**CALIFORNIA STATE UNIVERSITY, SACRAMENTO (CSUS)**

**COLLEGE OF ENGINEERING AND COMPUTER SCIENCE**

**CSC 244-01: Database System Design**

**SPRING 2021**

**COURSE PROJECT**

# MongoDB

Submitted by:

Harshitha Onkar (Sac State Id: 220262706)

## Abstract

The amount of Big Data generated has been on the rise in the past few years. Most of the Big Data that is generated is unstructured. To store, process and scale large amounts of unstructured data, a good NoSQL database is required. NoSQL databases allows us to store data without the rigidity of traditional schema-based relational databases. Based on data models, there are four major types of NoSQL databases: document-store, wide-column, key-value, and graph-based databases. This study focuses on MongoDB, a document-based NoSQL database which stores data in flexible, JSON-like format. The aim of this project is to use MongoDB as the database for developing a real-world application. The project focuses on understanding the data model of MongoDB, storing data, querying the database, and using Tableau as presentation layer to derive valuable insights from the data.

# Table of Contents

## Introduction

MongoDB is an open source, document-based, distributed NoSQL database. It was developed by 10gen company (currently known as MongoDB Inc.) and written in C++ [1]. Documents and Collections form the core of MongoDB data model design. The database and data model are built to support high performance, availability, and scalability [2]. An important feature of MongoDB is its support for flexible schema, that is two documents which belong to same collection can have different fields and JSON structure. This kind of flexibility allows NoSQL databases like MongoDB to store and manage unstructured data. Features like JSON data format, flexible schema, no complex joins, sharding, replication and high availability makes it a popular database in several application areas like Content Management, Customer Profile Personalization, Mobile and Social Infrastructure, Big Data and Internet of Things (IoT) [3].

The objective of this project is to explore the data model, architecture and query language of MongoDB and use it as the database to store manage and query real world data. Tableau is used as the visualization tool and presentation layer to gain valuable insights from the data.

## Project Design and Dataset

There are two workflows connecting to the database as shown in Fig.1

- For Querying and Loading database – Shell and MongoDB Compass is used
- For Data Visualization - Tableau is connected to database through ODBC Driver and BI connector.

