MongoDb Query

Show all available databases:

show dbs;

Select a particular database to access, e.g. mydb. This will create

use db_name;

Delete database command is used to drop a existing database

db.dropDatabase()

Create database student

use db_name;

Show all functions that can be used with the database:

db.mydb.help()

rename database

rename collection

db.oldCollectionName.renameCollection('newCollectionName');

Show Collection

Show collections

How can use Collection

use Collection_name

Create Collection

db.createCollection('Base');

How can Drop Collection

db.Collection_name.drop()

Note : In MongoDB, "**db**" refers to the current database Where you're working

Insertion Operation

db.Collection_name.insert({State: "Bihar", Marks: 500}) **or**

db.Collection_name..save({name: 'Tom', age: 28});

Insert One

db.Collection_name .insertOne({id:1,"Name":"Vivek",Age:20})

Note : This method inserts a single document into the collection.

Insert Many

db.Collection_name.insert([{Name: "Anurag", Age: 19, Class: "12th", Mark: "65%"},
{Name: "Priyanshu", Age: 21, Mark: "76%"}])

Note : This method inserts multiple documents into the collection.

Another way to Insertion

var studentData = [{ name: "David", age: 19, city: "Boston", state: "MA", country: "USA" },
{ name: "Emma", age: 20, city: "San Francisco", state: "CA", country: "USA" }];

db.students.insertMany(studentData)

Data type in Mongodb

1. String (string)

Used to store text data. Example: "name": "John Doe"

2. Integer (int and long)

Used to store numerical values without decimals.

int: 32-bit integer,long: 64-bit integer. Example: "age": 30

3. Double (double)

Used to store floating-point values. Example: "price": 19.99

4. Boolean (bool)

Used to store true or false values. Example: "isActive": true

5. Array (array)

Used to store lists of values. Example: "hobbies": ["reading", "traveling", "swimming"]

6.Date (`date`)

- Used to store date and time values.
- Example: `"createdAt": ISODate("2023-08-21T14:48:00Z")`

7. Null (`null`)

- Represents the absence of a value. Example: `"middleName": null`

Find Data / Retrive/REad Database

db.collectioname.**find()** // acess whole data in Document

db.Collectioname.**findOne()** // acess only one data in DATABASE (First One)

```
db.data.find({Name:"John"}) // find particular name,city person
```

Update data

*db.collectioname.**updateOne**({Name:"Priyanshu"},{**$set**:{Name:"Priyanshu Singh"}});

```
*db.collectioname.updateOne({Rollno:126},{$set:{name:"Priyanshu Kumar SIngh"}});
*db.Collectioname.updateMany({name: 'Tom'},{$set:{age: 30, salary:50000}})
db.data.updateMany({ Name: 'John'},{$set:{Age:26,Branch:"EC"}})

// $set: Sets the value of a field in a document
db.collection.updateOne(
  { _id: ObjectId("...") },
  { $set: { age: 30 } } );

// $unset: Removes a field from a document
db.collection.updateOne(
  { _id: ObjectId("...") },
  { $unset: { age: "" } } );

// $inc: Increments the value of a field by a specified amount
db.collection.updateOne(
  { _id: ObjectId("...") },
  { $inc: { salary: 100 } }
```

```
);

// $push: Adds an element to an array field
db.collection.updateOne(
  { _id: ObjectId("...") },
  { $push: { hobbies: "Reading" } }
);

// $addToSet: Adds an element to an array field if it doesn't already exist
db.collection.updateOne(
  { _id: ObjectId("...") },
  { $addToSet: { hobbies: "Reading" } }
);

// $pop: Removes the first or last element of an array field
db.collection.updateOne(
  { _id: ObjectId("...") },
  { $pop: { hobbies: -1 } } // Remove the first element
);

// $pull: Removes all instances of a value from an array field
db.collection.updateOne(
  { _id: ObjectId("...") },
  { $pull: { hobbies: "Gaming" } }
);

// $rename: Renames a field
db.collection.updateOne(
  { _id: ObjectId("...") },
  { $rename: { "old_field": "new_field" } }
);
```

