MongoDB Tutorial

MongoDB Tutorial

MongoDB tutorial provides basic and advanced concepts of SQL. Our MongoDB tutorial is designed for beginners and professionals.

MongoDB is a No SQL database. It is an open-source, cross-platform, document-oriented database written in C++.

Our MongoDB tutorial includes all topics of MongoDB database such as insert documents, update documents, delete documents, query documents, projection, sort() and limit() methods, create a collection, drop collection, etc. There are also given MongoDB interview questions to help you better understand the MongoDB database.

What is MongoDB

MongoDB is an open-source document database that provides high performance, high availability, and automatic scaling.

In simple words, you can say that - Mongo DB is a document-oriented database. It is an open source product, developed and supported by a company named 10gen.

MongoDB is available under General Public license for free, and it is also available under Commercial license from the manufacturer.

The manufacturing company 10gen has defined MongoDB as:

"MongoDB is a scalable, open source, high performance, document-oriented database." - 10gen

MongoDB was designed to work with commodity servers. Now it is used by the company of all sizes, across all industry.

History of MongoDB

The initial development of MongoDB began in 2007 when the company was building a platform as a service similar to window azure.

Window azure is a cloud computing platform and infrastructure, created by Microsoft, to build, deploy and manage applications and service through a global network.

MongoDB was developed by a NewYork based organization named 10gen which is now known as MongoDB Inc. It was initially developed as a PAAS (Platform as a Service). Later in 2009, it is introduced in the market as an open source database server that was maintained and supported by MongoDB Inc.

The first ready production of MongoDB has been considered from version 1.4 which was released in March 2010.

MongoDB2.4.9 was the latest and stable version which was released on January 10, 2014.

Purpose of Building MongoDB

It may be a very genuine question that - "what was the need of MongoDB although there were many databases in action?"

There is a simple answer:

All the modern applications require big data, fast features development, flexible deployment, and the older database systems not competent enough, so the MongoDB was needed.

The primary purpose of building MongoDB is:

- Scalability
- Performance
- High Availability
- Scaling from single server deployments to large, complex multi-site architectures.
- Key points of MongoDB
- Develop Faster
- Deploy Easier
- Scale Bigger

First of all, we should know what is document oriented database?

Example of Document-Oriented Database

MongoDB is a document-oriented database. It is a key feature of MongoDB. It offers a document-oriented storage. It is very simple you can program it easily.

MongoDB stores data as documents, so it is known as document-oriented database.

1. FirstName = "John",

2. Address = "Detroit",
3. Spouse = [{**Name**: "Angela"}].

4. FirstName ="John",
5. Address = "Wick"

There are two different documents (separated by ".").

Storing data in this manner is called as document-oriented database.

Mongo DB falls into a class of databases that calls Document Oriented Databases. There is also a broad category of database known as No SQL Databases.

Features of MongoDB

These are some important features of MongoDB:

1. Support ad hoc queries

In MongoDB, you can search by field, range query and it also supports regular expression searches.

2. Indexing

You can index any field in a document.

3. Replication

MongoDB supports Master Slave replication.

A master can perform Reads and Writes and a Slave copies data from the master and can only be used for reads or back up (not writes)

4. Duplication of data

MongoDB can run over multiple servers. The data is duplicated to keep the system up and also keep its running condition in case of hardware failure.

5. Load balancing

It has an automatic load balancing configuration because of data placed in shards.

Provides high performance.

- JSON data model with dynamic schemas
- Auto-sharding for horizontal scalability
- Built in replication for high availability
- Now a day many companies using MongoDB to create new types of applications, improve performance and availability.

Prerequisite

Before learning MongoDB, you must have the basic knowledge of SQL and OOPs.

A MongoDB Document

Records in a MongoDB database are called documents, and the field values may include numbers, strings, booleans, arrays, or even nested documents.

Example Document

```
{
        title: "Post Title 1",
        body: "Body of post.",
        category: "News",
        likes: 1,
        tags: ["news", "events"],
        date: Date()
}
```

How to set up the MongoDB environment

1. C:\Program Files\MongoDB\bin\mongod.exe

This will start the mongoDB database process. If you get a message "waiting for connection" in the console output, it indicates that the mongodb.exe process is running successfully.

For example:

When you connect to the MongoDB through the mongo.exe shell, you should follow these steps:

1. Open another command prompt.
2. At the time of connecting, specify the data directory if necessary.

> ***Note: If you use the default data directory while MongoDB installation, there is no need to specify the data directory.***

For example:

1. C:\mongodb\bin\mongo.exe

Install MongoDB Shell (mongosh)

There are many ways to connect to your MongoDB database.

We will start by using the MongoDB Shell, `mongosh`.

Use the official instructions to install `mongosh` on your operating system.

To verify that it has been installed properly, open your terminal and type:

```
mongosh --version
```

You should see that the latest verion is installed.

MongoDB supports many datatypes. Some of them are –

- **String** – This is the most commonly used datatype to store the data. String in MongoDB must be UTF-8 valid.
- **Integer** – This type is used to store a numerical value. Integer can be 32 bit or 64 bit depending upon your server.
- **Boolean** – This type is used to store a boolean (true/ false) value.
- **Double** – This type is used to store floating point values.
- **Min/ Max keys** – This type is used to compare a value against the lowest and highest BSON elements.
- **Arrays** – This type is used to store arrays or list or multiple values into one key.
- **Timestamp** – ctimestamp. This can be handy for recording when a document has been modified or added.
- **Object** – This datatype is used for embedded documents.
- **Null** – This type is used to store a Null value.
- **Symbol** – This datatype is used identically to a string; however, it's generally reserved for languages that use a specific symbol type.

- **Date** – This datatype is used to store the current date or time in UNIX time format. You can specify your own date time by creating object of Date and passing day, month, year into it.
- **Object ID** – This datatype is used to store the document's ID.
- **Binary data** – This datatype is used to store binary data.
- **Code** – This datatype is used to store JavaScript code into the document.
- **Regular expression** – This datatype is used to store regular expression.

MongoDB `mongosh` Create Collection

Create Collection using `mongosh`

- There are 2 ways to create a collection.

Method 1

- You can create a collection using the `createCollection()` database method.

Example

```
db.createCollection("posts")
```

The createCollection() Method

- MongoDB **db.createCollection(name, options)** is used to create collection.
- Syntax
- Basic syntax of **createCollection()** command is as follows –

```
db.createCollection(name, options)
```

- In the command, **name** is name of collection to be created. **Options** is a document and is used to specify configuration of collection.

Parameter	Type	Description
Name	String	Name of the collection to be created
Options	Document	(Optional) Specify options about memory size and indexing

Method 2

- You can also create a collection during the `insert` process.

