Marwadi University

01CE0412-Advanced Web Technology

Unit-4 MongoDB

Prof. Vaibhav K Matalia
Computer Engineering Department

Introduction of MongoDB

- It is NoSQL database.(We don’t need to write select,insert,update,delete query)
- MongoDB is a document database. It stores data in a type of JSON format called BSON(Binary JSON).
- A record in MongoDB is a document, which is a data structure composed of key value pairs similar to the structure of JSON objects.
- MongoDB is a document database and can be installed locally or hosted in the cloud.

Download MongoDB:-

Step 1:-Type download mongodb in google

Step 2:-Open mongodb.com/try/community

Step 3:-Click on mongodb community server tab and click on download button.

- Once installation is complete, MongoDB compass will come automatically.

Note:-Open C://ProgramFiles folder and verify mongodb folder is exist or not.bin and data are two important folder inside mongodb.

- MongoDB is a NoSQL database(NonSQL database).Oracle,MySQL,SQL Server all this are Structured Query Language database. It has a proper structure in which we have applied the query.
- The data stored in a collection.
- Collection don't have row and columns.
- Data stored in the form of object(In javascript we create an array that has multiple object)

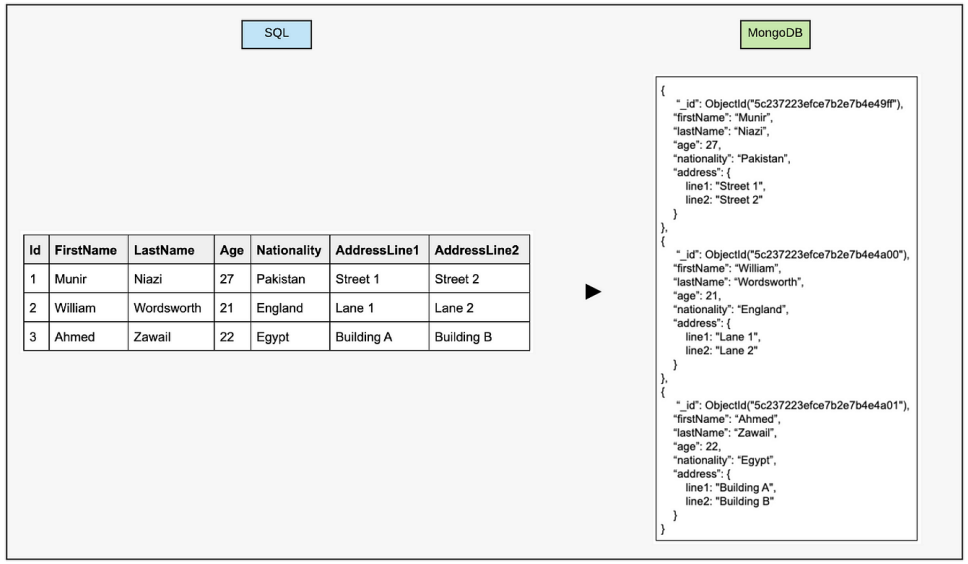
Download mongodb shell:-

- Open https://www.mongodb.com/try/download/shell
- Once installation is complete type mongosh in cmd.
- Type show dbs

MongoDB compass:-

- It is one GUI tool from which we can create database, create colletion,enter data into collection etc.
- It is similar to phpMyAdmin(Provide GUI from which we can access MySQL database).
- MongoDB provide 3 default database.

MongoDB vs SQL



- It is not necessary that all objects in mongodb have the same number keys.
- We can say mongodb is unstructured database, because number of keys will not same in all objects.

When to use MongoDB:-

- Assuming that there is a form, there are many optional fields in it. In such a case, if you use MySQL, memory will be wasted.

Note:-If you don’t want to use MongoDB compass you can use Robo 3T GUI tool.

Note:-When we click on connect button in mongodb compass so we will automatically connect to the local database.

MongoDB basic command:-

1.**show dbs**:-It will show all the database.

2.**use e-comm**:-Create new database.
Check database is visible in MongoDB compass or not.Type show dbs command in cmd.

3.**db.createCollection('products')**:-Create new collection(table)

4.**show collections**:-Display all collections

5.**db.products.drop()**:-Delete particular collection

6.**db.dropDatabase()**:-Delete database. It will delete current DB.

Note:-MongoDB is one database program and MongoDB is one GUI tool from which we can access MongoDB database.

CRUD Operation:-

Insert Operation:-

- We will insert data in collection using two methods.
  - Insert data using command line
  - Insert data using GUI

Command line:-

```
db.products.insertOne({name:"M52 black 8GB 128GB",company:"samsung",price:26000,category:"mobile"})
db.products.insertMany([{name:"Note10pro 6GB blue 64GB",company:"MI",price:11000,category:"mobile"},{name:"A33 carbon black 6GB 64GB",company:"samsung",price:15000,category:"mobile"}])
db.products.find()
db.products.find({company:"samsung"})
```

Note:-No need to give id when inserting data mongodb will take the id automatically

insert from UI:-

- Click on add data, click on insert document. When entering data from the user interface, the key will be in the string.