Update/Remove

```
db.people.deleteMany({name: 'Tom'})
 db.people.remove({name: 'Tom'}
db.Betech.deleteOne({ Name: "Priyanshu Singh" });
```

Rename data /column

```
db.data.updateOne({Age:23},{$rename:{"Name":"Fname"}})
```

Add new Field

```
db.data.updateMany(
    { "Name": "Priyanshu" }, // Condition to select documents
```

```
    { $set: { "membership": "gold" } } // Set the new field
);
```

Aggregation

$match

```
db.data.aggregate([{$match:{"Name":"Priyanshu"}}])
```

$Limit

```
db.data.find().limit(3);
```

Skip() :

```
db.data.find().skip(3);
```

Count( ):

```
db.data.find().count();
```

Sort():

```
db.data.find().sort({ Name: -1 });
```

Sum()

```
db.sales.aggregate([ { $group: { _id: "$product",
      totalSales: { $sum: "$amount" }  } }]);
```

AVG()

```
db.collection.aggregate([ {   $group: {  _id: "$groupField",
      average: { $avg: "$numericField" } } }]);
```

group()

```
db.temp.aggregate([
  {$group: {_id: "$Age", count: {$sum: 1}, data: {$push: "$
$ROOT"}}},  {$sort: {_id: 1}}]);

db.temp.aggregate([
  {$match: {Gender: "Female"}},
  {$group: {_id: "$Gender", count: {$sum: 1}, data: {$push: "$
$ROOT"}}}]);
```

min(), max()

```
db.temperatures.aggregate([
  {$group: {_id: null, minTemperature: {$min: "$temperature"},
maxTemperature: {$max: "$temperature"}}}
```

```
]);
```

toUpper()

```
db.users.aggregate([
  {$project: {_id: 0, upperName: {$toUpper: "$name"}}}
]);
```

$toLower()

```
db.users.aggregate([
  {$project: {_id: 0, lowerName: {$toLower: "$name"}}}
]);
```

lookup()

```
db.orders.aggregate([  {$lookup: { from: "customers",
localField: "customerId", foreignField: "_id", as: "customer"
  }} ]);
```

unwind()

```
db.products.aggregate([{$unwind: "$tags"}]);
```

$match: Filters documents to pass only those that match specified conditions.
$group: Groups documents by a specified expression and applies accumulator expressions.
$project: Reshapes documents by including, excluding, or renaming fields.
$sort: Sorts documents by specified fields.
$limit: Limits the number of documents passed to the next stage in the pipeline.
$unwind: Deconstructs an array field from the input documents to output a document for each element.
$lookup: Performs a left outer join to another collection in the same database to filter in documents from the "joined" collection for processing.
$sum: Calculates the sum of numeric values.
$avg: Calculates the average of numeric values.
$min: Returns the minimum value.
$max: Returns the maximum value.
$count: Returns the total number of documents in the aggregation pipeline.
$add: Adds two dates together.
$subtract: Subtracts two dates.
$concat: Concatenates strings.

Logical operator

***And *Or *Not**

And

```
db.data.find({  $and: [ { Gender: "Female" },{ Age: { $gt: 20 } }  ]});
```

or

```
db.data.find({  $or: [ { Gender: "Female" },{ Age: { $gt: 20 } }  ]});
```

not

```
db.data.find({  $not: [ { Gender: "Female" },{ Age: { $gt: 20 } }  ]});
```

Comparision Operator

$lt $gt $eq $ne $gte $lte $nin $in

$eq (Equal operator)

```
db.data.find({ Name: { $eq: 'Vivek' } });
```

$ne(Not equal to)

```
db.data.find({ Name: { $ne: 'Vivek' } });
```

$gt (Greater)

db.data.find({ "Age" : {$gt:20}})