Example

- We are here assuming `object` is a valid JavaScript object containing post data:

```
db.posts.insertOne(object)
```

- This will create the "posts" collection if it does not already exist.
- **Remember:** In MongoDB, a collection is not actually created until it gets content!

MongoDB mongosh Insert

Insert Documents

- There are 2 methods to insert documents into a MongoDB database.

insertOne()

- To insert a single document, use the `insertOne()` method.
- This method inserts a single object into the database.
- **Note:** When typing in the shell, after opening an object with curly braces "{" you can press enter to start a new line in the editor without executing the command. The command will execute when you press enter after closing the braces.

Example

```
db.posts.insertOne({
  title: "Post Title 1",
  body: "Body of post.",
  category: "News",
  likes: 1,
  tags: ["news", "events"],
  date: Date()
})
```

- **Note:** If you try to insert documents into a collection that does not exist, MongoDB will create the collection automatically.

insertMany()

- To insert multiple documents at once, use the `insertMany()` method.
- This method inserts an array of objects into the database.

Example

```
db.posts.insertMany([
  {
    title: "Post Title 2",
    body: "Body of post.",
    category: "Event",
    likes: 2,
    tags: ["news", "events"],
    date: Date()
  },
  {
    title: "Post Title 3",
    body: "Body of post.",
    category: "Technology",
    likes: 3,
    tags: ["news", "events"],
    date: Date()
```

```
    },
    {
      title: "Post Title 4",
      body: "Body of post.",
      category: "Event",
      likes: 4,
      tags: ["news", "events"],
      date: Date()
    }
  ])
```

The dropDatabase() Method

MongoDB **db.dropDatabase()** command is used to drop a existing database.

Syntax

Basic syntax of **dropDatabase()** command is as follows –

```
db.dropDatabase()
```

This will delete the selected database. If you have not selected any database, then it will delete default 'test' database.

Example

First, check the list of available databases by using the command, **show dbs**.

```
>show dbs
local     0.78125GB
mydb      0.23012GB
test      0.23012GB
>
```

If you want to delete new database **<mydb>**, then **dropDatabase()** command would be as follows –

```
>use mydb
switched to db mydb
>db.dropDatabase()
>{ "dropped" : "mydb", "ok" : 1 }
>
```

Now check list of databases.

```
>show dbs
local     0.78125GB
test      0.23012GB
>
```

MongoDB `mongosh` Find

Find Data

There are 2 methods to find and select data from a MongoDB collection, `find()` and `findOne()`.

`find()`

To select data from a collection in MongoDB, we can use the `find()` method.

This method accepts a query object. If left empty, all documents will be returned.

Example

```
db.posts.find()
```

`findOne()`

To select only one document, we can use the `findOne()` method.

This method accepts a query object. If left empty, it will return the first document it finds.

Note: This method only returns the first match it finds.