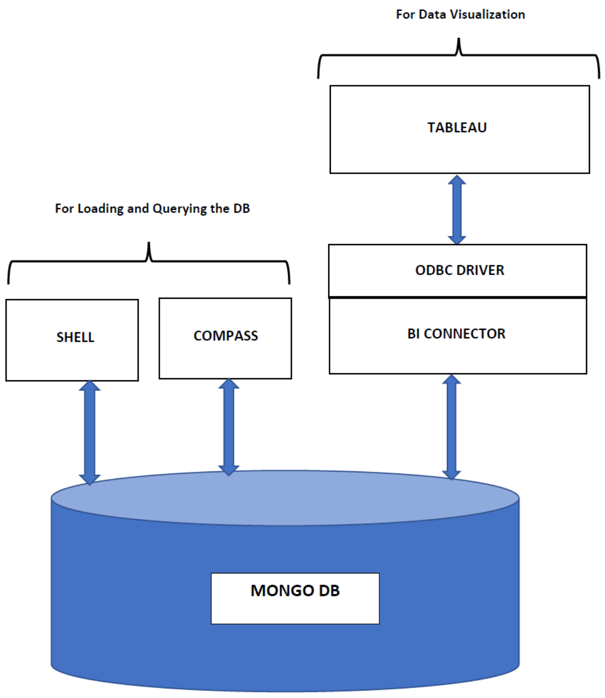


Fig 1. Project Design Flow

### Tools and Framework

MongoDB Server (Community), MongoDB Shell, Compass (GUI), Tableau, BI connector

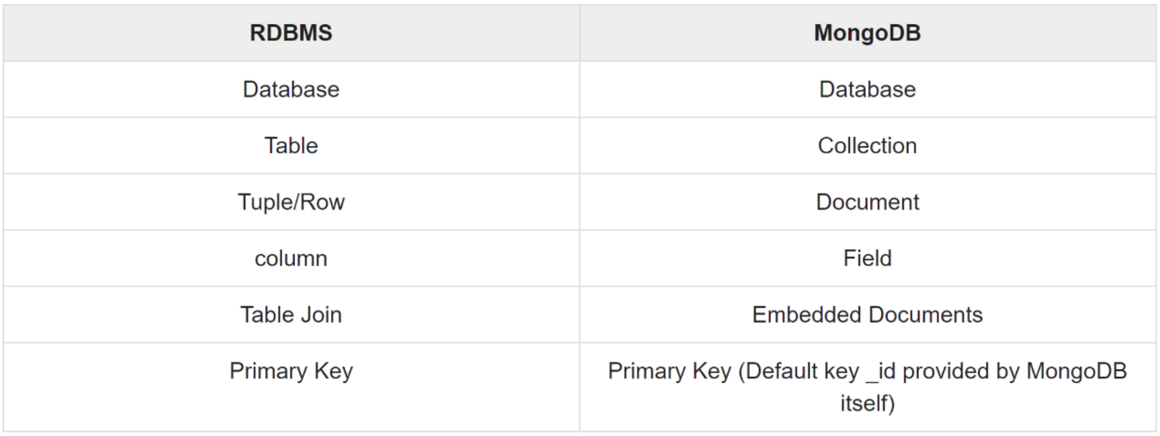
### Dataset

The dataset used for this project is Stack Overflow developer survey data. It consists of survey data of 80,000 developer across 180 countries regarding developer experience such as career satisfaction, job search to education and opinions on open-source software [4].

Link: https://insights.stackoverflow.com/survey

## Data Model

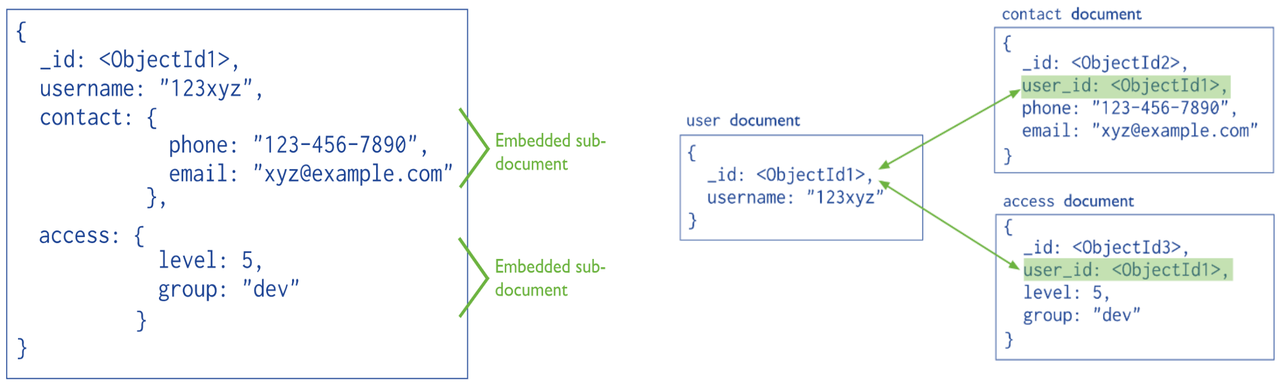
Documents form the core of MongoDB data model. A group of similar documents form a collection, and the information of single entity is stored as a single document in MongoDB. Collection is equivalent to a table and document to a record in Relational Database (RDBMS) terms. Fig.1 shows the mapping terminologies between MongoDB and RDBMS.



| RDBMS | MongoDB |
|---|---|
| Database | Database |
| Table | Collection |
| Tuple/Row | Document |
| column | Field |
| Table Join | Embedded Documents |
| Primary Key | Primary Key (Default key _id provided by MongoDB itself) |

Fig.2: Mapping of terminologies of RDBMS and MongoDB [5]

Unlike traditional SQL databases, where we must specify a table's schema before inserting data, MongoDB allows its collections to have flexible schema. That is: documents can have different structure. In MongoDB, the relationships (one-to-one, one-to-many, many-to-many) between tables is represented as either embedded document or referenced documents. Embedded documents allow denormalized information to be stored as part of the parent document whereas referenced documents store relationships as separate documents by including links or references from one document to another as seen in Fig.2.



```
{
  _id: <ObjectId1>,
  username: "123xyz",
  contact: {
              phone: "123-456-7890",
              email: "xyz@example.com"
            },
  access: {
             level: 5,
             group: "dev"
           }
}
```