Update Operation:-

Command line:-

- To update an existing document we can use the updateOne() or updateMany() methods.
- The first parameter is a query object to define which document or documents should be updated.
- The second parameter is an object defining the updated data.

updateOne():-It will update the first document that is found matching the provided query.
db.products.updateOne({company:"samsung"},{$set:{category:"samsung mobile"}})

updateMany():-It will update all documents that match the provided query.
db.products.updateMany({category:"samsung mobile"},{$set:{category:"mobile"}})

Insert if not found
If you would like to insert the document if it is not found, you can use the upsert option.
db.products.updateOne({name:"Nokia 5210"},{$set:{price:5500}},{upsert:true})

Delete Operation:-

- We can delete documents by using the methods deleteOne() or deleteMany().
- These methods accept a query object. The matching documents will be deleted.

deleteOne():-It will delete the first document that matches the query provided.

db.products.deleteOne({name:"Nokia 5210"})

deleteMany():-

db.products.deleteMany({category:"samsung"})

Connect Node with MongoDB:-

- First Install MongoDB npm package using below command:
  npm install mongodb
- This package will contain the code that can connect mongodb with the node.

Example:-

```
const {MongoClient}=require('mongodb');//Modern javascript syntax
//const MongoClient=require('mongodb').MongoClient; Same as above syntax
const url="mongodb://localhost:27017";
let client=new MongoClient(url);
async function getData()
{
    let result=await client.connect();
    let db=result.db('e-comm');
```


```
    let collection=db.collection('products');
    console.log(await collection.find().toArray());
}
getData();
```

Mongoose

- It is a package that help us to connect nodejs and mongodb.
- Mongodb has some limitations that are solved in Mongoose.
- Mongoose package was created by working on Mongodb.
- Mongoose provide some advance features as compare to mongodb.

Example:-products collection(only 5 field)

- In mongoose we can define field type(Number,String).

Install Mongoose:-

npm install mongoose

Schemas:-Validate the fields in the database.

Model:-Connect nodejs and mongodb.

Difference between schema and model

- We will define the fields of the database within the schema on which validation will be applied. The model will connect mongo and node using this schema.

Note:-Mongoose has two main parts first is schemas second is model.

Example:-Prevent Extra field

```
const mongose=require('mongoose');
const main=async()=>
{
    await mongose.connect("mongodb://127.0.0.1:27017/e-comm");
    const ProductSchema=new mongose.Schema({
        name:String,
        price:Number,
        company:String
});
```

```
const ProductModel=mongose.model('products',ProductSchema);
 let data=new ProductModel({
        name:"A53 8GB 128GB Black",
        price:33500,
        company:"samsung"
    });
    let result=await data.save();
    console.log(result);
}
main()
```

CRUD with mongoose:-

```
const mongose=require('mongoose');
mongose.connect("mongodb://127.0.0.1:27017/e-comm");
const ProductSchema=new mongose.Schema({
    name:String,
    price:Number,
    company:String,
    color:String
});
```

```
const insert=async()=>
{
    const Product=mongose.model('products',ProductSchema);
    let data=new Product({
        name:"A53 8GB 128GB",
        price:33500,
        company:"samsung",
        color:"Black"
    });
    let result=await data.save();
    console.log(result);
}
//insert()
```

```
const update=async()=>
{
    const Product=mongose.model('products',ProductSchema);
    let data=await Product.updateOne({name:"A53 8GB 128GB"},{$set:{price:23500}});
    console.log(data);
}
//update();
const deleteData=async()=>
{
    const Product=mongose.model('products',ProductSchema);
    let data=await Product.deleteOne({name:"A53 8GB 128GB"});
    console.log(data);
}
deleteData();
```

```
const display=async()=>
{
    const Product=mongose.model('products',ProductSchema);
    let data=await Product.find();
    console.log(data);
}
//display();
```

POST API with mongoose:-

config.js

```
const mongose=require('mongoose');
mongose.connect("mongodb://127.0.0.1:27017/e-comm");
```

product.js

```
const mongose=require('mongoose');
let productSchema=new mongose.Schema({
    pname:String,
    price:Number,
    quantity:Number
});
module.exports=mongose.model('products',productSchema);
```

index.js:-

```
const product=require('./product');
require('./config');
const express=require('express');
const app=express();
app.use(express.json());
app.post('/',async(req,resp)=>
{
    let data=new product(req.body);
    let result=await data.save();
    resp.send(result);
}).listen(4200);
```