Example

```
db.posts.findOne()
```

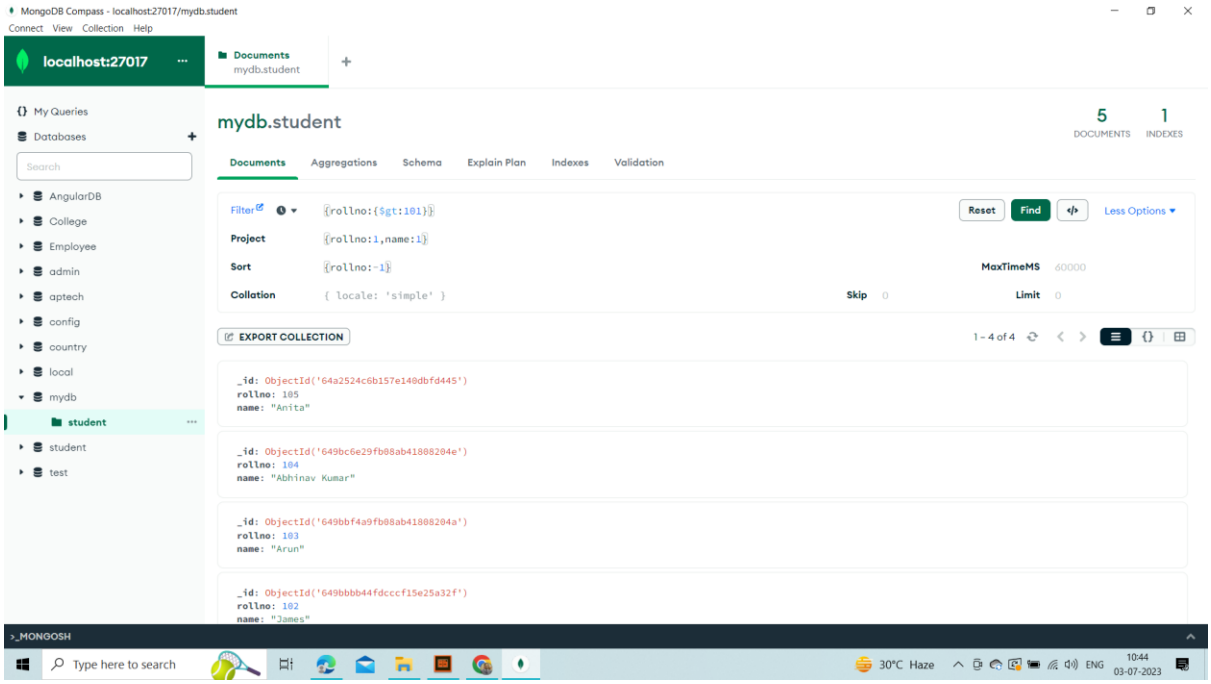
Querying Data

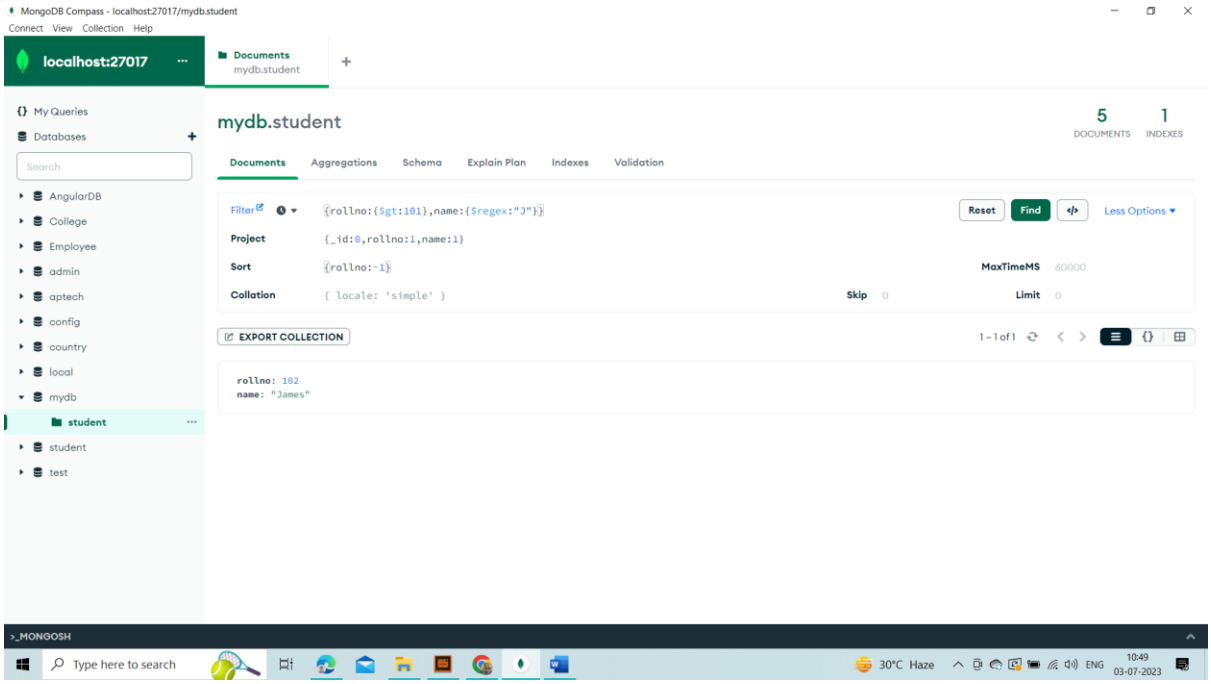
To query, or filter, data we can include a query in our `find()` or `findOne()` methods.

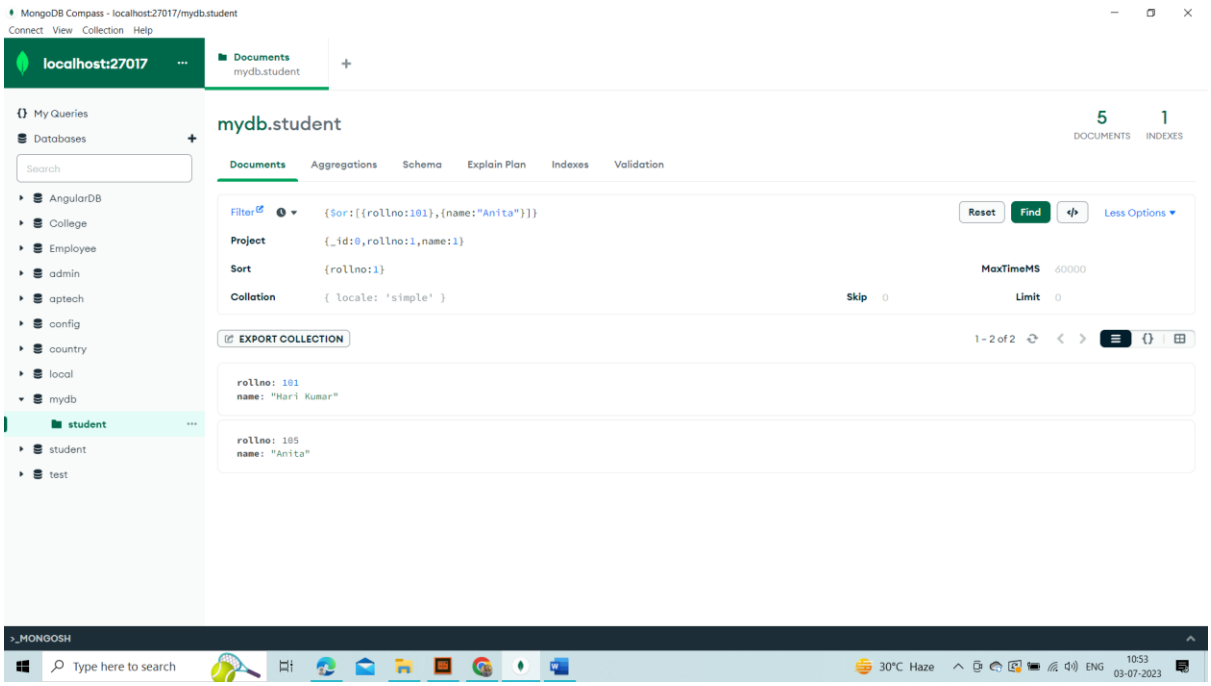
Example

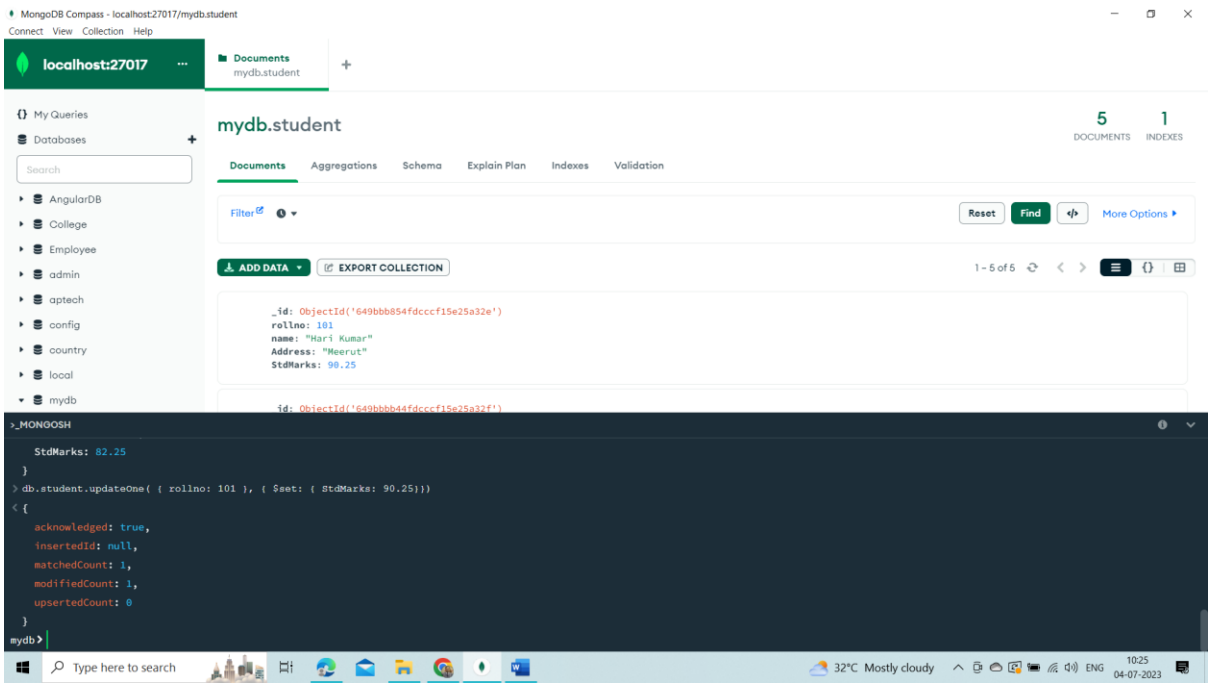
```
db.posts.find( {category: "News"} )
```