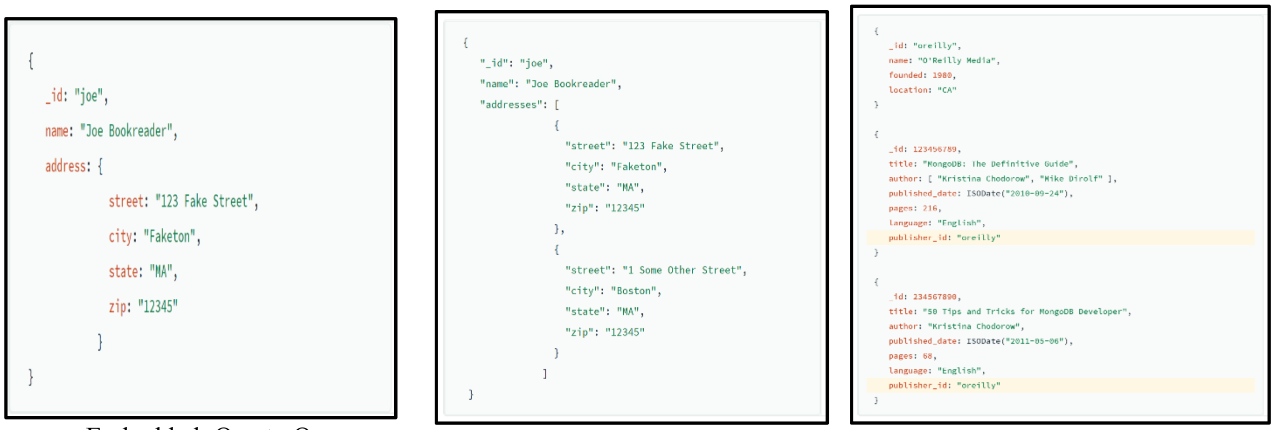
```
{
  _id: <ObjectId1>,
  username: "123xyz"
}
```

```
{
  _id: <ObjectId2>,
  user_id: <ObjectId1>,
  phone: "123-456-7890",
  email: "xyz@example.com"
}
```

```
{
  _id: <ObjectId3>,
  user_id: <ObjectId1>,
  level: 5,
  group: "dev"
}
```

Fig.3: Embedded and Reference documents

The choice of embedding or reference documents depends on how frequently data stored in them are accessed and performance requirements for the read queries. Usually, embedding is the default choice in most cases. Reference documents are generally used to represent complex many-to-many relationships, to avoid data duplication and when the referenced document data is occasionally required compared to the main document. Operations such as $lookup and $graphLookup are used to join reference documents [6].



```
{
  _id: "joe",
  name: "Joe Bookreader",
  address: {
             street: "123 Fake Street",
             city: "Faketon",
             state: "MA",
             zip: "12345"
           }
}
```

Embedded: One to One

```
{
   "_id": "joe",
   "name": "Joe Bookreader",
   "addresses": [
                  {
                    "street": "123 Fake Street",
                    "city": "Faketon",
                    "state": "MA",
                    "zip": "12345"
                  },
                  {
                    "street": "1 Some Other Street",
                    "city": "Boston",
                    "state": "MA",
                    "zip": "12345"
                  }
                ]
}
```

Embedded: One to Many

```
{
   _id: "oreilly",
   name: "O'Reilly Media",
   founded: 1980,
   location: "CA"
}

{
   _id: 123456789,
   title: "MongoDB: The Definitive Guide",
   author: [ "Kristina Chodorow", "Mike Dirolf" ],
   published_date: ISODate("2010-09-24"),
   pages: 216,
   language: "English",
   publisher_id: "oreilly"
}

{
   _id: 234567890,
   title: "50 Tips and Tricks for MongoDB Developer",
   author: "Kristina Chodorow",
   published_date: ISODate("2011-05-06"),
   pages: 68,
   language: "English",
   publisher_id: "oreilly"
}
```

Reference: One to Many

Fig.4: Embedded and Reference documents

## Query Language

MongoDB has two types of query language:

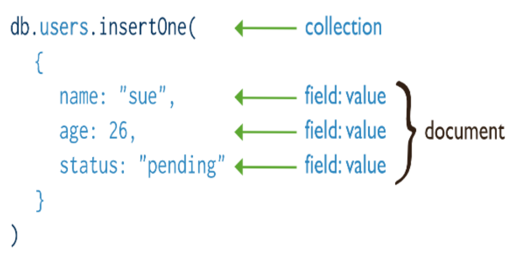
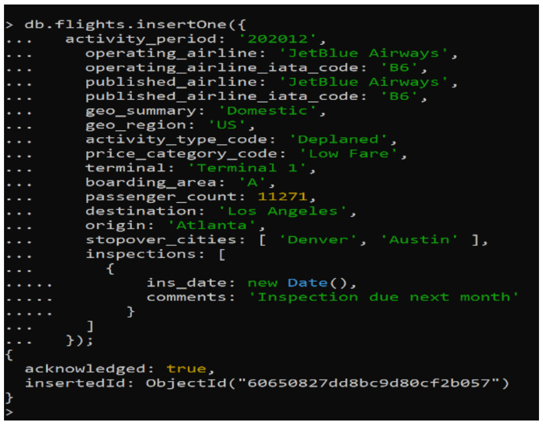
- MongoDB Query Language (MQL) - For basic CRUD operation on single collection documents [7].
- Aggregation Pipeline - For complex aggregations on documents that belong to multiple collections [8].