GET API:-

```
const product=require('./product');
require('./config');
const express=require('express');
const app=express();
app.use(express.json());
app.get('/',async(req,resp)=>
{
    let data=await product.find();
    resp.send(data);
}).listen(3001);
```

Marwadi University

Delete API:-

```
const product=require('./product');
require('./config');
const express=require('express');
const app=express();
app.use(express.json());
app.delete('/',async(req,resp)=>
{
    let result=await product.deleteOne({pname:req.body.pname});
    resp.send(result);
}).listen(3001);
```

```
const product=require('./product');
require('./config');
const express=require('express');
const app=express();
app.use(express.json());
app.delete('/:id',async(req,resp)=>
{
    let result=await product.deleteOne({_id:req.params.id});
    resp.send(result);
}).listen(3001);
```

Update API:-

```
const express=require('express');
require('./config');
const product=require('./product');
const app=express();
app.use(express.json());
app.put('/update/:_id',async(req,resp)=>
{
    let data=await product.updateOne(req.params,{$set:req.body})
    resp.send(data);
})
app.listen(3001);
```

Indexing

Evaluating Query performance in MongoDB:-

- We can measure query performance by 2 methods
  - Command line interface
  - Mongodb compass

Command line interface:-

- Create new database and collection and import large dataset. If you want to check query performance first you need large dataset(collection having multiple objects)
- Create jobapp database and indexdemo collection and import json file. Type db.users.find({name:'Eugene_1'})
- In MongoDB, there is one method called explain() from which we can measure query performance. This method provide some statistics.

- Type db.indexdemo.explain().find({name:'Eugene_1’}).This query will print some stats in console.
- There are multiple modes from which we can execute explain() function. Here we are using ‘executionStats’ mode which will provide some query information.
- Type db.indexdemo.explain('executionStats').find({name:'Eugene_1’})
  This query provide one property called executionStats.
- All the query has its own executionStage.If you execute above query collection scan stage is used to fetch the result.
- totalkeysExamined is equal to zero means indexing is not used in query.

Mongodb compass:-

- Click on collection type {name:’Eugene_1’}.Click on find button. If you want to check query performance click on Explain button.

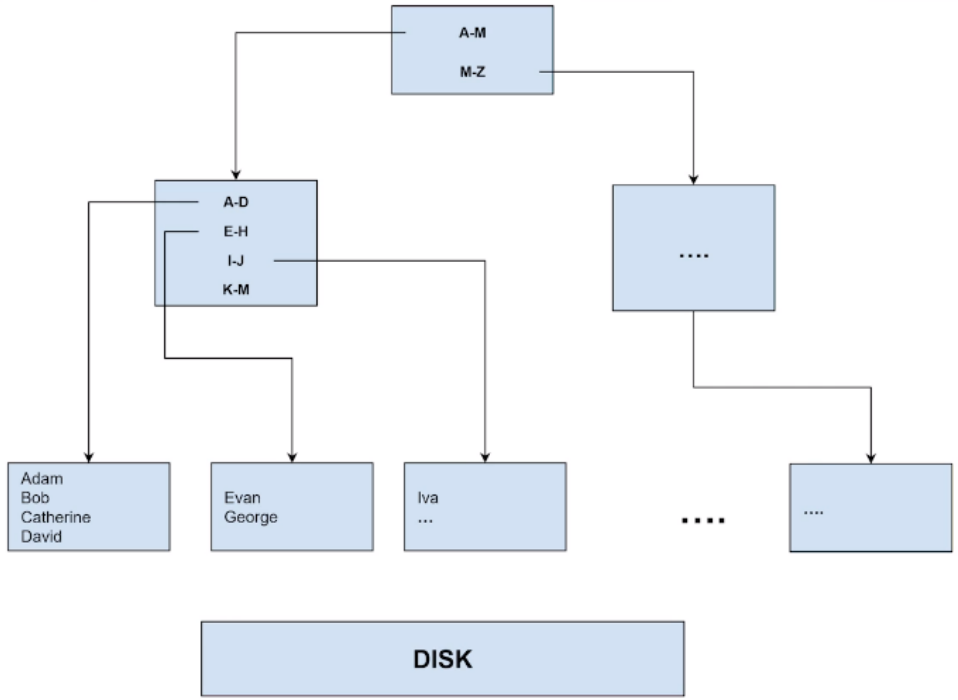
What is indexing?

- Real application database has thousands of records and data will be increasing day by day.
- When we use find() function to retrieve data whole collection will scan.It will takes some time and this is not efficient.
- To solve this problem indexing comes into picture.

Example:-Library of 10000 books

- If data is organized so we don’t need to scan all the documents to retrieve desired result.
- With indexing lookup will be fast and it is time saving process.
- Indexing organize the data so retrieval will be easy.
- Index store as a object. This object is used to find the desired result.