Projection


MongoDB Compass - localhost:27017/mydb.student
Connect View Collection Help
localhost:27017
Documents
mydb.student
My Queries
Databases
Search
AngularDB
College
Employee
admin
aptech
config
country
local
mydb
student
student
test
mydb.student
5
DOCUMENTS
1
INDEXES
Documents
Aggregations
Schema
Explain Plan
Indexes
Validation
Filter
{rollno:{$gt:101}}
Reset
Find
Less Options
Project
{rollno:1,name:1}
Sort
{rollno:-1}
MaxTimeMS
60000
Collation
{ locale: 'simple' }
Skip
0
Limit
0
EXPORT COLLECTION
1 - 4 of 4
_id: ObjectId('64a2524c6b157e140dbfd445')
rollno: 105
name: "Anita"
_id: ObjectId('649bc6e29fb08ab41808204e')
rollno: 104
name: "Abhinav Kumar"
_id: ObjectId('649bbf4a9fb08ab41808204a')
rollno: 103
name: "Arun"
_id: ObjectId('649bbbb44fdcccf15e25a32f')
rollno: 102
name: "James"
>_MONGOSH
Type here to search
30°C Haze
ENG
10:44
03-07-2023


MongoDB Compass - localhost:27017/mydb.student
Connect View Collection Help
localhost:27017
Documents
mydb.student
My Queries
Databases
Search
AngularDB
College
Employee
admin
aptech
config
country
local
mydb
student
student
test
mydb.student
5
DOCUMENTS
1
INDEXES
Documents
Aggregations
Schema
Explain Plan
Indexes
Validation
Filter
{rollno:{$gt:101},name:{$regex:"J"}}
Reset
Find
Less Options
Project
{_id:0,rollno:1,name:1}
Sort
{rollno:-1}
MaxTimeMS
60000
Collation
{ locale: 'simple' }
Skip
0
Limit
0
EXPORT COLLECTION
1 - 1 of 1
rollno: 102
name: "James"
>_MONGOSH
Type here to search
30°C Haze
ENG
10:49
03-07-2023





Both find methods accept a second parameter called `projection`.

This parameter is an `object` that describes which fields to include in the results.

Note: This parameter is optional. If omitted, all fields will be included in the results.

Example

This example will only display the `title` and `date` fields in the results.

```
db.posts.find({}, {title: 1, date: 1})
```

Notice that the `_id` field is also included. This field is always included unless specifically excluded.

We use a `1` to include a field and `0` to exclude a field.

Example

This time, let's exclude the `_id` field.

```
db.posts.find({}, {_id: 0, title: 1, date: 1})
```

Note: You cannot use both 0 and 1 in the same object. The only exception is the `_id` field. You should either specify the fields you would like to include or the fields you would like to exclude.

Let's exclude the date category field. All other fields will be included in the results.

Example

```
db.posts.find({}, {category: 0})
```

We will get an error if we try to specify both 0 and 1 in the same object.

Example

```
db.posts.find({}, {title: 1, date: 0})
```

Search Query

```
Employee> db.Empinfo.find()
[
  {
    _id: ObjectId("63c0df6c1c3d68b027325a32"),
    empid: 101,
    title: 'IT Company Employee',
    salary: 41000
  },
  {
    _id: ObjectId("63c0e07a1c3d68b027325a34"),
    empid: 102,
    title: 'Sales Company Employee',
    salary: 20000
  }
]
Employee>
```

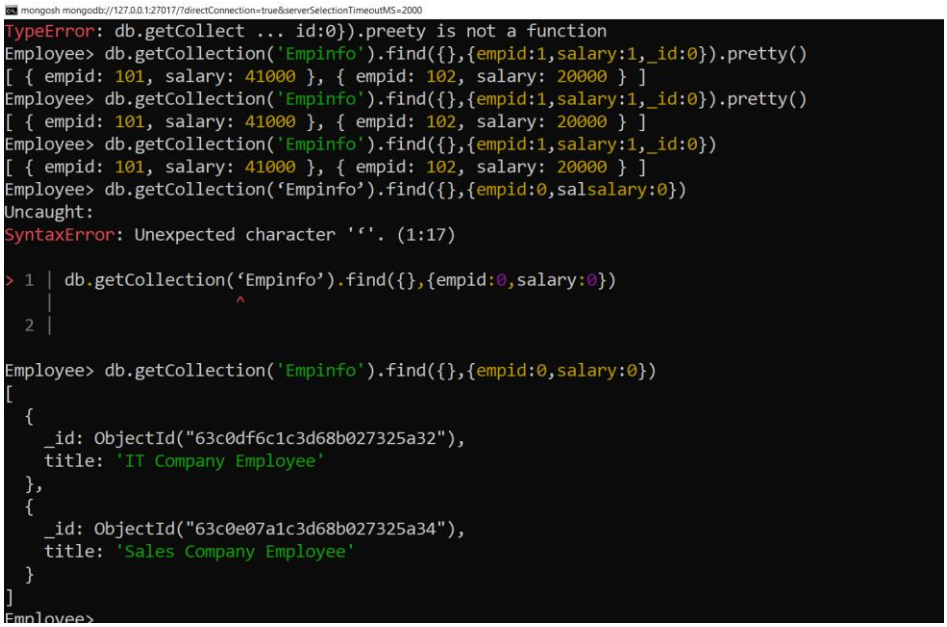
Search db.getCollection(‘Empinfo’).find({})

Search db.getCollection(‘Empinfo’).find({},{empid:1,salary:1})

