




Mongo Db Database




MongoDb Database

- MongoDB is a **NoSQL database**. In simple terms, it's a type of database used to store and manage large amounts of data, but it's different from traditional databases like MySQL or SQL Server.
- Here's an easy explanation of MongoDB:
 1. **NoSQL** means it doesn't use the standard tables and rows structure that traditional databases do. Instead, MongoDB stores data in a more flexible way using **documents**.
 2. Each **document** is like a **record** in a database and is made up of **key-value pairs**. You can think of a document as a **JSON object**, which is similar to how we write data in programming.



Example of a MongoDB document (in JSON format):

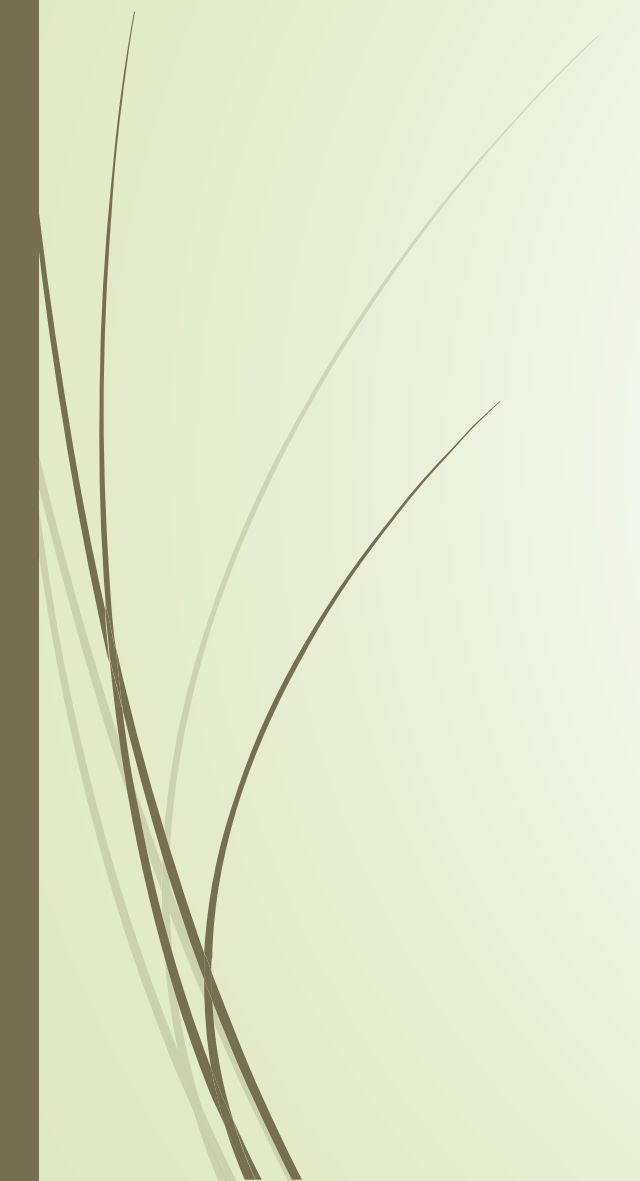
json

 Copy code

```
{  
  "name": "John",  
  "age": 30,  
  "email": "john@example.com"  
}
```



Why MongoDB is used


- Scalability: MongoDB can handle huge amounts of data and scale across multiple servers if needed.
 - Flexibility: You can store different types of data in the same collection without worrying about a fixed structure.
 - Speed: It's fast for retrieving and storing data because it uses a simpler, more flexible model compared to traditional databases.
- 

What is a Document in MongoDB?

- A **Document** is the basic unit of data in MongoDB. It is a set of key-value pairs, similar to JSON objects, where the keys are field names and the values can be of various types (e.g., string, number, array, etc.). Documents are stored in **collections**.

Example of a document:

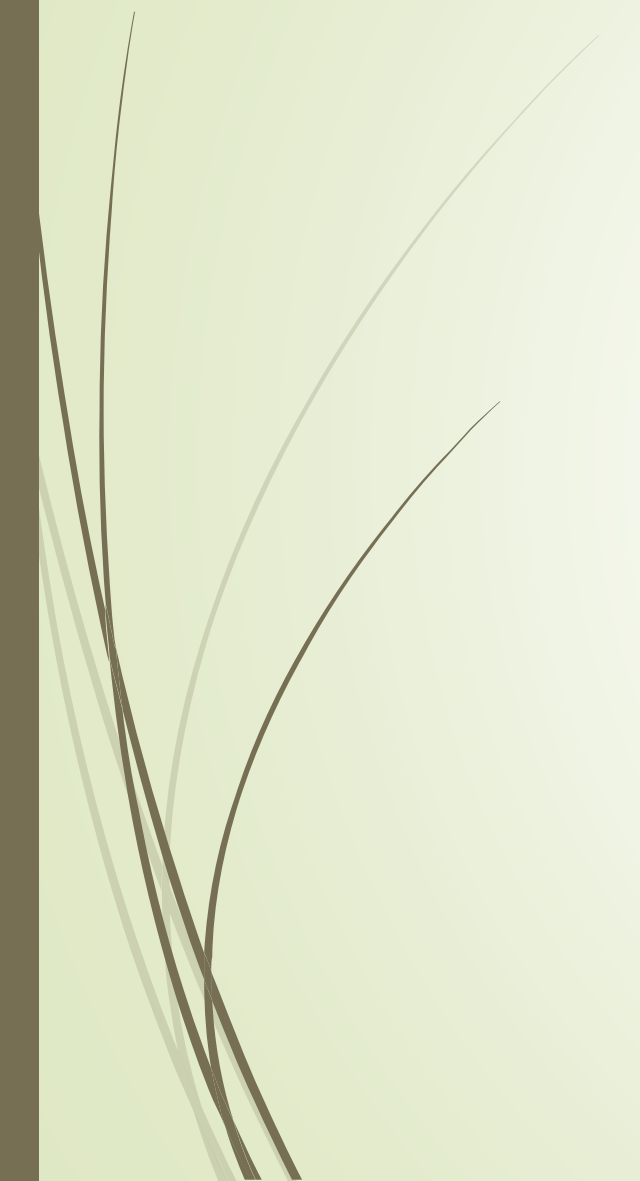
json

 Copy code

```
{  
  "name": "Alice",  
  "age": 25,  
  "email": "alice@example.com"  
}
```



What is a Collection in MongoDB?

- A **Collection** is a group of MongoDB documents, similar to a **table** in relational databases. Collections do not enforce a schema, meaning documents within the same collection can have different structures.
- 

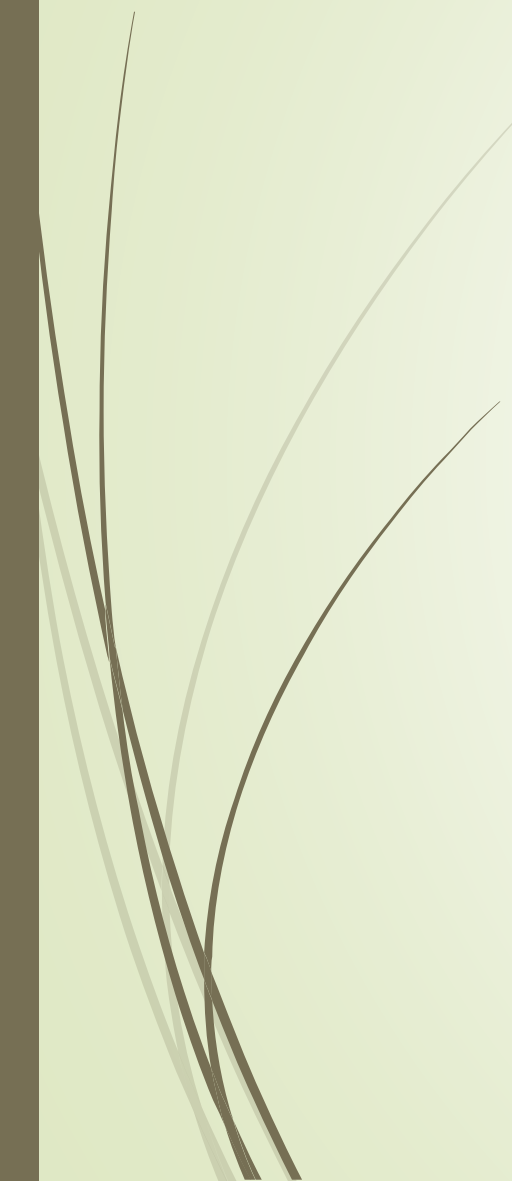


What are the differences between MongoDB and SQL databases?