$gte (Greater than equal to)

```
db.data.find({ Age: { $gte: 20 } });
```

$lt(Less than)

```
db.data.find({ Age: { $lt: 15 } });
```

$lte(Less than equal)

```
db.data.find({ Age: { $lte: 15 } })
```

$in(in)

```
db.data.find({ Age: { $in: [10, 15, 19] } });
```

$nin(not in)

```
db.data.find({ Age: { $nin: [10, 15, 19] } });
```

Elements Operator

1.$exists 2.$type 3.$size

exists : db.users.find({ email: { $exists: true } });

$type : db.users.find({ age: { $type: "int" } });

$size : db.users.find({ hobbies: { $size: 3 } });

Cursors

Cursors in MongoDB are used to efficiently retrieve large result sets from queries, providing control over the data retrieval process.

count() limit() skip() sort()

db.products.find({ price: { $gt: 250 } }).**count();**

db.products.find({ price: { $gt: 250 } }).**limit(5);**

db.products.find({ price: { $gt: 250 } }).l**imit(5).skip(2);**

db.products.find({ price: { $gt: 250 } }).**limit(5).skip(2).sort();**

Evaluation query Operator

$expr :

db.collection.find({ $expr: { $and: [{ $gt: ["$totalAmount", 100] }, { $lt: ["$totalAmount", 500] }] } })

Embedded document Insertion

db.users.insertOne({ name: "John Doe", age: 30, email: "johndoe@example.com",

address: { street: "123 Main St", city: "New York", state: "NY", zip: "10001" }});

Indexing

"That improves the efficiency of query operations. When you create an index on a collection, MongoDB builds a data structure using a B-tree, which allows it to quickly locate and retrieve the documents.

db.users.createIndex({ name: 1 });

Array

Arrays in MongoDB are used to store multiple values in a single field.

db.users.insertOne({ name: "Alice", age: 30, **hobbies: ["reading", "hiking", "coding"]**});

Array Method

- **$pull:** Removes all instances of a value from an array.
- **$pop**: Removes the first or last element of an array.

- **$pullAll**: Removes all instances of specified values from an array.
- **$push**: Adds a specified value to an array.
- **$** (Positional Operator): Updates the first matching element in an array.
- **$[]** (All Positional Operator): Updates all elements in an array.
- **$position**: Specifies the position to add an element with `$push`.
- **$addToSet**: Adds a value to an array only if it doesn't already exist.
- **$each**: Allows multiple values to be added to an array with `$push` or `$addToSet`.
- **$sort**: Sorts the elements in an array during an update operation.

```
// 1. $pull
db.collection.update(
  { name: "James" },
  { $pull: { hobbies: "gardening" } }
)
// 2. $pop
db.collection.update(
  { name: "James" },
  { $pop: { hobbies: 1 } } // Use -1 to remove the first item
)
// 3. $pullAll
db.collection.update(
  { _id: ObjectId("1234567890") },
  { $pullAll: { hobbies: ["reading", "swimming"] } }
)
// 4. $push
db.collection.update(
  { _id: ObjectId("1234567890") },
  { $push: { hobbies: "painting" } }
)
// 5. $ (Positional Operator)
```

```
db.collection.update(
   { "hobbies": "reading" },
   { $set: { "hobbies.$": "cooking" } }
)
// 6. $[] (All Positional Operator)
db.collection.update(
   {},
   { $inc: { "hobbies.$[]": 5 } }
)
// 7. $position
db.collection.update(
   { _id: ObjectId("1234567890") },
   { $push: { hobbies: { $each: ["singing"], $position: 1 } } }
)

// 8. $addToSet
db.collection.update(
   { _id: ObjectId("1234567890") },
   { $addToSet: { hobbies: "dancing" } }
)

// 9. $each
db.collection.update(
   { _id: ObjectId("1234567890") },
   { $push: { hobbies: { $each: ["singing", "dancing"] } } }
)
```