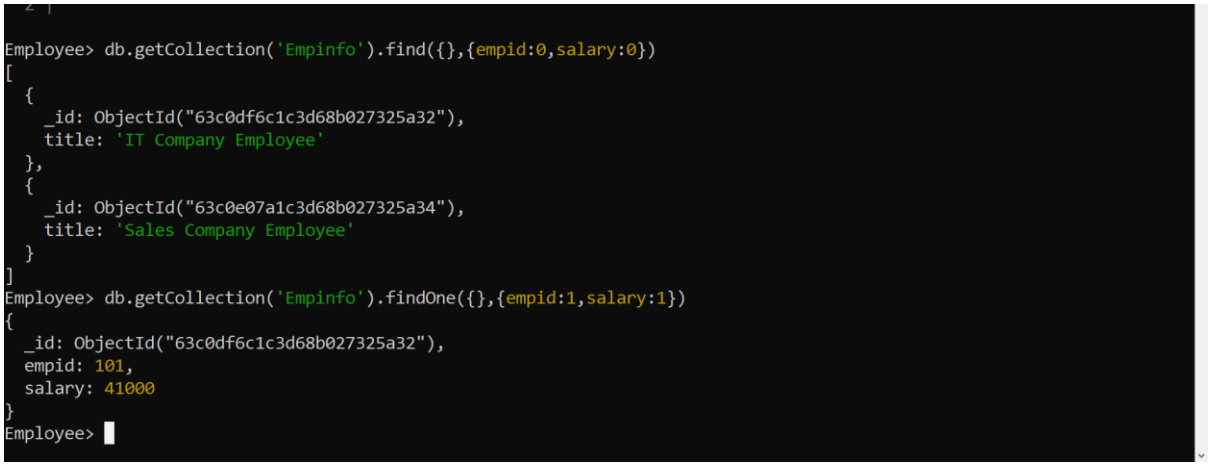
Search db.getCollection(‘Empinfo’).find({},{empid:1,salary:1,_id:0})

Zero means expect given field name

Search db.getCollection(‘Empinfo’).find({},{empid:0,salary:0})



db.getCollection(‘Empinfo’).findOne({},{empid:1,salary:1})



Use 1- true, o- False for sorting in ascending

Logical Condition operator

db.getCollection('Empinfo').find({salary:{$lt:10000}});

Regular expression

db.getCollection('Empinfo').find({name:{$regex:"p"}});

Logical Condition

db.getCollection('Empinfo').find({$and:[{empid:101},{salary:20000}]});

db.getCollection('Empinfo').find({$or:[{empid:101},{salary:20000}]});

salary:0 use for salary hide

db.getCollection('Empinfo').find({},{salary:0}]});

MongoDB mongosh Update

Update Document

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()

The `updateOne()` method will update the first document that is found matching the provided query.

Let's see what the "like" count for the post with the title of "Post Title 1":

Example

```
db.posts.find( { title: "Post Title 1" } )
```

Now let's update the "likes" on this post to 2. To do this, we need to use the `$set` operator.

Example

```
db.posts.updateOne( { title: "Post Title 1" }, { $set: { likes: 2 } } )
```

Check the document again and you'll see that the "like" have been updated.

Example

```
db.posts.find( { title: "Post Title 1" } )
```

Insert if not found

If you would like to insert the document if it is not found, you can use the `upsert` option.

Example

Update the document, but if not found insert it:

```
db.posts.updateOne(

  { title: "Post Title 5" },

  {
```

```
    $set:
      {
        title: "Post Title 5",
        body: "Body of post.",
        category: "Event",
        likes: 5,
        tags: ["news", "events"],
        date: Date()
      }
  },
  { upsert: true }
)
```

updateMany()

The `updateMany()` method will update all documents that match the provided query.

Example

Update `likes` on all documents by 1. For this we will use the `$inc` (increment) operator:

```
db.posts.updateMany({}, { $inc: { likes: 1 } })
```

MongoDB mongosh Delete

Delete Documents

We can delete documents by using the methods `deleteOne()` or `deleteMany()`.

These methods accept a query object. The matching documents will be deleted.

deleteOne()

The `deleteOne()` method will delete the first document that matches the query provided.

Example

```
db.posts.deleteOne({ title: "Post Title 5" })
```

deleteMany()

The `deleteMany()` method will delete all documents that match the query provided.

Example

```
db.posts.deleteMany({ category: "Technology" })
```

The drop() Method

MongoDB's **db.collection.drop()** is used to drop a collection from the database.

Syntax

Basic syntax of **drop()** command is as follows –

```
db.COLLECTION_NAME.drop()
```

Example

First, check the available collections into your database **mydb**.

```
>use mydb
switched to db mydb
>show collections
mycol
mycollection
system.indexes
tutorialspoint
>
```

Now drop the collection with the name **mycollection**.

```
>db.mycollection.drop()
true
>
```

Again check the list of collections into database.

```
>show collections
mycol
system.indexes
```

```
tutorialspoint
>
```

The pretty() Method

To display the results in a formatted way, you can use pretty() method.

Syntax

```
>db.COLLECTION_NAME.find().pretty()
```

RDBMS Where Clause Equivalents in MongoDB

To query the document on the basis of some condition, you can use following operations.

Operation	Syntax	Example	RDBMS Equivalent
Equality	{<key>:{$eg;<value>}}	db.mycol.find({"by":"tutorials point"}).pretty()	where by = 'tutorials point'
Less Than	{<key>:{$lt:<value>}}	db.mycol.find({"likes":{$lt:50}}).pretty()	where likes < 50
Less Than Equals	{<key>:{$lte:<value>}}	db.mycol.find({"likes":{$lte:50}}).pretty()	where likes <= 50
Greater Than	{<key>:{$gt:<value>}}	db.mycol.find({"likes":{$gt:50}}).pretty()	where likes > 50
Greater Than Equals	{<key>:{$gte:<value>}}	db.mycol.find({"likes":{$gte:50}}).pretty()	where likes >= 50
Not Equals	{<key>:{$ne:<value>}}	db.mycol.find({"likes":{$ne:50}}).pretty()	where likes != 50
Values in an array	{<key>:{$in:[<value1>, <value2>,.....<valueN>]}}	db.mycol.find({"name":{$in:["Raj", "Ram", "Raghu"]}}).pretty()	Where name matches any of the value in

			:["Raj", "Ram", "Raghu"]
Values not in an array	{<key>:{$nin:<value>}}	db.mycol.find({"name":{$nin:["Ramu", "Raghav"]}}).pretty()	Where name values is not in the array :["Ramu", "Raghav"] or, doesn't exist at all

MongoDB Query Operators

MongoDB Query Operators

There are many query operators that can be used to compare and reference document fields.

Comparison

The following operators can be used in queries to compare values:

- `$eq`: Values are equal
- `$ne`: Values are not equal
- `$gt`: Value is greater than another value
- `$gte`: Value is greater than or equal to another value
- `$lt`: Value is less than another value
- `$lte`: Value is less than or equal to another value
- `$in`: Value is matched within an array

Logical

The following operators can logically compare multiple queries.

- `$and`: Returns documents where both queries match
- `$or`: Returns documents where either query matches
- `$nor`: Returns documents where both queries fail to match
- `$not`: Returns documents where the query does not match

Evaluation

The following operators assist in evaluating documents.