### MongoDB Query Language (MQL)

MQL is based on JSON format. Due to its imperative nature, it is easier to understand and build complex MQL queries compared to SQL (which is declarative in nature). Basic CRUD operations can be performed using MQL. Following are the syntax and corresponding figures for basic operations:

#### CREATE

**Syntax:** `db.collection.insertOne(), db.collection.insertMany()`

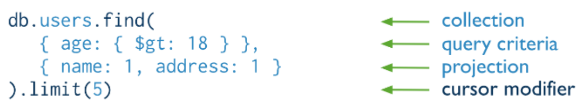
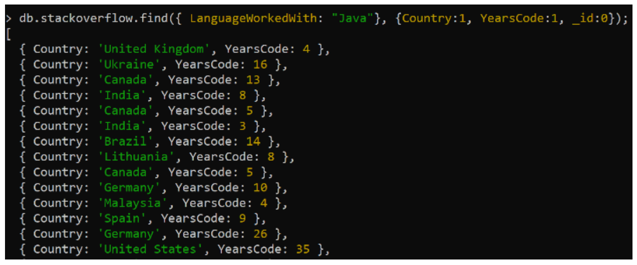
| Syntax | Example |
|---|---|
|  |  |

```
db.users.insertOne(
  {
    name: "sue",
    age: 26,
    status: "pending"
  }
)
```

```
> db.flights.insertOne({
...    activity_period: '202012',
...      operating_airline: 'JetBlue Airways',
...      operating_airline_iata_code: 'B6',
...      published_airline: 'JetBlue Airways',
...      published_airline_iata_code: 'B6',
...      geo_summary: 'Domestic',
...      geo_region: 'US',
...      activity_type_code: 'Deplaned',
...      price_category_code: 'Low Fare',
...      terminal: 'Terminal 1',
...      boarding_area: 'A',
...      passenger_count: 11271,
...      destination: 'Los Angeles',
...      origin: 'Atlanta',
...      stopover_cities: [ 'Denver', 'Austin' ],
...      inspections: [
...        {
.....          ins_date: new Date(),
.....          comments: 'Inspection due next month'
.....        }
...      ]
...    });
{
  acknowledged: true,
  insertedId: ObjectId("60650827dd8bc9d80cf2b057")
}
>
```

#### READ

**Syntax:** `db.<collection_name>.find({<filter>}, {<select>}),`
`db.<collection_name>.find().count()`

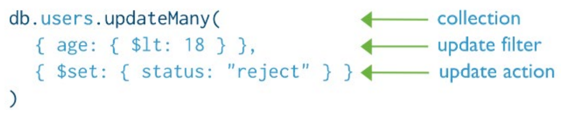
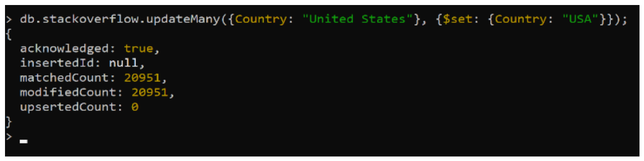
| Syntax | Example |
|---|---|
|  |  |

```
db.users.find(
   { age: { $gt: 18 } },
   { name: 1, address: 1 }
).limit(5)
```

```
> db.stackoverflow.find({ LanguageWorkedWith: "Java"}, {Country:1, YearsCode:1, _id:0});
[
  { Country: 'United Kingdom', YearsCode: 4 },
  { Country: 'Ukraine', YearsCode: 16 },
  { Country: 'Canada', YearsCode: 13 },
  { Country: 'India', YearsCode: 8 },
  { Country: 'Canada', YearsCode: 5 },
  { Country: 'India', YearsCode: 3 },
  { Country: 'Brazil', YearsCode: 14 },
  { Country: 'Lithuania', YearsCode: 8 },
  { Country: 'Canada', YearsCode: 5 },
  { Country: 'Germany', YearsCode: 10 },
  { Country: 'Malaysia', YearsCode: 4 },
  { Country: 'Spain', YearsCode: 9 },
  { Country: 'Germany', YearsCode: 26 },
  { Country: 'United States', YearsCode: 35 },
```

## UPDATE

**Syntax:** `db.<collection_name>.updateOne(filter, update, options)`, `db.<collection_name>.updateMany(filter, update, options)`

