wadii Boulila, Imed Riadh Farah, Imed Romdhani, Amir Hussain, “Big data and IoT-based applications in smart environments: A systematic review,” November 2020, Available: https://www.researchgate.net/publication/346009509_Big_Data_and_IoT-based_Applications_in_Smart_Environments_A_Systematic_Review
4) Somphop Chanthakit, Phongsak Keeratiwintakorn, Choopan Rattanapoka, “An IoT System Design with Real-Time Stream Processing and Data Flow Integration,” December 2019, Available: https://ieeexplore.ieee.org/document/8999968
5) Roman Čerešňák, Michal Kvet , “Comparison of query performance in relational a non-relation databases,” July 2019, Available: https://www.sciencedirect.com/science/article/pii/S2352146519301887
6) Sarita Padhy, G Mayil Muthu Kumaran, “A Quantitative Performance Analysis between MongoDB and Oracle NoSQL,” February 2020, Available: https://ieeexplore.ieee.org/document/8991245
7) https://www.epa.gov/air-trends/particulate-matter-pm10-trends
8) https://www.epa.gov/air-trends/particulate-matter-pm25-trends
9) O3 concentration in air
10) NO concentration in air
11) https://www.euro.who.int/__data/assets/pdf_file/0020/123059/AQG2ndEd_5_5carbonmonoxide.PDF
12) https://www.cse.scu.edu/~m1wang/
13) https://www.planetwatch.io/
14) https://coinmarketcap.com/currencies/planetwatch/

15) https://www.codit.eu/blog/5-ways-to-better-manage-datetime-in-iot-solutions/?country_sel=be

16) https://yeahexp.com/data-type-for-latitude-and-longitude/

17) PM 2.5 concentration in air

18) https://www.kaggle.com/

19) https://www.planetwatch.io/white-paper/pdf/white-paper.pdf