- `$regex`: Allows the use of regular expressions when evaluating field values
- `$text`: Performs a text search
- `$where`: Uses a JavaScript expression to match documents

MongoDB Update Operators

MongoDB Update Operators

There are many update operators that can be used during document updates.

Fields

The following operators can be used to update fields:

- `$currentDate`: Sets the field value to the current date
- `$inc`: Increments the field value
- `$rename`: Renames the field
- `$set`: Sets the value of a field
- `$unset`: Removes the field from the document

Array

The following operators assist with updating arrays.

- `$addToSet`: Adds distinct elements to an array
- `$pop`: Removes the first or last element of an array
- `$pull`: Removes all elements from an array that match the query
- `$push`: Adds an element to an array

MongoDB Aggregation Pipelines

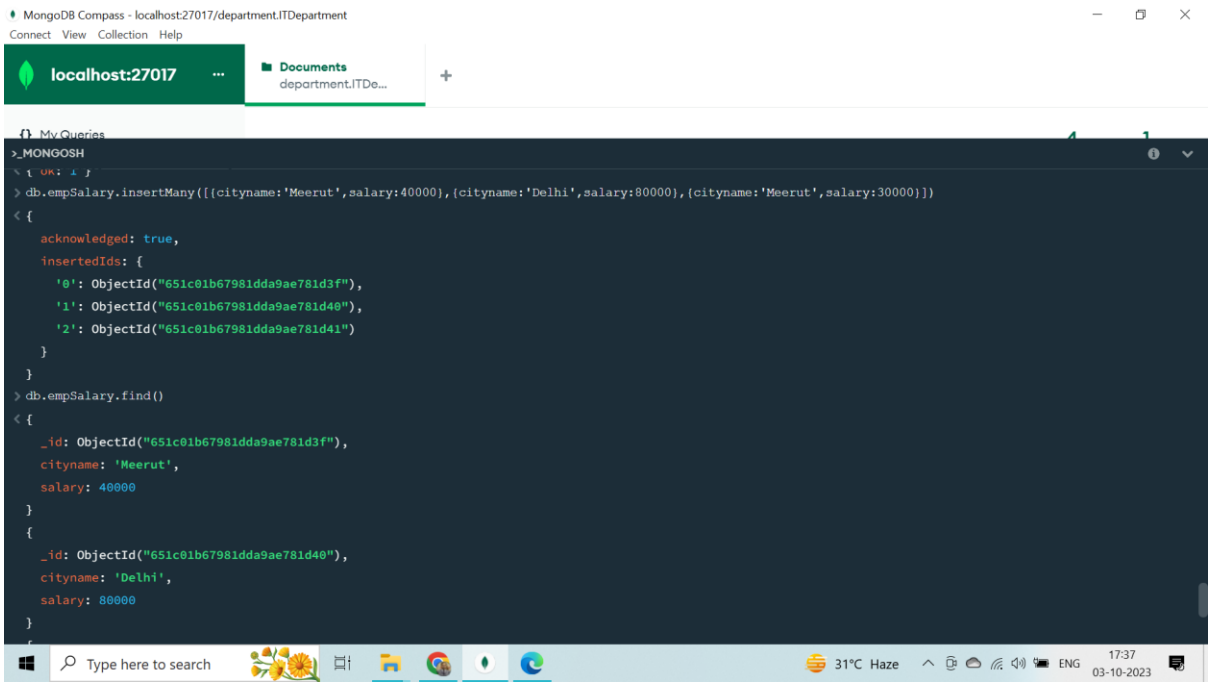
Aggregation Pipelines

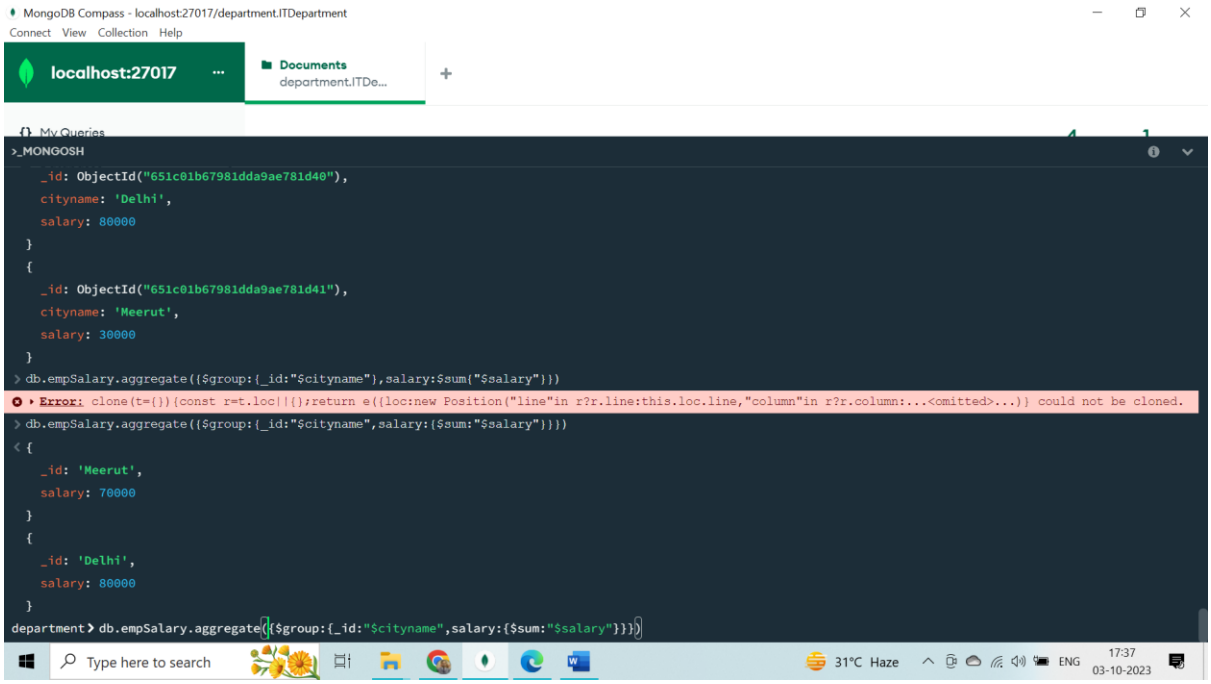
- Aggregation operations allow you to group, sort, perform calculations, analyze data, and much more.
- Aggregation pipelines can have one or more "stages". The order of these stages are important. Each stage acts upon the results of the previous stage.