// 10. $sort

```
db.collection.update(
  { _id: ObjectId("1234567890") },
  { $push: { hobbies: { $each: ["singing", "dancing"], $sort: 1 } } }
)
```

String

1. **$concat**

```
db.collection.update(   { _id: ObjectId("1234567890") },
  { $set: { fullName: { $concat: ["$firstName", " ", "$lastName"] } } })
```

2. $toLower

```
db.collection.aggregate(
  [  { $project: { lowerCaseName: { $toLower: "$name" } } }   ])
```

3. $toUpper

```
db.collection.aggregate(
  [    { $project: { upperCaseName: { $toUpper: "$name" } } }  ])
```

4. $substr

db.collection.aggregate([{ $project: { substring: { $substr: ["$name", 0, 5] } } }])

5. $trim

```
db.collection.aggregate(   [      { $project: { trimmedName: { $trim: { input: "$name" } } } }
  ])
```

6. $ltrim

```
db.collection.aggregate(   [      { $project: { leftTrimmedName: { $ltrim: { input: "$name" } } }
}  ])
```

8. $rtrim

```
db.collection.aggregate(   [      { $project: { rightTrimmedName: { $rtrim: { input:
"$name" } } } }   ])
```

9. $replaceOne

```
db.collection.update(
   { _id: ObjectId("1234567890") },
   { $set: { name: { $replaceOne: { input: "$name", find: "John", replacement: "Jonathan" } } } }
)
```

10. $replaceAll

```
db.collection.update(
   { _id: ObjectId("1234567890") },
   { $set: { name: { $replaceAll: { input: "$name", find: "John", replacement: "Jonathan" } } } }
)
```

Perform validation of schema

```
db.createCollection("users", {   validator: {  $jsonSchema: {   bsonType: "object",   required: ["username", "email"],     properties: {nusername: {m bsonType: "string",   description: "must be a string and is required"
 },  email: { bsonType: "string",pattern: "^\\S+@\\S+\\.\\S+$",description: "must be a string and match the regular expression pattern" },
  age: {bbsonType: "int", minimum: 18, description: "must be an integer and greater than or equal to 18"
        } }}})
```

Insertion perform

```
db.users.insertOne({  username: "Saurabh",  email: "saurabh@gmail.com",  age: 21})
```

MongoDB

MongoDB is open-source a document-oriented NoSQL database system that stores a large amount of data in the form of documents, providing high scalability, flexibility, and performance. MongoDB stores data in a JSON document structure form. cross-platform database system.

Note : its provide nested insertion

MongoDB vs MySQL

MongoDB	MySQL
Database	Database
Collection	Table
Document	Row , Column

Where to Use MongoDB?

- Mobile and Social Infrastructure
- Data Hub . Big Data .Content Management

Supported Languages by MongoDB

MongoDB provides official driver support for:

- C • C++ •Java •Node.js •PHP •Python

Types of NoSQL Databases

NoSQL databases can be classified into 4 basic types:

1. Key-value store NoSQL database 2.Document store NoSQL database

3.Column store NoSQL database 3.Graph-based NoSQL database

Why MongoDB is Known as the Best NoSQL Database?

MongoDB is considered the best NoSQL database because it is:

- Document Oriented • Rich Query language
- High Performance •Easily Scalable

Does MongoDB Support Primary-Key, Foreign-Key Relationship?

No, by default, MongoDB doesn't support primary key-foreign key relationships.

MongoDB Document Representation

Yes, MongoDB uses BSON (Binary JSON) to represent document structure.

Language Used for MongoDB Implementation

MongoDB is written and implemented in C++.

Collections in MongoDB

Collections, documents, and databases are important parts of MongoDB. A database contains collections, and a collection contains documents. The documents contain data.