- Data Model: MongoDB uses documents and collections (NoSQL), whereas SQL databases use tables and rows.
- Schema: MongoDB has a flexible schema, while SQL databases enforce a fixed schema.
- Scaling: MongoDB can be easily sharded (horizontal scaling), while SQL databases typically scale vertically (by upgrading hardware).
- Joins: MongoDB generally avoids using joins (though you can use lookup for aggregation), while SQL databases use them extensively.

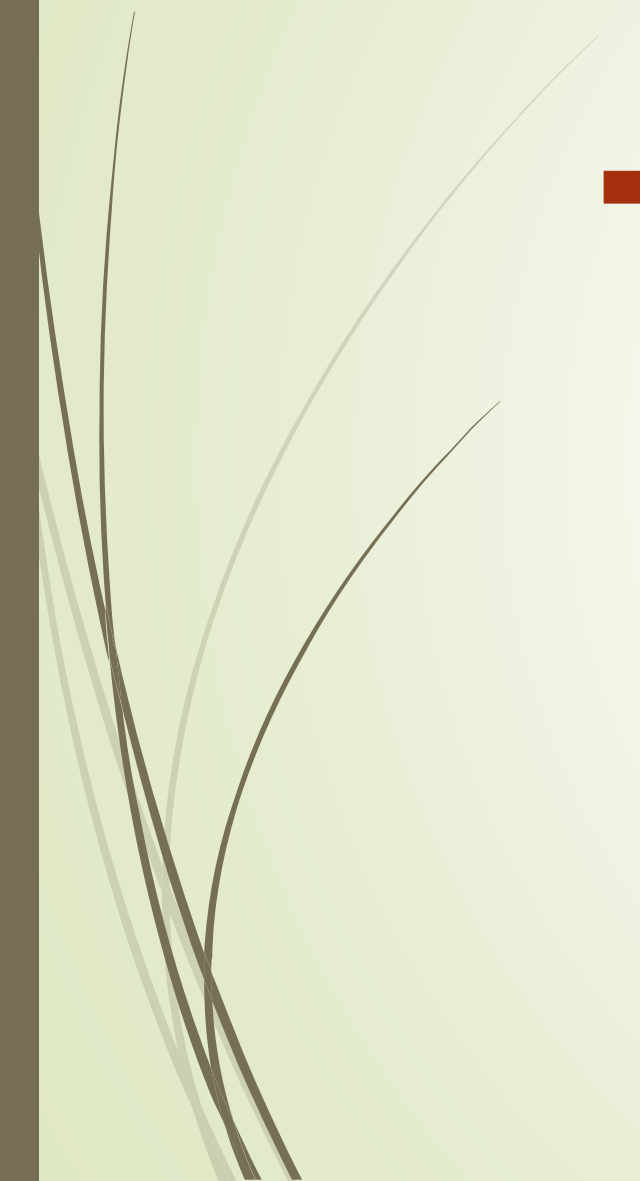



What are Indexes in MongoDB?

- An Index is a special data structure used by MongoDB to improve query performance. Indexes help MongoDB to quickly locate and retrieve documents, especially for large datasets.
 - By default, MongoDB creates an index on the `_id` field of each document.
 - You can create custom indexes using `db.collection.createIndex()`.
- 

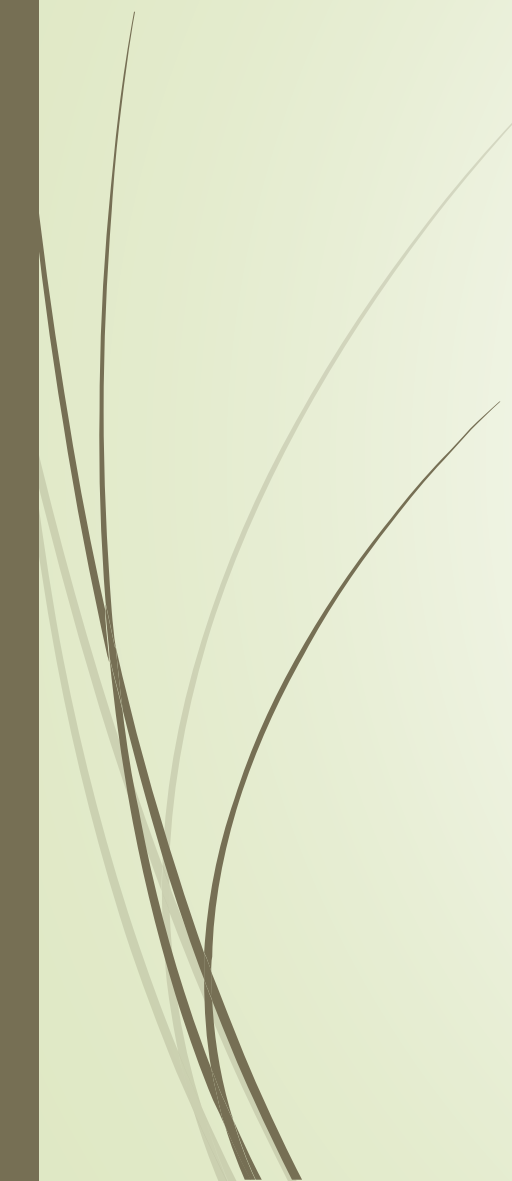


What is Sharding in MongoDB?

- **Sharding** is the process of distributing data across multiple machines to handle large datasets or high-throughput operations. Each shard contains a subset of the data. MongoDB automatically distributes data among the shards based on a shard key.
- 

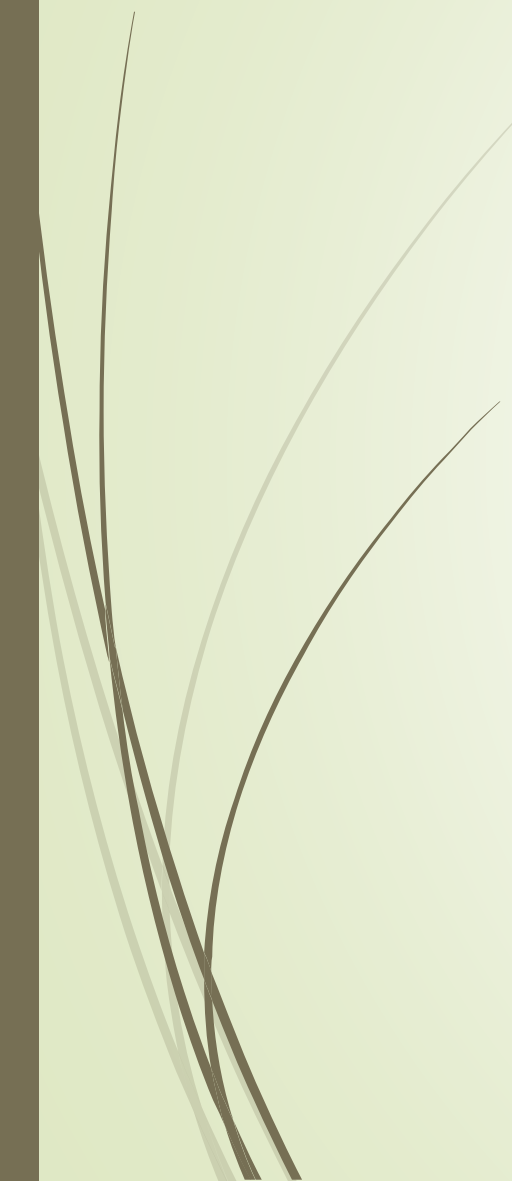


What are Aggregation Pipelines in MongoDB?

