# FlowDB

Monjoy Narayan Choudhury (IMT2020502)
Hardik Khandelwal (IMT2020509)
Tejas Sharma (IMT2020548)

# Problem Statement

Our objective is to allow the user who is familiar with SQL to run their queries on a distributed system like mongoDB without knowing how the query is being run internally.

In the themes given to us, this lies well under the category of Data Ingestion and Structured Data processing and Analytics.

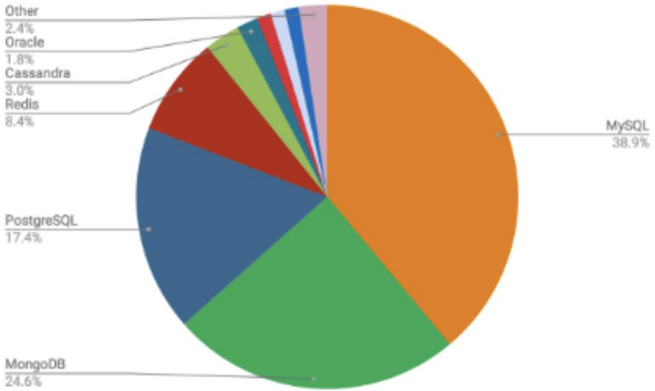
Our goal is to give a GUI-based data ingestion pipeline where the user can submit their default dataset in form of a csv and can define schema based on the fields present on the csv.

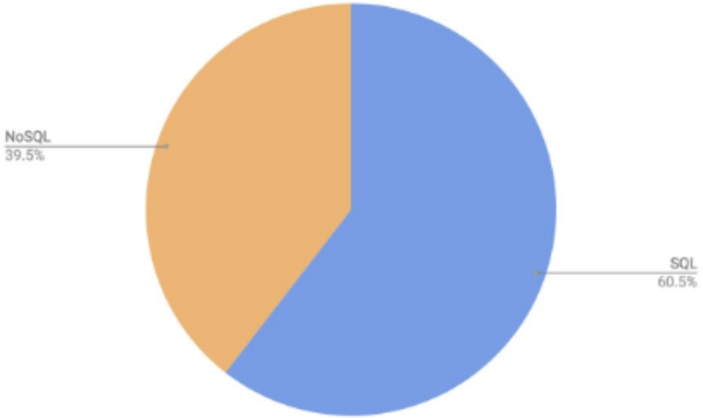
They can perform data cleaning steps like null value handling and enforce datatypes on various fields. Along with this, we also provide an interactive way to perform simple queries (Select and Aggregate queries)

In overall we want to abstract the MongoDB syntax away from users.

This allows users with some amount of SQL knowledge to perform basic queries without knowing anything about MongoDB

# Motivation




In a study shown by ScaleGrid in 2019, a majority of the user base is still using classical SQL-based databases with MySQL being majorly used.

It is also noticed that out of the 39.5 % share which NoSQL has, about 63% of it is filled with MongoDB users.

FlowDB aims to provide  interactive GUI-based SQL approach with MongoDB running under the hood giving the advantage of both SQL and NoSQL so that it can cater to users who want to leverage the technical advantages which NoSQL brings but couldn't do so due to the lack of them having the underlying concepts of operating so.

Also we wanted to avoid introducing another "Language" as we are pretty proficient in remembering syntax :)

# Implementation Aspects

- Data ingestion - Involves creating a baseline collection which used to insert values into a user specified schema.
- NULL and duplicate checks - Involves running these constraints on the user specified schema.
- Query processing - Involves writing appropriate queries in mongoDB for the SQL query specified by the user.

# Approach

Our application is built on the MERN stack comprising MongoDB as our datastore, React as the frontend, and Express and Node as the backend to handle server response and query the database.

Using React as the frontend enabled us to structure our pages in components, which helped in reusability. While the reason for using MongoDB is due to the following:

1. Schema flexibility
2. Scalability
3. Integration with Node.js:
4. JSON format

# Challenges We Faced

- Enforcing Data types for schema - We chose to typecast the values being inserted from the baseline collection at the time of creating the user specified schema.
- Designing queries - We limited the choice of queries to simple aggregation and select queries which can be extended in the future.
- Testing - Due to lack of time we haven’t extensively unit tested our components

# Performance Aspect

We evaluated our system made on the following dataset: Paris House Pricing.

It consists of about 1,00,000 rows.

To benchmark we sample multiple sizes. Our sample sizes consist of small (100 rows), medium (1000 rows), large(10,000 rows) and full which consist of the entire dataset.

To introduce null values we randomly take 3 rows and 3 columns and replace its original value with Null

We ran 2 benchmark tests - one on ingestion and the other on query processing.

We run an ingestion query for 5 self-made columns in our own schema. We try the different null replacement strategies i.e. do nothing, drop the row, replace with mean, and computed the time.

For the next set we perform queries on a single column for various sizes of sample datasets as performed above. The queries consist of the following: Finding houses with >=100 square meters area, Finding the minimum-sized house, Finding the average of the sizes., Finding all houses which have size > than the 75th percentile

The time is computed in milliseconds from when the submit button is clicked and a request is sent till the moment the response from the server with the status is received

| Dataset Size | Normal ingestion | Null removed | Null replace with mean |
|---|---|---|---|
| Small (100) | 111.2 | 156.2 | 176.2 |
| Medium (1000) | 253 | 255 | 259 |
| Large (10,000) | 748 | 750 | 776 |
| Complete (1,00,000) | 1908.8 | 1908.4 | 1914.8 |

**Table 1. Ingestion time in milliseconds calculated from when the request is sent from the frontend to the time the frontend receives the response from the back-end**

| Dataset Size | Select Query with >= | MIN | AVERAGE | >75% |
|---|---|---|---|---|
| Small (100) | 44 | 40.70000002 | 55 | 51.40000001 |
| Medium (1000) | 62.5 | 89.299 | 97 | 215.2 |
| Large (10,000) | 69.90000001 | 89.59999999 | 97.11 | 101.2 |
| Complete (1,00,000) | 71.222 | 93.73 | 98 | 140 |

**Table 2. Query time in milliseconds calculated from when the request is sent from the frontend to the time the frontend receives the response from the back-end**

# Demo

# Conclusion and lessons learnt

It is indeed possible to have a best of both worlds and create a system which allows someone to perform data ingestion.

Our approach is “dataset independent” so its highly generalisable for any csv file. This allows our application to be used in many fields: especially in fields where technology literacy may not be that high (for eg: data entry in finance, data entry professionals in medical science)

However, such system can never replace the traditional scripting methods

Designing an interactive system for complicated queries will be tough to represent as well as implement especially in case of multiple nesting and group queries.

**Further Direction**

Add more datatypes for adding constraints

Add the notion of references (foreign keys) and changing ID (primary key)

Come up with a better output UI

Implement joins

Improve interactivity: Allow user to drag drop logical blocks for performing queries similar to MIT scratch . This would truly bring the “flow” in FlowDB

# Thank You