- Indexing does not modify order of data, but it creates 1 pointer that points to existing documents.
- MongoDB uses B-Tree(Balanced Tree) data structure to store index.



- We want to find a person whose name is David, So first MongoDB scan first block(root block),then scan left block again scan left block. Here mongodb find the pointer to David. This pointer points to actual object in collection.
- Here information is organized so we don't need to scan all the objects. Information is organized in the form of pointer.

Conclusion:-

- Indexes make accessing data easier and faster. It improves overall system performance.
- Without indexes to retrieve any information, the whole collection will be scanned.
- Mongodb uses a B-Tree data structure to store indexes.
- Indexes store data in an organized way for easy access.Here,actual data is not organized, but indexes are stored separately and it consist of points to actual data.

Default index:-

- Id field has default index and all collection has id field.
- We can check default index using Mongodb compass and through command line(db.indexdemo.getIndexes()).

Having index vs without index

- Open Mongodb compass and search {name:’Eugene_1’}.Search any object using id.
- Open command line interface and type

db.indexdemo.explain('executionStats').find({name:’Eugene_1’})

db.indexdemo.explain('executionStats').find({_id:ObjectId('63bcf75ac98093f126b4a99a')})

Creating index:-

Syntax:-db.collection.createIndex({keyName:1|-1},options)

db.indexdemo.createIndex({name:1})

db.indexdemo.getIndexes()

db.indexdemo.explain('executionStats').find({name:’Eugene_1’})

- We can also create an index using Mongodb compass.

Deleting indexes:-

db.indexdemo.dropIndex('name_1’)

- We can also delete an index using Mongodb compass.

Index options:-

Syntax:-db.collection.createIndex({keyName:1|-1},options)

- This parameter is used to change index name(default is fieldname_ascending/descending).With this parameter we can set primary key.
- Create new collection(users) and add below query db.users.createIndex({email:1},{name:'email_index',unique:true})
- Add below query in indexdemo collection db.indexdemo.createIndex({name:1},{unique:true})→Error(duplicate value is already stored)

Indexing Types:-

Single field index

Compound index

Multikey index

Compound Index:-

Syntax:-db.collection.createIndex({key-name:-1|1,key-name:-1|1})

- This will help in sorting. With this index performance of sorting query will improve.

db.indexdemo.createIndex({salary:1,city:1})

- To check above query open mongodb compass and type {salary:20000} and {city:1}

Aggregation

- Till now we are using find() function to retrieve the result.
  collection.find()

  collection.find({username:"vaibhav"})

  collection.find({age:{$gt:30}})

Requirement

1. How many user do we have in each city?
2. I want count of students who have studied in different University
3. I want to find maximum/minimum salary from collection
4. We want users whose age is greater than 30 city wise

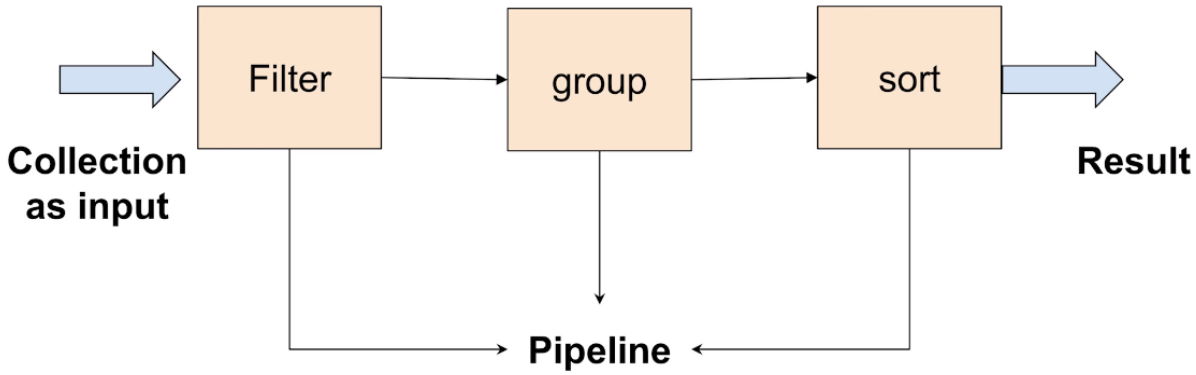
- To process this query, we will need to process multiple documents to get computed results. This process of getting computed result is known as aggregation.
- Aggregation means processing multiple documents in a collection to get computed results.
- Aggregation is very useful, when we have lots of data stored and we wish to process it.

What is aggregation framework?

- While find() has its own use-cases,MongoDB has an aggregation framework that allows developers to get aggregated results.
- It provides an aggregate() method using which you can get computed results.
- The aggregation framework breaks login into simple parts and processes each part one by one.
- This might resemble a waterfall model where the output of one stage is given as input to the next stage.