- Aggregation Pipelines are used to process and transform data in MongoDB. They allow you to perform operations such as filtering, grouping, sorting, and projecting data, in a pipeline format.
 - Common stages in an aggregation pipeline include `$match`, `$group`, `$sort`, and `$project`.
- 

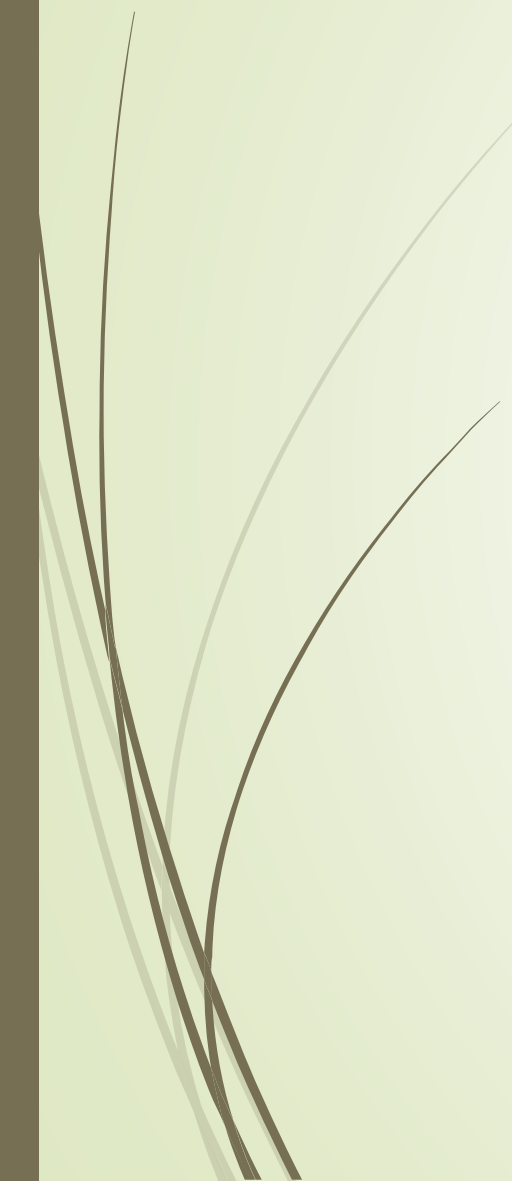


What is the purpose of the `$_id` field in MongoDB?

- The `$_id` field is a unique identifier for each document in MongoDB. By default, MongoDB generates a unique `_id` for every document if it is not provided by the user. The `_id` field can be any data type, but typically it is an `ObjectId`.
- 



What is the difference between `find()` and `aggregate()` in MongoDB?

- `find()`: It is used to query documents in a collection based on certain criteria. It's ideal for simple queries.
 - `aggregate()`: It is used for more complex queries that require transformations, grouping, sorting, and other advanced operations. It allows using the aggregation pipeline for data processing.
- 



What are Transactions in MongoDB?

- MongoDB supports multi-document **transactions** since version 4.0. This means you can perform multiple operations on different documents or collections and ensure that all operations are committed or rolled back together (ACID properties).



Explain the term “Joins” in MongoDB.

- Unlike relational databases, MongoDB doesn't support joins in the traditional way. However, you can use the \$lookup stage in an aggregation pipeline to simulate a join between two collections.



How does MongoDB ensure data integrity?

- MongoDB ensures data integrity through **replication**, **journaling**, and **write concerns**. Write concerns specify the level of acknowledgment requested from MongoDB for write operations, ensuring data consistency.

Relational DB

database

table

row

column

MongoDB

database

collection

document

field



Create Database

```
employees> show dbs
Employees      80.00 KiB
admin          40.00 KiB
config         108.00 KiB
local          40.00 KiB
employees> use Student
switched to db Student
Student> |
```



Show databases

- **Use**

- **Show dbs**

Create Collection

```
employees> show dbs
Employees      80.00 KiB
admin          40.00 KiB
config         108.00 KiB
local          40.00 KiB
employees> use Student
switched to db Student
Student> db.students.insertOne({ })
```

If there is no present Student collection That it created

Find Single Document based on fields

```
Student> db.students.findOne( { name : 'Somesh Marathe'})
{
  _id: ObjectId('677be0a67607894e434eeb8b'),
  name: 'Somesh Marathe',
  age: 22
}
Student>
```

Update into Document

```
Student> db.students.updateOne({ name : 'Somes Marathe' } , { $set : { age : 23 } } )
{
  acknowledged: true,
  insertedId: null,
  matchedCount: 1,
  modifiedCount: 1,
  upsertedCount: 0
}
Student> db.students.find()
[
  { _id: ObjectId('677be0957607894e434eeb8a'), name: 'ram', age: 18 },
  {
    _id: ObjectId('677be0a67607894e434eeb8b'),
    name: 'Somes Marathe',
    age: 23
  }
]
Student>
```

Nested Document Limit

```
Student> db.students.findOne( {name : 'Somesh Marathe'})
{
  _id: ObjectId('677be0a67607894e434eeb8b'),
  name: 'Somesh Marathe',
  age: 23,
  idCard: { hasPanCard: false, hasIdCard: false }
}
Student>
```

Here Id Card itself is won document

Update All Document or Insert new Field in All Document

```
Student> db.students.updateMany( {}, { $set : { hobbies : [ ' dancing' , 'acting' ] } } )
{
  acknowledged: true,
  insertedId: null,
  matchedCount: 2,
  modifiedCount: 2,
  upsertedCount: 0
}
Student> db.students.find();

[
  {
    _id: ObjectId('677be0957607894e434eeb8a'),
    name: 'ram',
    age: 18,
    hobbies: [ ' dancing', 'acting' ]
  },
  {
    _id: ObjectId('677be0a67607894e434eeb8b'),
    name: 'Somesh Marathe',
    age: 23,
    idCard: { hasPanCard: false, hasIdCard: false },
    hobbies: [ ' dancing', 'acting' ]
  }
]
```

Searching In Nested Document

```
Student> db.students.findOne({ 'idCard.hasPanCard' : false})
{
  _id: ObjectId('677be0a67607894e434eeb8b'),
  name: 'Somesb Marathe',
  age: 23,
  idCard: { hasPanCard: false, hasIdCard: false },
  hobbies: [ ' dancing', 'acting' ]
}
Student> |
```

Search in List

```
Student> db.students.find({hobbies : 'acting'})
[
  {
    _id: ObjectId('677be0957607894e434eeb8a'),
    name: 'ram',
    age: 18,
    hobbies: [ 'dancing', 'acting' ]
  },
  {
    _id: ObjectId('677be0a67607894e434eeb8b'),
    name: 'Somesh Marathe',
    age: 23,
    idCard: { hasPanCard: false, hasIdCard: false },
    hobbies: [ 'dancing', 'acting' ]
  }
]
Student>
```