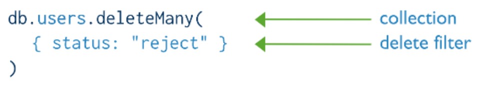
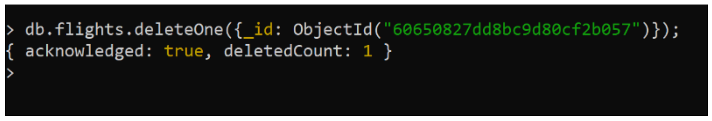
| Syntax | Example |
|---|---|
|  |  |

```
db.users.updateMany(
   { age: { $lt: 18 } },
   { $set: { status: "reject" } }
)
```

```
> db.stackoverflow.updateMany({Country: "United States"}, {$set: {Country: "USA"}});
{
  acknowledged: true,
  insertedId: null,
  matchedCount: 20951,
  modifiedCount: 20951,
  upsertedCount: 0
}
> _
```

## DELETE

**Syntax:** `db.collection.deleteOne({filter})`, `db.collection.deleteMany({filter})`

| Syntax | Example |
|---|---|
|  |  |

```
db.users.deleteMany(
   { status: "reject" }
)
```

```
> db.flights.deleteOne({_id: ObjectId("60650827dd8bc9d80cf2b057")});
{ acknowledged: true, deletedCount: 1 }
>
```

## Aggregation Pipeline

MongoDB's aggregation pipeline works like a data processing pipeline where documents enter a multi-stage pipeline, and the document is transformed at each stage to obtain the final aggregated result. The Aggregation Framework provides effective data grouping and filtering operations hence they are the preferred method for data aggregations in MongoDB [8].

Fig.4 shows an example of MongoDB aggregation pipeline that consists of three stages: $match - which filters the input, $group - that aggregates or groups all the documents based on group key, $sort – used to sort the output in ascending or descending order [9].

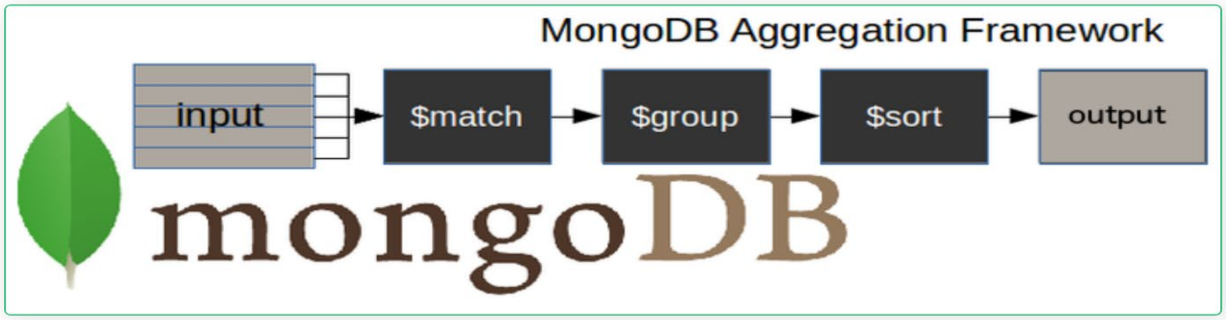
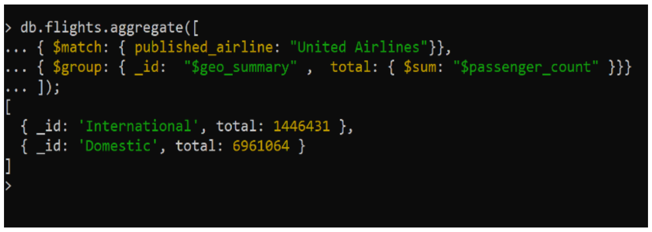


Fig.5: MongoDB Aggregation Pipeline

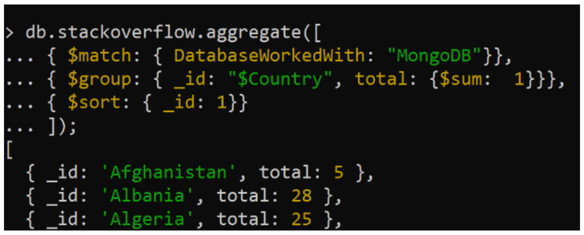
The input to aggregation pipeline can be documents from single or multiple collections. The output of one stage becomes the input for the next successive stage. The pipeline thus performs successive transformations on the input to obtain the final output [9].