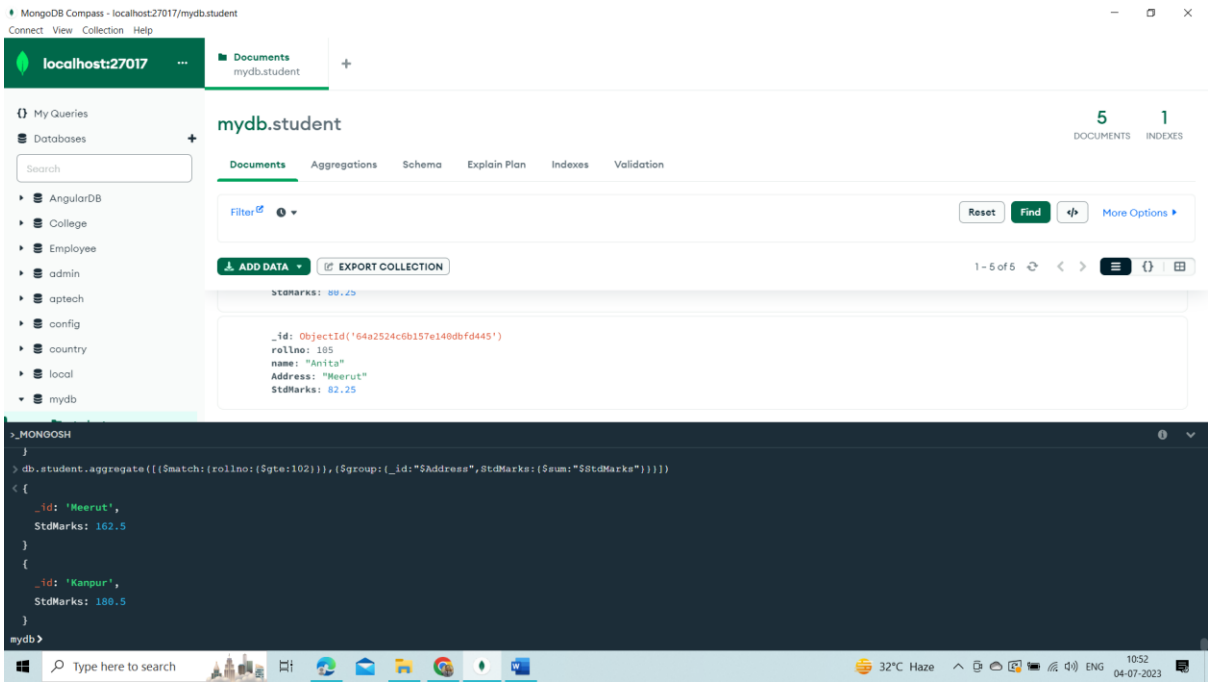
Example

```
db.posts.aggregate([
  // Stage 1: Only find documents that have more than 1 like
  {
    $match: { likes: { $gt: 1 } }
  },
  // Stage 2: Group documents by category and sum each categories likes
  {
    $group: { _id: "$category", totalLikes: { $sum: "$likes" } }
  }
])
```

```
[ { _id: 'News', totalLikes: 3 }, { _id: 'Event', totalLikes: 8 } ]
Atlas atlas-8iy36m-shard-0 [primary] blog>
```







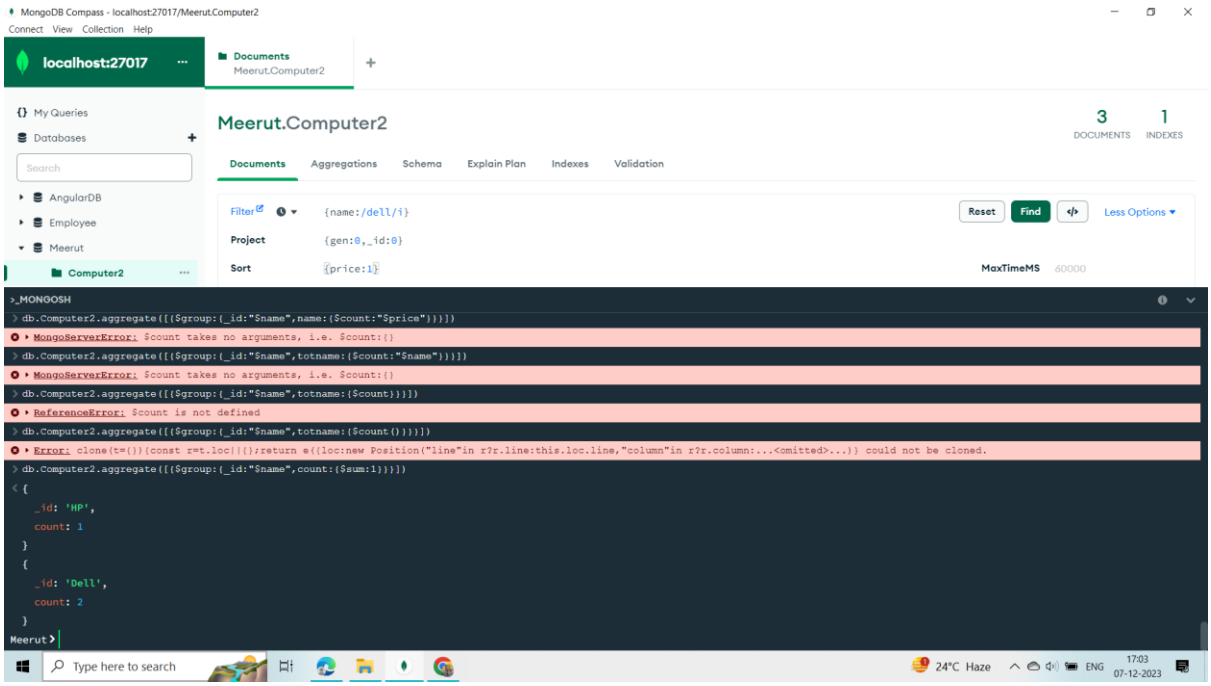
Sorting with Aggregate

```
db.student.aggregate([{$sort:{rollno:-1}}])
```