create



C

R

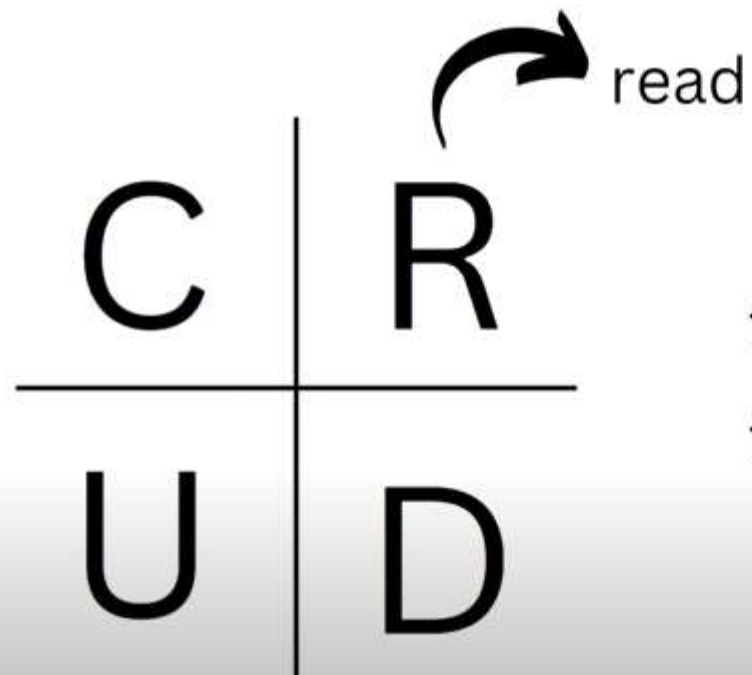
U

D

`insertOne(data, options)`

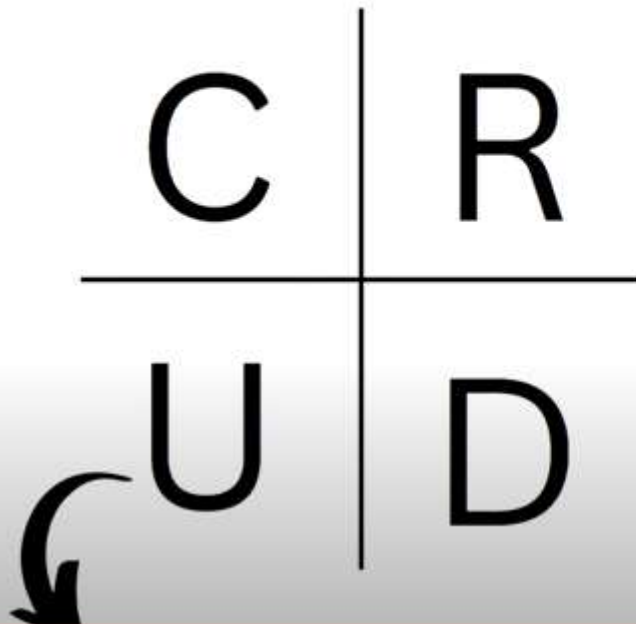
`insertMany(data, options)`





`find(filter, options)`

`findOne(filter, options)`

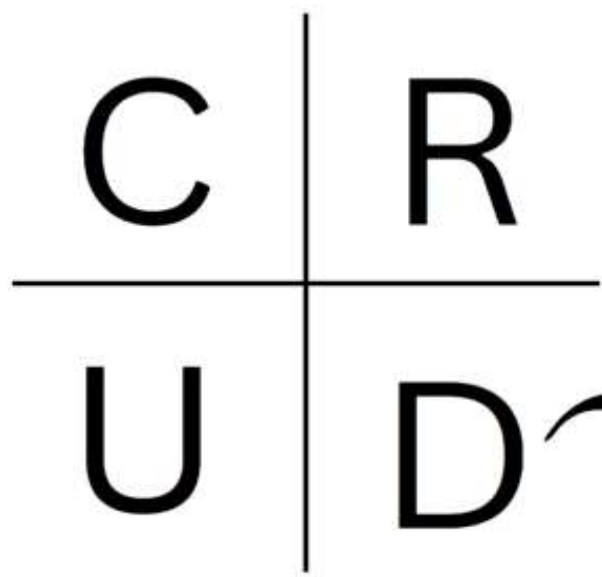


`updateOne(filter,data, options)`

`updateMany(filter,data, options)`

`relaceOne(filter,data, options)`





deleteOne(filter, options)
deleteMany(filter, options)


delete





How to see collection

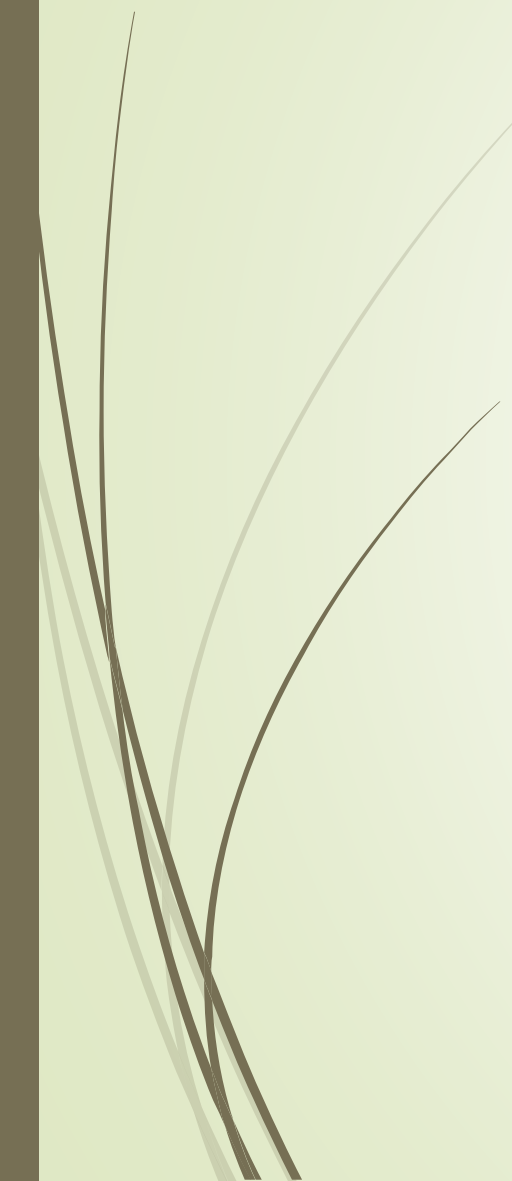
- ➡ Show Collection cmd

Find() , FindOne()

```
students
Student> db.students.find();
[
  {
    _id: ObjectId('677be0957607894e434eeb8a'),
    name: 'ram',
    age: 18,
    hobbies: [ 'dancing', 'acting' ]
  },
  {
    _id: ObjectId('677be0a67607894e434eeb8b'),
    name: 'Somesh Marathe',
    age: 23,
    idCard: { hasPanCard: false, hasIdCard: false },
    hobbies: [ 'dancing', 'acting' ]
  }
]
Student> db.students.findOne();
{
  _id: ObjectId('677be0957607894e434eeb8a'),
  name: 'ram',
  age: 18,
  hobbies: [ 'dancing', 'acting' ]
}
Student> db.students.findOne({ age : 18})
{
  _id: ObjectId('677be0957607894e434eeb8a'),
  name: 'ram',
  age: 18,
  hobbies: [ 'dancing', 'acting' ]
}
Student> db.students.findOne( { hobbies : 'dancing'})
null
Student> db.students.findOne( { hobbies : 'acting'})
{
  _id: ObjectId('677be0957607894e434eeb8a'),
  name: 'ram',
  age: 18,
  hobbies: [ 'dancing', 'acting' ]
}
Student> |
```



Limit() , count() , forEach() in MongoDB

- Limit() : for how many document want.
 - Count() : check how many document present in Collection.
 - Foreach() : retrieve all the document in the list instead of given 20 starting document.
 - Find() : give only 20 starting document
- 

```
]
Student> db.students.find().count();
2
Student> db.students.find().limit(1);
[
  {
    _id: ObjectId('677be0957607894e434eeb8a'),
    name: 'ram',
    age: 18,
    hobbies: [ 'dancing', 'acting' ]
  }
]
Student> db.students.find().forEach( (x) => {printjson(x)});
{
  _id: ObjectId('677be0957607894e434eeb8a'),
  name: 'ram',
  age: 18,
  hobbies: [
    'dancing',
    'acting'
  ]
}
{
  _id: ObjectId('677be0a67607894e434eeb8b'),
  name: 'Somesh Marathe',
  age: 23,
  idCard: {
    hasPanCard: false,
    hasIdCard: false
  },
  hobbies: [
    'dancing',
    'acting'
  ]
}
```