Syntax for Aggregation pipeline*: db.collectionName.aggregate(pipeline, options)*

*pipeline = [ { $match : { … }, { $group : { … }, { $sort : { … }, …]*



```
> db.flights.aggregate([
... { $match: { published_airline: "United Airlines"}},
... { $group: { _id:  "$geo_summary" ,  total: { $sum: "$passenger_count" }}}
... ]);
[
  { _id: 'International', total: 1446431 },
  { _id: 'Domestic', total: 6961064 }
]
>
```



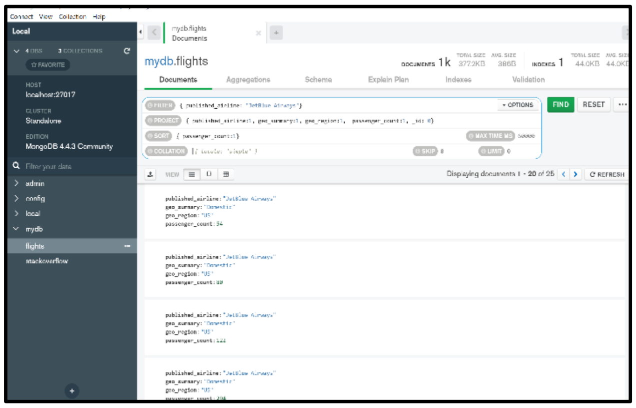
```
> db.stackoverflow.aggregate([
... { $match: { DatabaseWorkedWith: "MongoDB"}},
... { $group: { _id: "$Country", total: {$sum:  1}}},
... { $sort: { _id: 1}}
... ]);
[
  { _id: 'Afghanistan', total: 5 },
  { _id: 'Albania', total: 28 },
  { _id: 'Algeria', total: 25 },
```

Fig.6: Aggregation Examples

## MongoDB Compass (GUI)

MongoDB Compass is a graphical user interface that allows users to explore, analyze, and interact with the data stored in a MongoDB [10]. It is an open-source software that can be downloaded from MongoDB official website. Some of the functionality provide by Compass are:

- Allows users to quickly visualize and explore schema to understand the frequency, types, and ranges of fields in your data set.
- Easier CRUD functionality using interactive visual editors
- Validating data using JSON schema validators
- Intuitive Aggregation pipelines that can be saved for future use
- Manage and analyze the utilization of indexes


Connect View Collection Help
Local
4 DBS 3 COLLECTIONS
FAVORITE
HOST localhost:27017
CLUSTER Standalone
EDITION MongoDB 4.4.3 Community
Filter your data
admin
config
local
mydb
flights
stackoverflow
mydb.flights Documents
mydb.flights
DOCUMENTS 1k TOTAL SIZE 377.2KB AVG. SIZE 386B
INDEXES 1 TOTAL SIZE 44.0KB AVG. SIZE 44.0KB
Documents Aggregations Schema Explain Plan Indexes Validation
OPTIONS FIND RESET
VIEW
Displaying documents 1 - 20 of 25
REFRESH

```
FILTER     { published_airline: "JetBlue Airways"}
PROJECT    { published_airline:1, geo_summary:1, geo_region:1,  passenger_count:1, _id: 0}
SORT       { passenger_count:1}                              MAX TIME MS 5000
COLLATION  { locale: 'simple' }                  SKIP 0           LIMIT 0
```

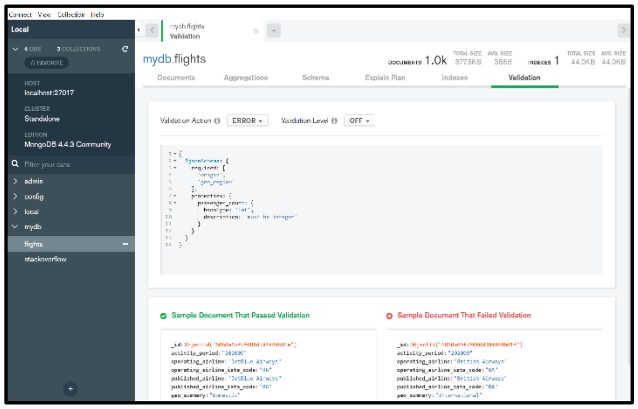
```
published_airline:"JetBlue Airways"
geo_summary:"Domestic"
geo_region:"US"
passenger_count:54

published_airline:"JetBlue Airways"
geo_summary:"Domestic"
geo_region:"US"
passenger_count:88

published_airline:"JetBlue Airways"
geo_summary:"Domestic"
geo_region:"US"
passenger_count:122

published_airline:"JetBlue Airways"
geo_summary:"Domestic"
geo_region:"US"
passenger_count:204
```

Fig.7: MongoDB Document Query Screen


Connect View Collection Help
Local
4 DBS 3 COLLECTIONS
FAVORITE
HOST localhost:27017
CLUSTER Standalone
EDITION MongoDB 4.4.3 Community
Filter your data
admin
config
local
mydb
flights
stackoverflow
mydb.flights Validation
mydb.flights
DOCUMENTS 1.0k TOTAL SIZE 377.5KB AVG. SIZE 386B
INDEXES 1 TOTAL SIZE 44.0KB AVG. SIZE 44.0KB
Documents Aggregations Schema Explain Plan Indexes Validation
Sample Document That Passed Validation
Sample Document That Failed Validation