Marwadi University

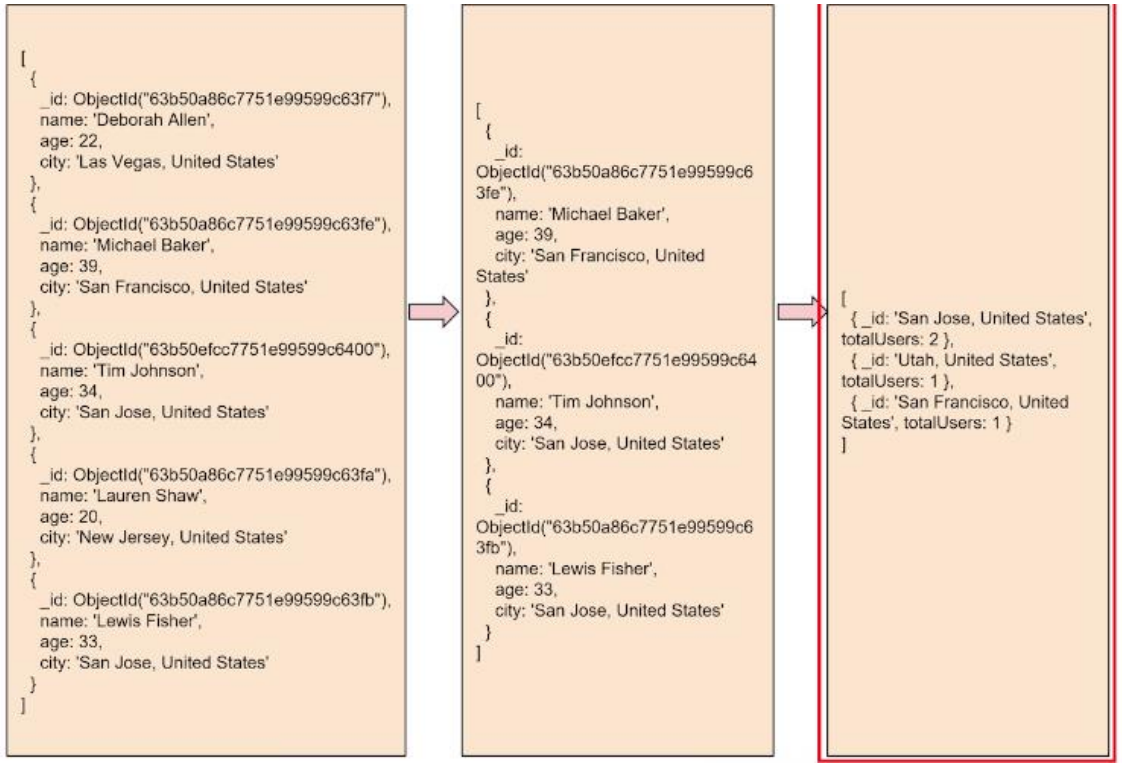
What is aggregation pipeline?



- To get computed result, In diagram as you can see multiple stages. Each stage is responsible for doing certain transformation on the data. This transformed data is passed into the next stage as an input.
- The output from the final stage is the actual computed value that the user requires.
- This entire series of stages that does transformation is known as aggregation pipeline.

- So you can see we have the collection here as an input, then we have multiple stages like filter. So we filter out some results, we do some grouping. We do some sorting and then we get the final result. And these stages are known as pipeline.

Count of users having age greater than 30



First query using aggregation framework

```
db.users.aggregate([{$match:{age:{$gt:30}}},{$group:{_id:"$city",totalUsers:{$sum:1}}}]
)
```