Greater Than , Less Than , greater Than Equal To vice versa

```
Student> db.students.find({age : {$lt:19}})
[
  {
    _id: ObjectId('677be0957607894e434eeb8a'),
    name: 'ram',
    age: 18,
    hobbies: [ 'dancing', 'acting' ]
  }
]
Student> db.students.find({age : {$lt:25}})
[
  {
    _id: ObjectId('677be0957607894e434eeb8a'),
    name: 'ram',
    age: 18,
    hobbies: [ 'dancing', 'acting' ]
  },
  {
    _id: ObjectId('677be0a67607894e434eeb8b'),
    name: 'Somesh Marathe',
    age: 23,
    idCard: { hasPanCard: false, hasIdCard: false },
    hobbies: [ 'dancing', 'acting' ]
  }
]
```

```
Student> db.students.find({age : {$lte:23}})
[
  {
    _id: ObjectId('677be0957607894e434eeb8a'),
    name: 'ram',
    age: 18,
    hobbies: [ 'dancing', 'acting' ]
  },
  {
    _id: ObjectId('677be0a67607894e434eeb8b'),
    name: 'Somesh Marathe',
    age: 23,
    idCard: { hasPanCard: false, hasIdCard: false },
    hobbies: [ 'dancing', 'acting' ]
  }
]
Student> db.students.find({age : {$gte:23}})
[
  {
    _id: ObjectId('677be0a67607894e434eeb8b'),
    name: 'Somesh Marathe',
    age: 23,
    idCard: { hasPanCard: false, hasIdCard: false },
    hobbies: [ 'dancing', 'acting' ]
  }
]
Student> db.students.find({age : {$gt:23}})
Student>
```

```
Student> db.students.find({age : {$gt:23}})

Student> db.students.find({age : {$lte:23}}).count();
2
Student> db.students.find({age : {$lte:23}}).count().limit(1);
TypeError: db.students.fi ... ount().limit is not a function
Student> db.students.find({age : {$lte:23}}).limit(1);
[
  {
    _id: ObjectId('677be0957607894e434eeb8a'),
    name: 'ram',
    age: 18,
    hobbies: [ ' dancing', 'acting' ]
  }
]
Student> |
```

```
Student> db.students.find({age : {$lte:23 , $gte : 18}});
[
  {
    _id: ObjectId('677be0957607894e434eeb8a'),
    name: 'ram',
    age: 18,
    hobbies: [ ' dancing', 'acting' ]
  },
  {
    _id: ObjectId('677be0a67607894e434eeb8b'),
    name: 'Somesb Marathe',
    age: 23,
    idCard: { hasPanCard: false, hasIdCard: false },
    hobbies: [ ' dancing', 'acting' ]
  }
]
Student> db.students.find({age : {$lte:23 , $gte : 18}}).limit(1);
[
  {
    _id: ObjectId('677be0957607894e434eeb8a'),
    name: 'ram',
    age: 18,
    hobbies: [ ' dancing', 'acting' ]
  }
]
Student> |
```

InsertOne() , Insert()

```
Student> db.students.insert( { name : "Krishn" , age : 22});
DeprecationWarning: Collection.insert() is deprecated. Use insertOne, insertMany, or bulkWrite.
{
  acknowledged: true,
  insertedIds: { '0': ObjectId('677bf5907607894e434eeb8c') }
}
Student> db.students.insert( { name : "Narayan" , age : 24});
{
  acknowledged: true,
  insertedIds: { '0': ObjectId('677bf59f7607894e434eeb8d') }
}
Student> db.students.insertMany( [{ name : "Lakshmi" , age : 21} , { name : "Radha" , age : 19} ])
{
  acknowledged: true,
  insertedIds: {
    '0': ObjectId('677bf5da7607894e434eeb8e'),
    '1': ObjectId('677bf5da7607894e434eeb8f')
  }
}
```





```
Student> db.students.find();
[
  {
    _id: ObjectId('677be0957607894e434eeb8a'),
    name: 'ram',
    age: 18,
    hobbies: [ 'dancing', 'acting' ]
  },
  {
    _id: ObjectId('677be0a67607894e434eeb8b'),
    name: 'Somesh Marathe',
    age: 23,
    idCard: { hasPanCard: false, hasIdCard: false },
    hobbies: [ 'dancing', 'acting' ]
  },
  {
    _id: ObjectId('677bf5907607894e434eeb8c'),
    name: 'Krishn',
    age: 22
  },
  {
    _id: ObjectId('677bf59f7607894e434eeb8d'),
    name: 'Narayan',
    age: 24
  },
  {
    _id: ObjectId('677bf5da7607894e434eeb8e'),
    name: 'Lakshmi',
    age: 21
  },
  { _id: ObjectId('677bf5da7607894e434eeb8f'), name: 'Radha', age: 19 }
]
Student>
```

UpdateOne , UpdateMany

```
Student> db.students.updateOne({name : "Krishn"} , {$set : {age : 24}})
{
  acknowledged: true,
  insertedId: null,
  matchedCount: 1,
  modifiedCount: 1,
  upsertedCount: 0
}
```

```
Student> db.students.updateMany( {} , {$set : {hobbies : ['acting' , 'dancing'] }})
{
  acknowledged: true,
  insertedId: null,
  matchedCount: 6,
  modifiedCount: 6,
  upsertedCount: 0
}
Student> db.students.find();
```



```
Student> db.students.find({age : { $lt : 20}})
```

```
[
  {
    _id: ObjectId('677be0957607894e434eeb8a'),
    name: 'ram',
    age: 18,
    hobbies: [ 'acting', 'dancing' ]
  },
  {
    _id: ObjectId('677bf5da7607894e434eeb8f'),
    name: 'Radha',
    age: 19,
    hobbies: [ 'acting', 'dancing' ]
  }
]
```

```
Student> db.students.updateMany({age : { $lt : 20}} , {$set : {age : 20}})
```

```
{
  acknowledged: true,
  insertedId: null,
  matchedCount: 2,
  modifiedCount: 2,
  upsertedCount: 0
}
```

DeleteOne() , DeleteMany() in MongoDB

```
Student> db.students.deleteMany({age : 18})
{ acknowledged: true, deletedCount: 0 }
Student> db.students.deleteMany({age : 20})
{ acknowledged: true, deletedCount: 2 }
Student> db.students.deleteOne({age : 25})
{ acknowledged: true, deletedCount: 0 }
Student> db.students.deleteOne({age : 24})
{ acknowledged: true, deletedCount: 1 }
Student> db.students.find();
[
  {
    _id: ObjectId('677be0a67607894e434eeb8b'),
    name: 'Somesh Marathe',
    age: 23,
    idCard: { hasPanCard: false, hasIdCard: false },
    hobbies: [ 'acting', 'dancing' ]
  },
  {
    _id: ObjectId('677bf59f7607894e434eeb8d'),
    name: 'Narayan',
    age: 24,
    hobbies: [ 'acting', 'dancing' ]
  },
  {
    _id: ObjectId('677bf5da7607894e434eeb8e'),
    name: 'Lakshmi',
    age: 21,
    hobbies: [ 'acting', 'dancing' ]
  }
]
Student> |
```

Projection In MongoDB


Projection in MongoDB refers to the process of specifying which fields of a document you want to include or exclude in the result set of a query. Instead of retrieving the entire document, projection allows you to retrieve only the fields you're interested in, which can improve performance and reduce unnecessary data transfer.

How to use Projection in MongoDB:

Projection can be applied to queries using the `find()` method. You can specify the fields you want to include (or exclude) in the returned documents.