Validation Action: ERROR
Validation Level: OFF

```
{
  $jsonSchema: {
    required: [
      'origin',
      'geo_region'
    ],
    properties: {
      passenger_count: {
        bsonType: 'int',
        description: 'must be integer'
      }
    }
  }
}
```

```
_id: ObjectId("[illegible]")
activity_period: "202009"
operating_airline: "JetBlue Airways"
operating_airline_iata_code: "B6"
published_airline: "JetBlue Airways"
published_airline_iata_code: "B6"
geo_summary: "Domestic"
```

```
_id: ObjectId("[illegible]")
activity_period: "202009"
operating_airline: "British Airways"
operating_airline_iata_code: "BA"
published_airline: "British Airways"
published_airline_iata_code: "BA"
geo_summary: "International"
```

Fig.8: MongoDB Schema Validator Screen

## Tableau as Presentation layer

Tableau is a visual analytics platform that helps to understand and gain valuable insights from the data through charts and interactive dashboards [11]. It can connect to a wide variety of data sources such as .csv files, MS Access, SQL databases and NoSQL databases.

In this project, Tableau is used as the presentation layer to explore, analyze, and derive valuable insights from the StackOverflow Developer data using dynamic charts and other visualization techniques. We connect Tableau to MongoDB via ODBC Driver and MongoDB BI connector. Since the underlying database is schema-less NoSQL DB, an ODBC driver is required to map the schema-less DB data into tabular form and vice-versa [12] [13].

Following are screenshots of charts and dashboards created using Tableau Desktop:

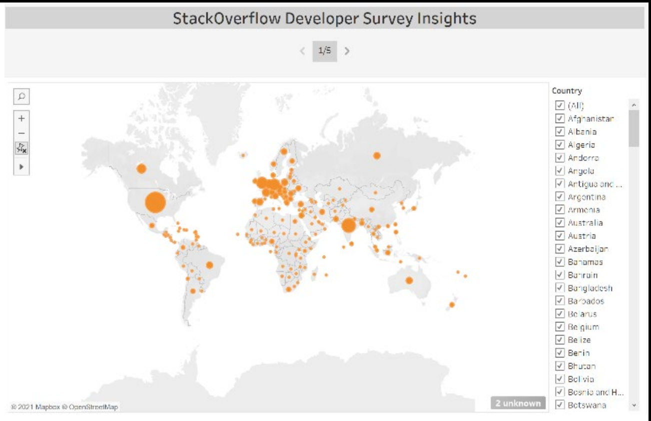


Fig.9: Country-Wise Survey Distribution

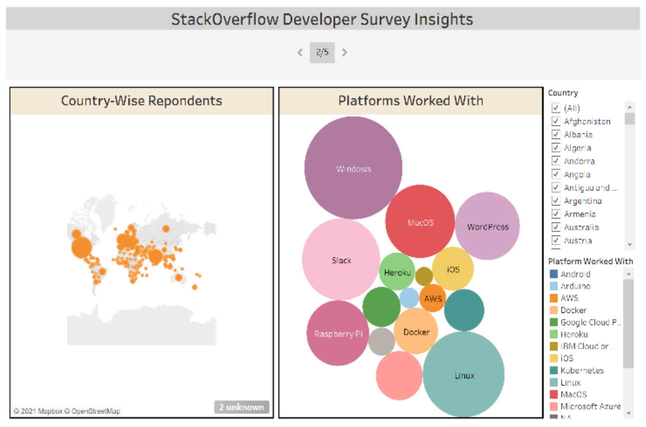


Fig.10: Platform Popularity

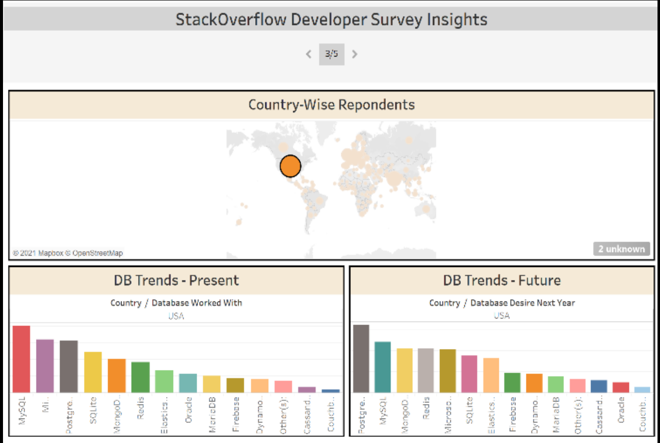


Fig.11: DB Trends Present and Future

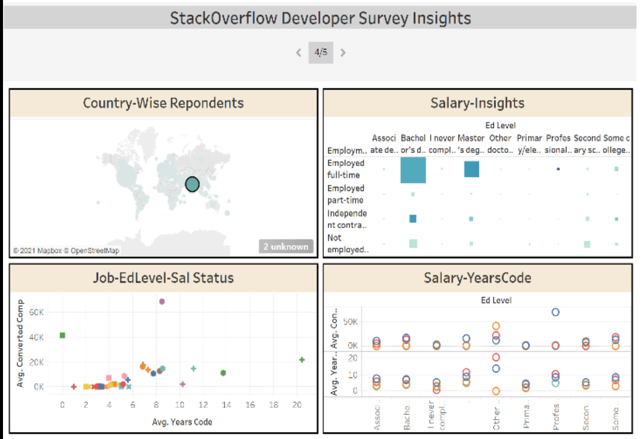


Fig.12: Salary-Education-Employment Insights

## Applications of MongoDB

Some of the real-world application areas of MongoDB are Content Management, Product Cataloging, Customer Analytics & Personalization and Ecommerce. With features like Distributed database, flexible schema, scalability, sharding and cloud services, the popularity and application areas of MongoDB are growing exponentially. Following are few real-world MongoDB use cases:

- Forbes and FACEIT for Content Management
- Aadhar, Shutterfly and MetLife for Mainframe Offloading
- Bosch for processing data from Internet of Things
- Expedia and eBay for Customer Analytics and Personalization

## Project Reflection and Takeaways

This project gave me an opportunity to learn two of the most powerful and upcoming technologies in Data Analytics – MongoDB and Tableau. Most of the DB experience that I had previously was in RDBMS, so learning about different types of NoSQL databases and MongoDB was refreshing. It helped me understand the use cases and application areas of NoSQL databases, the data models and query language.

It was interesting to learn about one of the most widely used BI tools – Tableau and how it can be used to create dynamic dashboards and stories from the underlying data.

## Conclusion

With growing needs to store, manage, and process Big Data, the need for technologies and tools like MongoDB and Tableau cannot be understated. The flexible schema, scalability and distributed nature of MongoDB coupled with a powerful visual analytics tool like Tableau helps to manage, query, gain important analytical and predictive information hidden within the Big Data.

This project was helpful to understand and get a quick start on how NoSQL databases like MongoDB are increasingly being used to store and manage structureless Big Data where more emphasis is given on query performance rather than adhering to rigid schema structures as in traditional RDBMS. It helped in exploring one of the most powerful Business Analytics and Intelligence tool – Tableau. I hope to build on the skills gained during this project and use it for future projects.

## References

[1] "Wikipedia: MongoDB," [Online]. Available: https://en.wikipedia.org/wiki/MongoDB.

[2] H. Krishnan, S. M. Elayidom and S. T, "MongoDB – a comparison with NoSQL databases," *International Journal of Scientific & Engineering Research,* vol. 7, no. 5, 2016.

[3] "MongoDB - Advantages," [Online]. Available: https://www.tutorialspoint.com/mongodb/mongodb_advantages.htm.

[4] "Stack Overflow Annual Developer Survey," [Online]. Available: https://insights.stackoverflow.com/survey.

[5] "MongoDB - Overview," [Online]. Available: https://www.tutorialspoint.com/mongodb/mongodb_overview.htm.

[6] "MongoDB: Data Model Design," [Online]. Available: https://docs.mongodb.com/manual/core/data-model-design/.

[7] "MongoDB CRUD operations," [Online]. Available: https://docs.mongodb.com/manual/crud/.

[8] "MongoDB: Aggregation," [Online]. Available: https://docs.mongodb.com/manual/aggregation/.

[9] "Aggregation, The Beginner's Guide to MongoDB," [Online]. Available: https://studio3t.com/knowledge-base/articles/mongodb-aggregation-framework/#aggregation-stages.

[10] S. Wickramasinghe, "MongoDB Compass: Using the Mongo GUI," [Online]. Available: https://www.bmc.com/blogs/mongodb-compass/.

[11] "Tableau," [Online]. Available: https://www.tableau.com/why-tableau/what-is-tableau.

[12] "MongoDB BI Connector," [Online]. Available: https://help.tableau.com/current/pro/desktop/en-us/examples_mongodb.htm.

[13] "TLS/SSL, Connect from Tableau Desktop without Authentication or TLS/SSL," [Online]. Available: https://docs.mongodb.com/bi-connector/current/connect/tableau-no-auth/#std-label-connect-with-tableau.