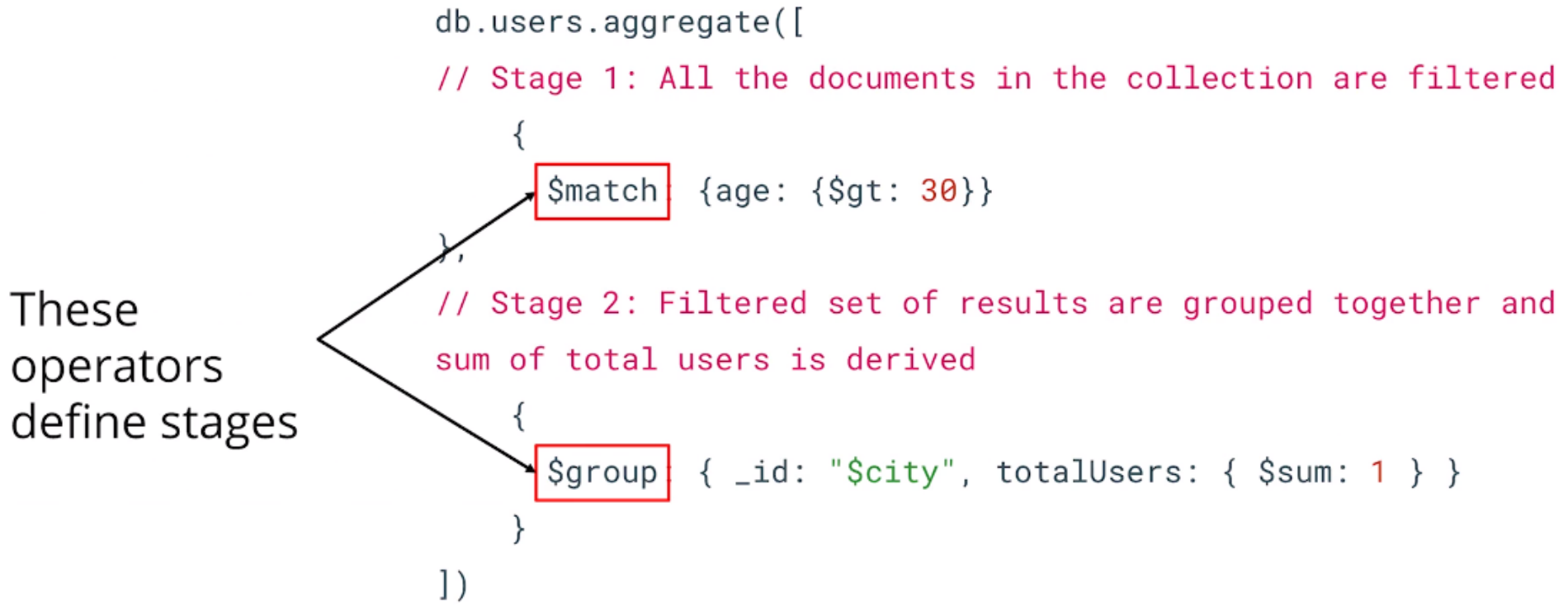
- Here we are using an array because there are two stages.
- id helps you define the key using which you are grouping.
- Above query has two stage

  $match:-used to filter data
  $group:-used to group data citywise

- Execute below query

```
db.users.aggregate([{$match:{age:{$gt:30}}}])
db.users.aggregate([{$group:{_id:"$city",totalUsers:{$sum:1}}}])
```

Operators in aggregation



```
db.users.aggregate([
// Stage 1: All the documents in the collection are filtered
    {
      $match: {age: {$gt: 30}}
},
// Stage 2: Filtered set of results are grouped together and
sum of total users is derived
    {
      $group: { _id: "$city", totalUsers: { $sum: 1 } }
    }
])
```

This is an accumulator that is used when data is grouped together

Marwadi University

List of Stage Operator:-

Operator	Usecase
$match	Filters documents that you might need for the next stage of processing
$group	Groups documents based on the field specified
$sort	Sorts the documents in the order specified
$project	Selects specified fields to display instead of displaying all fields
$skip	Skips no. of documents specified
$limit	Limits the first n number of documents
$out	Writes documents to new collection
$unwind	Unwinds arrays into documents

Marwadi University

List of Accumulators:-

Operator	Usecase
$sum	Returns the calculated sum of the specified value from all documents
$avg	Returns the calculated average of the specified value from all documents
$count	Returns the count of the specified value from all documents
$min	Returns the min of the specified value from all documents
$max	Returns the max of the specified value from all documents
$first	Returns the first document from the group
$last	Returns the last document from the group

Grouping with $avg:-

db.users.aggregate([{$group:{_id:"$city",averageAge:{$avg:"$age"}}}])

Finding minimum and maximum using $min and $max

Citywise minimum age:-

db.users.aggregate([{$group:{_id:"$city",minimumAge:{$min:"$age"}}}])

Citywise maximum age:-

db.users.aggregate([{$group:{_id:"$city",maximumAge:{$max:"$age"}}}])

Getting first and last document

db.users.aggregate([{$group:{_id:"$city",firstApplication:{$first:"$lastApplication"}}}])

db.users.aggregate([{$sort:{lastApplication:1}},{$group:{_id:"$city",firstApplication:{$first:"$lastApplication"}}}])

Data Modelling

- Data Modeling is the process of understanding and analyzing your business requirements and then defining structure for your data.
- This would consist of you to defining structure of data/how your data is stored, relationship between your data.

How does data modeling work for relational databases

- You define a schema which would dictate how your data is stored.
- You then normalize your schema to reduce the duplicates.

Note:-Normalization is the process to minimize the redundant information.

How does it work in MongoDB?