Syntax for Projection:

js

 Copy code

```
db.collection.find(query, projection)
```




```

new_db> db.students.find({}, {name:1})
[
  { _id: ObjectId("6381f42e93194280a2e6be03"), name: 'Ram' },
  { _id: ObjectId("6381f42e93194280a2e6be04"), name: 'Shyam' },
  { _id: ObjectId("6381f42e93194280a2e6be05"), name: 'Motu' },
  { _id: ObjectId("6381f42e93194280a2e6be06"), name: 'Mohan' },
  { _id: ObjectId("6381f42e93194280a2e6be07"), name: 'Manan' },
  { _id: ObjectId("6381f42e93194280a2e6be08"), name: 'Ankit' },
  { _id: ObjectId("6381f42e93194280a2e6be09"), name: 'Goli' },
  { _id: ObjectId("6381f42e93194280a2e6be0a"), name: 'Pinku' },
  { _id: ObjectId("6381f42e93194280a2e6be0b"), name: 'Mathur' },
  { _id: ObjectId("6381f42e93194280a2e6be0c"), name: 'Ayush' },
  { _id: ObjectId("6381f42e93194280a2e6be0d"), name: 'Amit' },
  { _id: ObjectId("6381f42e93194280a2e6be0e"), name: 'Tom' },
  { _id: ObjectId("6381f42e93194280a2e6be0f"), name: 'Ajay' },
  { _id: ObjectId("6381f42e93194280a2e6be10"), name: 'Tappu' },
  { _id: ObjectId("6381f42e93194280a2e6be11"), name: 'Anuj' },
  { _id: ObjectId("6381f79a93194280a2e6be20"), name: 'Tim' },
  { _id: ObjectId("6381f79a93194280a2e6be21"), name: 'Sam' },
  { _id: ObjectId("6381f79a93194280a2e6be22"), name: 'Patlu' },
  { _id: ObjectId("6381f79a93194280a2e6be23"), name: 'Bilal' },
  { _id: ObjectId("6381f79a93194280a2e6be24"), name: 'Tarak' }
]


```

Here id is by default exclusive that's mean jab tak ham usko 0 nhi karenge mongodb usko bhi include karega


```

mongosh mongodb://127.0.0.1
new_db> db.students.find({}, {name:1, _id:0})
[
  { name: 'Ram' }, { name: 'Shyam' },
  { name: 'Motu' }, { name: 'Mohan' },
  { name: 'Manan' }, { name: 'Ankit' },
  { name: 'Goli' }, { name: 'Pinku' },
  { name: 'Mathur' }, { name: 'Ayush' },
  { name: 'Amit' }, { name: 'Tom' },
  { name: 'Ajay' }, { name: 'Tappu' },
  { name: 'Anuj' }, { name: 'Tim' },
  { name: 'Sam' }, { name: 'Patlu' },
  { name: 'Bilal' }, { name: 'Tarak' }
]
Type "it" for more
new_db> |

```



```
Student> db.students.find({}, {name : 1} , {_id : 0})
[
  { _id: ObjectId('677be0a67607894e434eeb8b'), name: 'Somesh Marathe' },
  { _id: ObjectId('677bf59f7607894e434eeb8d'), name: 'Narayan' },
  { _id: ObjectId('677bf5da7607894e434eeb8e'), name: 'Lakshmi' }
]
Student> db.students.find({}, {name : 1 , _id : 0})
[
  { name: 'Somesh Marathe' },
  { name: 'Narayan' },
  { name: 'Lakshmi' }
]
Student> |
```



Datatype In MongoDB

➡ We can use typeof to check the datatype

```
anazom> typeof db.companyData.findOne().name
string
anazom> typeof db.companyData.findOne().isFunded
boolean
anazom> typeof db.companyData.findOne().funding
number
anazom> typeof db.companyData.findOne().employees
object
anazom> typeof db.companyData.findOne().foundedOn
object
anazom> typeof db.companyData.findOne().foundedOnTimestamp
object
```



```
    }  
  ]  
Employees> db.Employe.insertOne( { name : "bittu" , age : 19 , Status : true  
  , Salary : 1100000 , Identity : { PanCard : 0 , IdCard : 0 } , Joining : {  
  date : new Date() , time : new Timestamp()}} );  
{  
  acknowledged: true,  
  insertedId: ObjectId('677bfff84c0dee663254eeb87')  
}
```

Delete Database In MongoDB

```
mongosh mongod://127.0.0.1:27017
mydb> use mydb
already on db mydb
mydb> show collections
products
mydb> db.dropDatabase()
{ ok: 1, dropped: 'mydb' }
mydb> show collections

mydb> |
```

```
mongosh mongod://127.0.0.1:27017
mydb> show collections
products
products_new
mydb> db.products_new.drop()
true
mydb> show collections
products
mydb> |
```

```
Employees> use Employees;
already on db Employees
Employees> show collections;
Employee
Student
Employees> db.Employee.Drop();
TypeError: db.Employee.Drop is not a function
Employees> db.Employee.drop();
true
Employees> show collections;
Student
Employees> db.dropDatabase();
{ ok: 1, dropped: 'Employees' }
Employees> |
```

Ordered Option In Insert Command

```
Student> use employee;
switched to db employee
employee> db.employee.insertMany([ { _id : "A" , name : "A" , age : 18} , { _id : "B" , name : "B" , age : 19}])
{ acknowledged: true, insertedIds: { '0': 'A', '1': 'B' } }
employee> db.students.find();

employee> db.employee.find();
[ { _id: 'A', name: 'A', age: 18 }, { _id: 'B', name: 'B', age: 19 } ]
employee> db.employee.insertMany([ { _id : "C" , name : "C" , age : 18} , { _id : "B" , name : "D" , age : 19} , { _id : "D" , name : "D" , age : 20}]);
Uncaught:
MongoBulkWriteError: E11000 duplicate key error collection: employee.employe index: _id_ dup key: { _id: "B" }
Result: BulkWriteResult {
  insertedCount: 1,
  matchedCount: 0,
  modifiedCount: 0,
  deletedCount: 0,
  upsertedCount: 0,
  upsertedIds: {},
  insertedIds: { '0': 'C' }
}
Write Errors: [
  WriteError {
    err: {
      index: 1,
      code: 11000,
      errmsg: 'E11000 duplicate key error collection: employee.employe index: _id_ dup key: { _id: "B" }',
      errInfo: undefined,
      op: { _id: 'B', name: 'D', age: 19 }
    }
  }
]
employee> |
```

```
employee> db.employe.find();
[
  { _id: 'A', name: 'A', age: 18 },
  { _id: 'B', name: 'B', age: 19 },
  { _id: 'C', name: 'C', age: 18 }
```

```
employee> db.employe.insertMany([ { _id : "E" , name : "E" , age : 18} , { _id : "B" , name : "D" , age : 19} , { _id : "D" , name : "D" , age : 20} ] , {ordered : false});
```

Uncaught:

MongoBulkWriteError: E11000 duplicate key error collection: employee.employe index: _id_ dup key: { _id: "B" }

Result: BulkWriteResult {

```
  insertedCount: 2,
  matchedCount: 0,
  modifiedCount: 0,
  deletedCount: 0,
  upsertedCount: 0,
  upsertedIds: {},
  insertedIds: { '0': 'E', '2': 'D' }
```

}

Write Errors: [

WriteError {

err: {

index: 1,

code: 11000,

errmsg: 'E11000 duplicate key error collection: employee.employe index: _id_ dup key: { _id: "B" }',

errInfo: undefined,

op: { _id: 'B', name: 'D', age: 19 }

}

}

]

```
employee> db.employe.find();
```

[

```
  { _id: 'A', name: 'A', age: 18 },
```

```
  { _id: 'B', name: 'B', age: 19 },
```

```
  { _id: 'C', name: 'C', age: 18 },
```

```
  { _id: 'E', name: 'E', age: 18 },
```

```
  { _id: 'D', name: 'D', age: 20 }
```

]

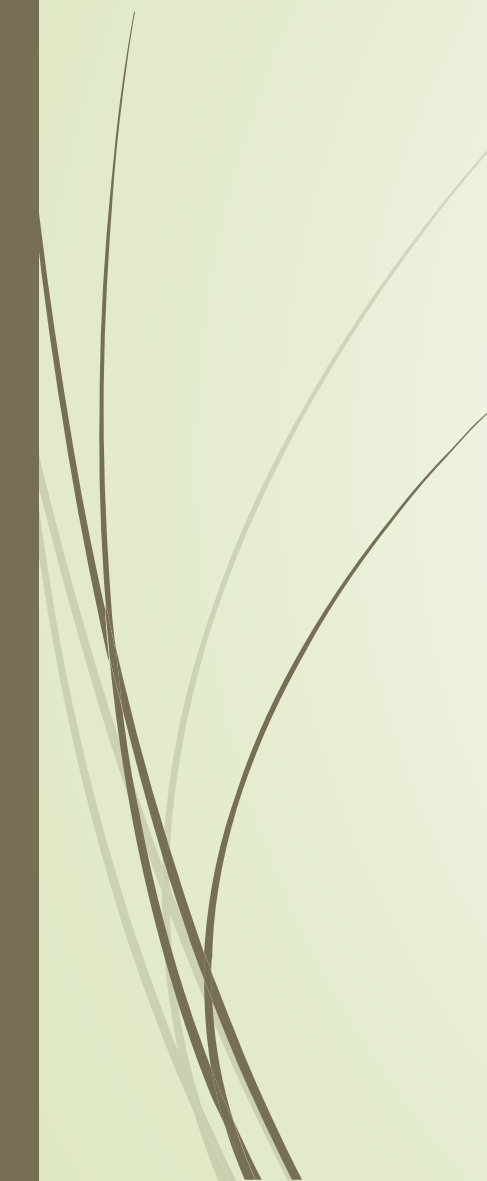

```
employee> |
```




Shema Validation

➡ Create Collection Command

```
Student> db.createCollection("techer");  
{ ok: 1 }  
Student>
```