MYSQL VS MongoDB

Aspect	MySQL	MongoDB
Release Date	23 May 1995	11 February 2009
Written in	C and C++	C, C++, and Java
Data Model	Table, Row, Columns, Joins	Collection, Document, Field
Usage by Organizations	Twitter, YouTube, Netflix	Twitter, Paypal, NASA
Schema Requirement	Requires schema definition	No prior schema required
Join Operation Support	Supports Joins	Doesn’t support Joins

What are the advantages of MongoDB?

- It supports cross-platform • It supports both types of scaling – Horizontal & Vertical
- It provides High Performance • It is easily scalable It does not require any complex joins to retrieve data

What is MongoDB Shell

- MongoDB Shell is a JavaScript shell that allows us to interact with MongoDB instances using the command line. It is very useful to perform any administrative work along with any other data operations-related commands

Document : a document is a basic unit of data storage and represents a single record within a collection. Documents are stored in BSON (Binary JSON) format

Replica set :A replica set in mongodb is a group of mongodb servers that stores same data to provide redundancy and high avaliablity

Sharing : Sharing in mongodb is method of partitioning data across multiple servsers to improve performances and sacliblity

Indexing in MongoDB

In MongoDB, indexing plays a crucial role in improving query performance by efficiently locating and retrieving documents from collections. Indexes are data structures that store a small portion of the collection’s data set in an easy-to-traverse form.

BSON :
BSON, which stands for Binary JSON, is a binary-encoded serialization of JSON-like documents. It is the primary data representation format used by MongoDB to store and exchange data. BSON extends the JSON model by adding some additional data types

Data Types : BSON supports a wider range of data types compared to JSON. Some of the data types supported by BSON include:

- Double •String •Object •Array • Boolean • Null • Integer •Long • Date

Json : In MongoDB, JSON (JavaScript Object Notation) plays a fundamental role in representing and interacting with data. MongoDB stores data in a binary-encoded format called BSON (Binary JSON), which is a binary representation of JSON-like documents. Here's how JSON is used in MongoDB:

Embedded Document : Embedded or Nested Document. Embedded document or nested documents are those types of documents which contain a document inside another document

Introduction to NoSQL

NoSQL is a type of database management system (DBMS) that is designed to handle and store large volumes of unstructured and semi-structured data. NoSQL databases store data in the form of collections of documents in JSON or BSON format.

Example : Mongodb , Amazon DynamoDB

Advantage of Nosql :

1. Ideal for Data Storage: One of the exclusive NoSQL database advantages is its ability to store a large amount of data.

2.Performance: NoSQL databases are designed to handle large amounts of data and traffic, which means that they can offer improved performance compared to traditional relational databases.

3.Cost-effectiveness: NoSQL databases are often more cost-effective than traditional relational databases, as they are typically less complex and do not require expensive hardware or software.

4.Agility: Ideal for agile development.

Disadvantages of NoSQL:

Do you have Mongodb Schema : Mongodb uses dynamic Schemas Without defining the Structure you can cretye collection This dynamic schema allows for flexibility in data modeling and makes it easy to store heterogeneous data within the same collection.

Command Use to backup : Mongodump

Command restore backup : Mongorestore

Aggregation : Aggregation in MongoDB is a framework that processes data records and returns computed results. It is used for performing operations such as filtering, grouping, sorting, and summarizing data

What are the differences between the update and save methods in MongoDB?

`Update` Method:

- The `update` method is used to modify existing documents in a collection based on specified criteria

`save` Method:

- The `save` method is used to either update an existing document in a collection or insert a new document if the document does not already exist.**Sharding :** Sharding is a method for distributing data across multiple servers in MongoDB. It allows for horizontal scaling by splitting large datasets into smaller, more manageable pieces called shards

Atlas : MongoDB Atlas is a fully managed cloud database service provided by MongoDB. It simplifies the process of deploying, managing, and scaling MongoDB databases by handling many of the administrative tasks associated with running a database

MongoDB

SQL