- MongoDB does not have a defined structure.
- It has flexible schema
- With MongoDB,you can store data in the form of documents in the way you want

- Even documents in a single collection can be different from each other and have different fields.
- This is very different from SQL where you need to have a schema definition in place before you begin to store your data.
- With MongoDB

  You first focus on building your application that will be used by your users, rather than worrying about how you store data

  Once you have built out application and have clarity as to how things would work, you define your data model and start storing data

Marwadi University

Prons of Data Modeling with MongoDB

- Since the schema we are working with is flexible, it does not forces us to define the structure of data at the beginning itself.
- You as a developer have the leverage of iterating your data model as your application evolves and grows.
- With MongoDB, we have multiple design options that we can choose from depending on your requirements and need.
- Changing the data model later on once your application evolves is very easy since there is no defined structure or schema that MongoDB has. Remember its NoSQL database.

Things to remember about data models

- Data modeling is a continuous and iterative process. So don't think of data modeling as a onetime activity. But as your application evolves, your requirement will also evolve. Hence you as an application developer will always iterate over it.So don't stress of getting it perfect on the first attempt.
- Data model and schema designs are important when it comes to MongoDB or NoSQL databases.

Data Types in MongoDB

Double(store floating point values)	String
Integer	Boolean
Object ID	Null
Array	Timestamp
Date	Code

Modeling your database

- Before creating/Modeling database first we need to understand our business.
- Data modeling is a three step process.
    - Evaluate application and its requirements
    - Defining entities and relationships
    - Finalize the data model

Evaluate application and requirements

- If your application has not yet launched so first we need to focus on below points
    - We need to understand who will be the stakeholders
    - What data would they like to see
    - How frequent will be the access to the data they wish to see?
    - How big is the data in terms of the size?

- If your application is already live:
    - We need to understand who are the current stakeholders in our system
    - How are they currently using the data?
    - What is the frequency and size of the data that they are accessing
- What data is actually important from system endpoint.(If multiple stockholders are wanting to look at a particular data, then that data might be important from system endpoint.)
- If any data is important what is the frequency in which it is accessed?
- Until what date in the past you need to have data for?

Marwadi University

Type of job	Job	Objective	Frequency	Importance
Write	Apply for jobs	Allows users to apply to jobs on website	10k job applications everyday	High
Read	List all jobs on website	Show all jobs that user can apply for	5k visitors daily	High
Read	History of applications	Show users list of all jobs that he has applied for	2k users check their history daily	Medium
Read	History of support ticket raised	Show list of all support tickets raised by users	10 users check their support ticket history daily	Low

Defining entities and relationships

- MongoDB allows data to be stored in 2 ways
    - Embedded
    - Linked
- You can define relationships as below
    - One to One relationships having embedded documents
    - One to Many relationships having embedded documents
    - One to Many relationships having document references

Finalize the data model

- Here we define the model for the database along with indexes, fields and so on.

Marwadi University

Embedded Documents

- MongoDB follows document approach where there are documents instead of rows as in SQL.
- Embedded documents are documents where one document is stored inside another document.

```
{
    _id: ObjectId("63b25d8abd3738edb929bca8"),
    name: 'Michael Morgan',
    age: 23,
    city: 'San Jose, United States',
    location: [ 40.6892, 74.0445 ],
    hobbies: [ 'Dancing' ],
    state: 'Texas',
    education: [
      {
      university: 'California University',
      start: '02-2015',
      end: '02-2016',
      degree: 'BBA'
    }
   ]
  }
```

Linked documents or referenced documents

User document

```
{
    _id: ObjectId("63b25d8abd3738edb929bca8"),
    name: 'Michael Morgan',
    age: 23,
    city: 'San Jose, United States',
    location: [ 40.6892, 74.0445 ],
    hobbies: [ 'Dancing' ],
    state: 'Texas',
    education: [
    133,
    144
        ]
  }
```

Education collection

First document

```
{
      _id: 133,
      university: 'California University',
      start: '02-2015',
      end: '02-2016',
      degree: 'BBA'
 }
```

Second document:

```
{
      _id: 144,
      university: 'California University',
      start: '02-2015',
      end: '02-2016',
      degree: 'BBA'
}
```

Storage Engine

- The storage engine is the component of the database that is responsible for managing how data is stored, both in memory and on disk.
- MongoDB supports multiple storage engines, as different engines perform better for specific workloads.
- Choosing the appropriate storage engine for your use case can significantly impact the performance of your applications.

WiredTiger Storage Engine (Default)

- WiredTiger is the default storage engine starting in MongoDB 3.2.
- It is well-suited for most workloads and is recommended for new deployments.
- WiredTiger provides a document-level concurrency model, checkpointing, and compression, among other features.
- In MongoDB Enterprise, WiredTiger also supports Encryption at Rest.

In-Memory Storage Engine:-

- Available in MongoDB Enterprise. Rather than storing documents on-disk, it retains them in memory for more predictable data latencies.