JS javascript.js > ...

```
1 db.createCollection("Staf",{
2   validator:{
3     $jsonSchema:{
4       required:['name','age','address'],
5       properties:{
6         name:{
7           bsonType : "String" ,
8           description : "must be a number"
9         },
10        age:{
11          bsonType:"number",
12          description:"must be a number"
13        },
14        address:{
15          bsonType: "String" ,
16          description : "must be String"
17        }
18      }
19    }
20  },
21  validationAction: 'error'
22 })
```



```
Student> db.Staf.insertOne({name: "Somesh" , age : 22 , address:123})
Uncaught:
MongoServerError: Document failed validation
Additional information: {
  failingDocumentId: ObjectId('677c9e38e49a6e031d4eeb87'),
  details: {
    operatorName: '$jsonSchema',
    schemaRulesNotSatisfied: [
      {
        operatorName: 'properties',
        propertiesNotSatisfied: [
          {
            propertyName: 'address',
            description: 'must be String',
            details: [
              {
                operatorName: 'bsonType',
                specifiedAs: { bsonType: 'string' },
                reason: 'type did not match',
                consideredValue: 123,
                consideredType: 'int'
              }
            ]
          }
        ]
      }
    ]
  }
}
```




```
mongosh mongodb://127.0.0.1
Student> db.Staf.find();
[
  {
    _id: ObjectId('677c9e20e49a6e031d4eeb86'),
    name: 'Somesh',
    age: 22,
    address: '123'
  },
  {
    _id: ObjectId('677c9efbe49a6e031d4eeb88'),
    name: 'Somesh',
    age: 22,
    address: 'Silicon Indore'
  }
]
Student> db.Staf.insertOne({name : "Somesh" , age: 22 , address : "Silicon Indore" , number : 8234021112 })
{
  acknowledged: true,
  insertedId: ObjectId('677c9f37e49a6e031d4eeb89')
}
Student> db.Staf.find();
[
  {
    _id: ObjectId('677c9e20e49a6e031d4eeb86'),
    name: 'Somesh',
    age: 22,
    address: '123'
  },
  {
    _id: ObjectId('677c9efbe49a6e031d4eeb88'),
    name: 'Somesh',
    age: 22,
    address: 'Silicon Indore'
  },
  {
    _id: ObjectId('677c9f37e49a6e031d4eeb89'),
    name: 'Somesh',
    age: 22,
    address: 'Silicon Indore',
    number: 8234021112
  }
]
```


Modification in Schema

```
JS temp.js
C: > Users > vipul > Documents > JS temp.js > validator > $jsonSchema > properties > author > items

20 db.runCommand({
21   collMod: 'nonfiction',
22   validator: {
23     $jsonSchema: {
24       required: ['name', 'price', 'author'],
25       properties: {
26         name: {
27           bsonType: 'string',
28           description: 'must be a string and required'
29         },
30         price: {
31           bsonType: 'number',
32           description: 'must be a number and required'
33         },
34         author: {
35           bsonType: 'array',
36           description: 'must be an array and is required',
37           items: [
38             {
39               bsonType: 'object',
40               required: ['name', 'email'],
41             }
42           ]
43         }
44       }
45     }
46   }
47 })
```



```
JS temp.js
C:\Users> vipul > Documents > JS temp.js > ...
20 db.runCommand({
21   collMod: 'nonfiction',
22   validator: {
23     $jsonSchema: {
24       required: ['name', 'price', 'authors'],
25       properties: {
26         name: {
27           bsonType: 'string',
28           description: 'must be a string and required'
29         },
30         price: {
31           bsonType: 'number',
32           description: 'must be a number and required'
33         },
34         authors: {
35           bsonType: 'array',
36           description: 'must be an array and is required',
37           items: {
38             bsonType: 'object',
39             required: ['name', 'email'],
40             properties: {
41               name: {
42                 bsonType: 'string'
43               },
44               email: {
45                 bsonType: 'string'
46               }
47             }
48           }
49         }
50       }
51     }
52   }
53 })
```





```
Student> db.runCommand({
...
...   collMod : 'techer' ,
...   validator:{
...     $jsonSchema:{
...       required : ["name" , "age" , "department"],
...       properties :{
...
...         name :{
...           bsonType : "string",
...           description:"name must be string"
...         },
...         age :{
...           bsonType : "number",
...           description:"name must be string"
...         },
...       },
...       departement :{
...         bsonType : "array",
...         description:"name must be array",
...
...         items:{
...           bsonType : 'object' ,
...           required:['name', 'courseCode'],
...           properties:{
...             name:{
...               bsonType : "string" ,
...               description : "must be a number"
...             },
...             courseCode:{
...               bsonType:"number",
...               description:"must be a number"
...             }
...           }
...         }
...       }
...     }
...   }
... }
```

Write concern



In MongoDB, **Write Concern** refers to the level of acknowledgment the application requires from MongoDB when performing write operations. It determines how strictly MongoDB ensures the data is written to the database before confirming success.

Write Concern Levels

1. `w: 0` - Unacknowledged:

- The write operation is sent to the server, but no acknowledgment is returned.
- No guarantee that the operation succeeded.
- Fast but risky.

2. `w: 1` - Acknowledged:

- MongoDB confirms the write operation after it has been written to the primary.
- Default setting.

3. `w: "majority"` - Majority:

- The operation is acknowledged only after a majority of the nodes in the replica set confirm the write.
- Provides a balance between reliability and performance.

4. `w: <number>` - Custom Level:

- Specifies the number of nodes (including the primary) that must acknowledge the write before it's considered successful.

5. `journal: true/false` - Journaling:

- Ensures that the write operation is committed to the journal on disk.

- Default setting.

3. `w: "majority"` - Majority:

- The operation is acknowledged only after a majority of the nodes in the replica set confirm the write.
- Provides a balance between reliability and performance.

4. `w: <number>` - Custom Level:


- Specifies the number of nodes (including the primary) that must acknowledge the write before it's considered successful.