MMAPv1:-

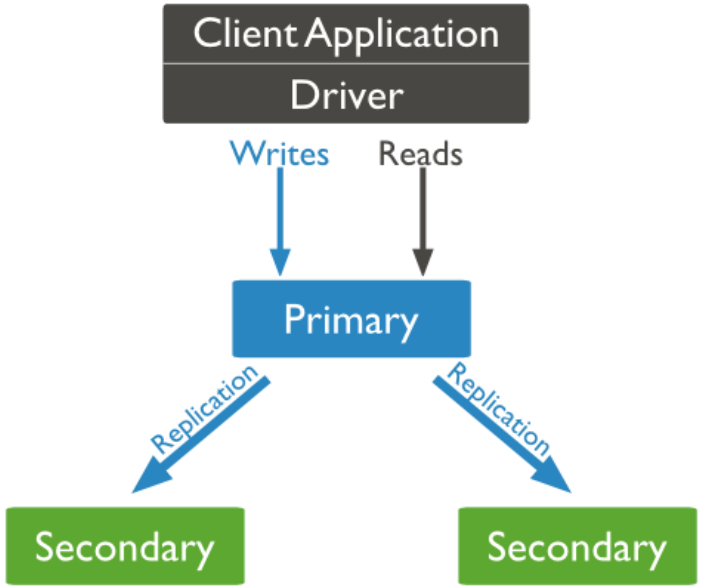
- It is MongoDB's original storage engine(prior to 3.2) based on memory mapped files.
- It excels at workloads with high volume inserts, reads and in-place updates.
- With MMAPv1,MongoDB automatically uses all free memory on the machine as its cache.
- System resource monitors show that MongoDB uses a lot of memory, but its usage is dynamic.

Replication

- Replication is the process of synchronizing data across multiple servers.
- Replication provides redundancy and increases data availability with multiple copies of data on different database servers.
- Replication protects a database from the loss of a single server.
- Replication also allows you to recover from hardware failure and service interruptions.
- With additional copies of the data, you can dedicate one to disaster recovery, reporting, or backup.

Why Replication?

- To keep your data safe
- High (24*7) availability of data
- Disaster recovery
- No downtime for maintenance



- Read scaling (extra copies to read from)
- Replica set is transparent to the application

How Replication Works in MongoDB

- MongoDB achieves replication by the use of replica set.
- A replica set is a group of mongod instances that host the same data set.
- In a replica, one node is primary node that receives all write operations. All other instances, such as secondaries, apply operations from the primary so that they have the same data set. Replica set can have only one primary node.
- Replica set is a group of two or more nodes (generally minimum 3 nodes are required).
- In a replica set, one node is primary node and remaining nodes are secondary.

- All data replicates from primary to secondary node.
- At the time of automatic failover or maintenance, election establishes for primary and a new primary node is elected.
- After the recovery of failed node, it again join the replica set and works as a secondary node.
- A typical diagram of MongoDB replication is shown in which client application always interact with the primary node and the primary node then replicates the data to the secondary nodes.

Replica Set Features

- A cluster of N nodes
- Any one node can be primary
- All write operations go to primary
- Automatic failover(fault tolerance)
- Automatic recovery
- Consensus election of primary

Performance Consideration

Marwadi University

Indexing Strategy:

- Choose the fields to index carefully based on your application's query patterns. Index only the fields that are frequently used in queries and sorting.
- Avoid over-indexing, as it can lead to increased storage requirements and slower write operations.

Index Size:

- Indexes consume storage space. It's essential to balance the performance gains from indexing with the increased storage requirements.
- Consider factors like the size of your data, available disk space, and query performance.

Index Maintenance:

- Regularly monitor and maintain your indexes. MongoDB provides tools like the db.collection.reIndex() command and the Index Intersection feature to help manage indexes efficiently.

Covered Queries:

- A covered query is one where all the fields needed for a query are part of an index, eliminating the need for MongoDB to access the documents themselves. This can significantly improve query performance.

Index Hints:

- MongoDB allows you to specify index hints to influence query execution plans.
- Use hints when you have a good understanding of your data and query patterns.

Note:-hint() method is used to force MongoDB to use a specific index for a query.

```
db.restaurants_new.find().hint( "cuisine_1" );
```

Query Optimization:

- Use the explain() method to analyze query execution plans and identify areas for optimization. Adjust your queries or indexes based on this analysis.

Sharding:

For large datasets, consider sharding your MongoDB cluster. Sharding distributes data across multiple servers and can improve query performance by parallelizing data retrieval.

Hardware and Resources:

- Ensure that your hardware resources (CPU, RAM, and disk) are appropriately sized for your MongoDB workload to avoid resource bottlenecks.

Note:-A bottleneck occurs when there is not enough capacity to meet the demand.

Regular Backups and Maintenance:

- Regularly backup and optimize your database to ensure long-term performance.
- Use tools like mongodump and mongorestore for backups and consider periodic compaction for storage efficiency.

Marwadi University

Thank You