5. `journal: true/false` - Journaling:


- Ensures that the write operation is committed to the journal on disk.
- Example: `{ w: 1, journal: true }`

6. `wtimeout` - Write Timeout:

- Specifies the maximum time (in milliseconds) to wait for the write concern to be satisfied.
- Example: `{ w: "majority", wtimeout: 5000 }`




```
products> db.books.insertOne({name:"A", price:1})
{
  acknowledged: true,
  insertedId: ObjectId("638ebeae26d2bf648ec28e91")
}
products> db.books.insertOne({name:"B", price:2},{writeConcern:{w:0}})
{
  acknowledged: false,
  insertedId: ObjectId("638ebedb26d2bf648ec28e92")
}
products>
```





Atomicity in Mongo Db

➤ **Atomicity in MongoDB**

- Atomicity refers to the concept where a series of database operations are treated as a single, indivisible unit. In MongoDB, atomicity ensures that either all operations in a transaction are successfully completed, or none of them are applied.
- 



Importjson in *Mongo* DB



Comparison Operator


```
Student> db.students.find({age : 23})
[
  {
    _id: ObjectId('677be0a67607894e434eeb8b'),
    name: 'Somesh Marathe',
    age: 23,
    idCard: { hasPanCard: false, hasIdCard: false },
    hobbies: [ 'acting', 'dancing' ]
  }
]
Student> db.students.find({age : {$eq : 5}})
Student> db.students.find({age : {$eq : 23}})
[
  {
    _id: ObjectId('677be0a67607894e434eeb8b'),
    name: 'Somesh Marathe',
    age: 23,
    idCard: { hasPanCard: false, hasIdCard: false },
    hobbies: [ 'acting', 'dancing' ]
  }
]
Student> db.students.find({age : {$ne : 23}})
[
  {
    _id: ObjectId('677bf59f7607894e434eeb8d'),
    name: 'Narayan',
    age: 24,
    hobbies: [ 'acting', 'dancing' ]
  },
  {
    _id: ObjectId('677bf5da7607894e434eeb8e'),
    name: 'Lakshmi',
    age: 21,
    hobbies: [ 'acting', 'dancing' ]
  }
]
Student> |
```

```
Student> db.students.find({age : {$gt : 23}})
[
  {
    _id: ObjectId('677bf59f7607894e434eeb8d'),
    name: 'Narayan',
    age: 24,
    hobbies: [ 'acting', 'dancing' ]
  }
]
Student> db.students.find({age : {$lt : 23}})
[
  {
    _id: ObjectId('677bf5da7607894e434eeb8e'),
    name: 'Lakshmi',
    age: 21,
    hobbies: [ 'acting', 'dancing' ]
  }
]
Student> db.students.find({age : {$in : [23,24]}})
[
  {
    _id: ObjectId('677be0a67607894e434eeb8b'),
    name: 'Somesh Marathe',
    age: 23,
    idCard: { hasPanCard: false, hasIdCard: false },
    hobbies: [ 'acting', 'dancing' ]
  },
  {
    _id: ObjectId('677bf59f7607894e434eeb8d'),
    name: 'Narayan',
    age: 24,
    hobbies: [ 'acting', 'dancing' ]
  }
]
Student>
```

```
Student> db.students.find({age : {$in : [23,24]}})
[
  {
    _id: ObjectId('677be0a67607894e434eeb8b'),
    name: 'Somesh Marathe',
    age: 23,
    idCard: { hasPanCard: false, hasIdCard: false },
    hobbies: [ 'acting', 'dancing' ]
  },
  {
    _id: ObjectId('677bf59f7607894e434eeb8d'),
    name: 'Narayan',
    age: 24,
    hobbies: [ 'acting', 'dancing' ]
  }
]
Student> db.students.find({age : {$nin : [23,24]}})
[
  {
    _id: ObjectId('677bf5da7607894e434eeb8e'),
    name: 'Lakshmi',
    age: 21,
    hobbies: [ 'acting', 'dancing' ]
  }
]
Student>
```

Logical Operator

```
mongosh mongodb://127.0.0.1:27027/college
college> db.students.find({ $or: [ {age:{$lte:10}} , {age:{$gte:12}} ]})
[
  {
    _id: ObjectId("6396a083f3b30b3e904d794f"),
    name: 'Sita',
    age: 5,
    Hobbies: [ 'Walk', 'Cricket' ],
    identity: { hasPanCard: false, hasAdhaarCard: true }
  },
  {
    _id: ObjectId("6396a083f3b30b3e904d7953"),
    name: 'Ram',
    age: 10,
    Hobbies: [ 'Walk', 'Cricket' ],
    identity: { hasPanCard: false, hasAdhaarCard: true }
  },
  {
    _id: ObjectId("6396a083f3b30b3e904d7954"),
    name: 'Geeta',
    age: 12,
    Hobbies: [ 'Gaming', 'Cooking' ],
    identity: { hasPanCard: false, hasAdhaarCard: true }
  }
]
college>
```



```
Student> db.students.find({$nor : [ {age : {$lte: 11}} , {age : {$gte : 25}}
  ]});
[
  {
    _id: ObjectId('677be0a67607894e434eeb8b'),
    name: 'Somesh Marathe',
    age: 23,
    idCard: { hasPanCard: false, hasIdCard: false },
    hobbies: [ 'acting', 'dancing' ]
  },
  {
    _id: ObjectId('677bf59f7607894e434eeb8d'),
    name: 'Narayan',
    age: 24,
    hobbies: [ 'acting', 'dancing' ]
  },
  {
    _id: ObjectId('677bf5da7607894e434eeb8e'),
    name: 'Lakshmi',
    age: 21,
    hobbies: [ 'acting', 'dancing' ]
  }
]
Student> |
```

Nor is used when we want both condition are false then it executes

mongosh mongodb://127.0.0.1

```
college> db.students.find({ $and:[ { age:{$lt:11} }, { Hobbies:'Walk' } ] })
[
  {
    _id: ObjectId("6396a083f3b30b3e904d794f"),
    name: 'Sita',
    age: 5,
    Hobbies: [ 'Walk', 'Cricket' ],
    identity: { hasPanCard: false, hasAdhaarCard: true }
  },
  {
    _id: ObjectId("6396a083f3b30b3e904d7953"),
    name: 'Ram',
    age: 10,
    Hobbies: [ 'Walk', 'Cricket' ],
    identity: { hasPanCard: false, hasAdhaarCard: true }
  }
]
college> |
```



```
Student> db.students.find({age:{$lt : 22}, age:{$gt :20}})
[
  {
    _id: ObjectId('677be0a67607894e434eeb8b'),
    name: 'Somesh Marathe',
    age: 23,
    idCard: { hasPanCard: false, hasIdCard: false },
    hobbies: [ 'acting', 'dancing' ]
  },
  {
    _id: ObjectId('677bf59f7607894e434eeb8d'),
    name: 'Narayan',
    age: 24,
    hobbies: [ 'acting', 'dancing' ]
  },
  {
    _id: ObjectId('677bf5da7607894e434eeb8e'),
    name: 'Lakshmi',
    age: 21,
    hobbies: [ 'acting', 'dancing' ]
  }
]
Student> |
```

In this case mongo db ignore the first condition and give result based on last condition


```
Student> db.students.find({$and : [ {age : {$gt : 20}} , {age : {$lt : 25}}]
});
[
  {
    _id: ObjectId('677be0a67607894e434eeb8b'),
    name: 'Somesh Marathe',
    age: 23,
    idCard: { hasPanCard: false, hasIdCard: false },
    hobbies: [ 'acting', 'dancing' ]
  },
  {
    _id: ObjectId('677bf59f7607894e434eeb8d'),
    name: 'Narayan',
    age: 24,
    hobbies: [ 'acting', 'dancing' ]
  },
  {
    _id: ObjectId('677bf5da7607894e434eeb8e'),
    name: 'Lakshmi',
    age: 21,
    hobbies: [ 'acting', 'dancing' ]
  }
]
Student> db.students.find({$and : [ {age : {$gt : 22}} , {age : {$lt : 25}}]
});
[
  {
    _id: ObjectId('677be0a67607894e434eeb8b'),
    name: 'Somesh Marathe',
    age: 23,
    idCard: { hasPanCard: false, hasIdCard: false },
    hobbies: [ 'acting', 'dancing' ]
  },
  {
    _id: ObjectId('677bf59f7607894e434eeb8d'),
    name: 'Narayan',
    age: 24,
    hobbies: [ 'acting', 'dancing' ]
  }
]
Student> |
```



 mongosh mongod://127.0.0.1:27021

```
college> db.students.find({ $and: [ {age: { $not: { $lt: 11 } } }, { age: { $not: { $gt: 5 } } } ] })
```

```
college>
```



\$exists and \$type

```
Student> db.students.find({ number: {$exists : true}});  
[  
  {  
    _id: ObjectId('677bf5da7607894e434eeb8e'),  
    name: 'Lakshmi',  
    age: 21,  
    hobbies: [ 'cooking' ],  
    number: 823402  
  }  
]  
Student> |
```

```
Student> db.students.find({haspancard : {exists : true , $type : 8}});  
Student> db.students.find({haspancard : {exists : true , $type : 'bool'}});  
Student>
```