Sorting with Sort Function

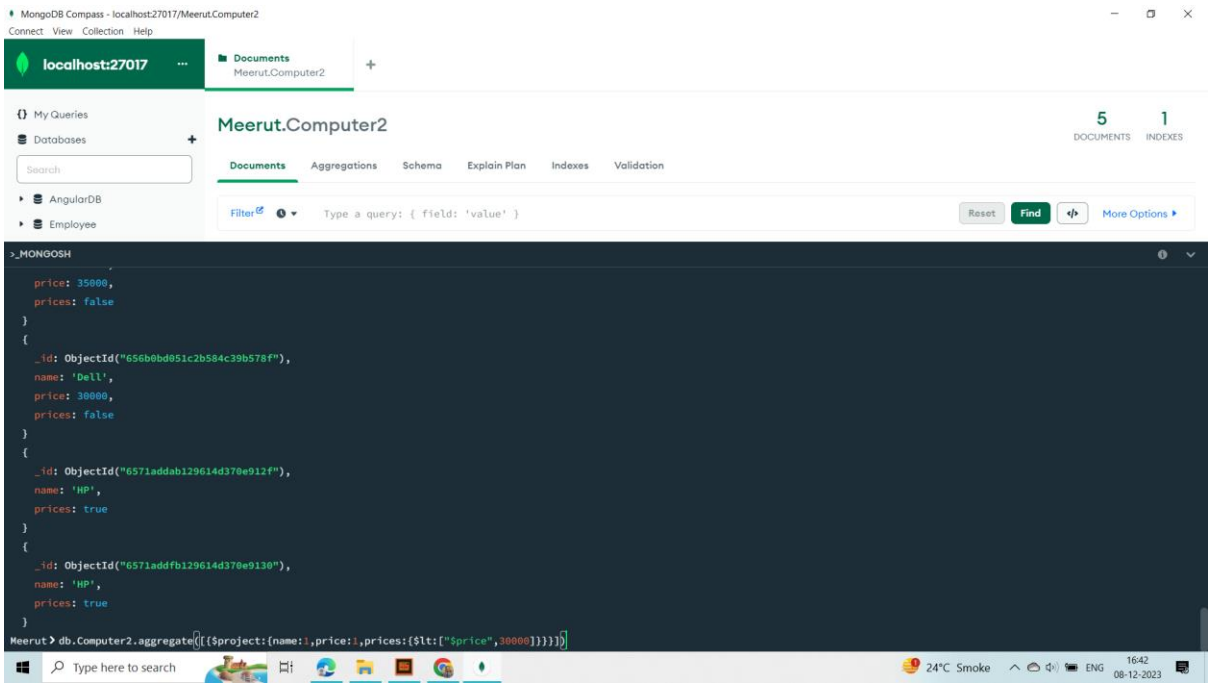
```
db.student.find({}).sort({rollno:-1})
```

Count with sum



- # Sample Data
- To demonstrate the use of stages in a aggregation pipeline, we will load sample data into our database.
- From the MongoDB Atlas dashboard, go to Databases. Click the ellipsis and select "Load Sample Dataset". This will load several sample datasets into your database.
- In the next sections we will explore several aggregation pipeline stages in more detail using this sample data.

Aggregate with Project with $lt



MongoDB Aggregation $group

Aggregation $group

This aggregation stage groups documents by the unique `_id` expression provided.

Don't confuse this `_id` expression with the `_id` ObjectId provided to each document.

Example

In this example, we are using the "sample_airbnb" database loaded from our sample data in the Intro to Aggregations section.

```
db.listingsAndReviews.aggregate(

    [ { $group : { _id : "$property_type" } } ]

)
```

```
[
  { _id: 'Aparthotel' },
  { _id: 'Apartment' },
  { _id: 'Barn' },
  { _id: 'Bed and breakfast' },
  { _id: 'Boat' },
  { _id: 'Boutique hotel' },
  { _id: 'Bungalow' },
  { _id: 'Cabin' },
  { _id: 'Camper/RV' },
  { _id: 'Campsite' },
  { _id: 'Casa particular (Cuba)' },
  { _id: 'Castle' },
  { _id: 'Chalet' },
  { _id: 'Condominium' },
  { _id: 'Cottage' },
  { _id: 'Earth house' },
  { _id: 'Farm stay' },
  { _id: 'Guest suite' },
  { _id: 'Guesthouse' },
  { _id: 'Heritage hotel (India)' }
]
Atlas atlas-8iy36m-shard-0 [primary] sample_airbnb>
```

This will return the distinct values from the `property_type` field.

MongoDB Aggregation $limit

Aggregation `$limit`

This aggregation stage limits the number of documents passed to the next stage.

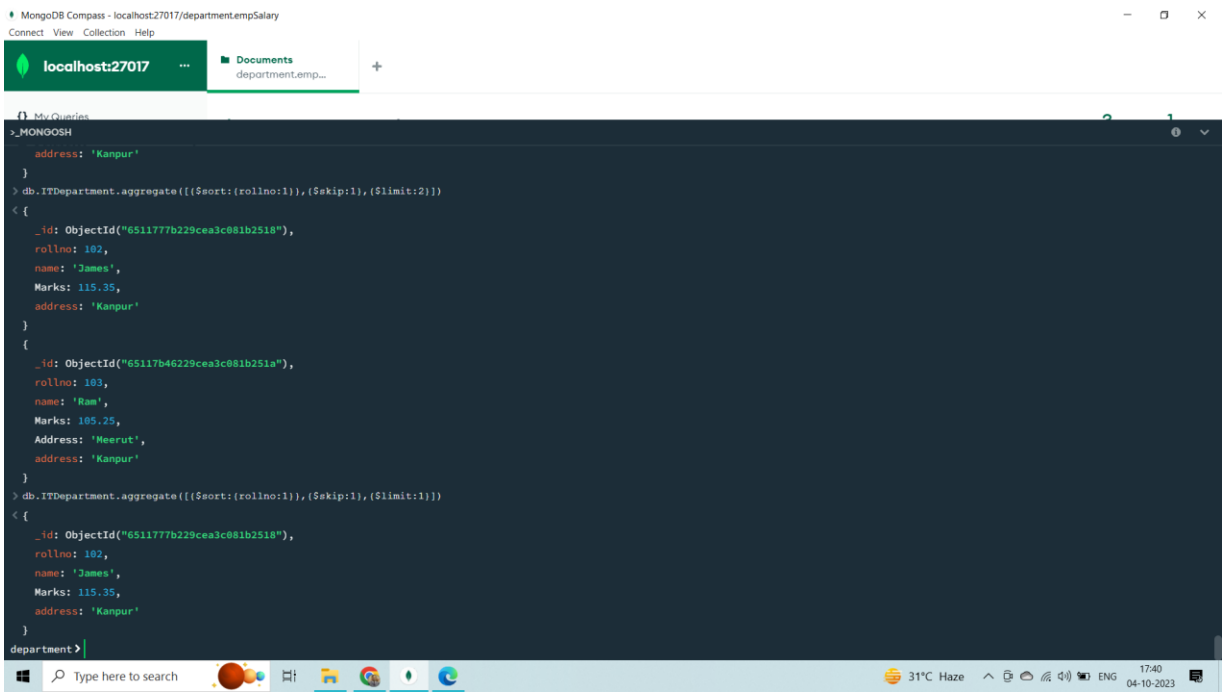
Example

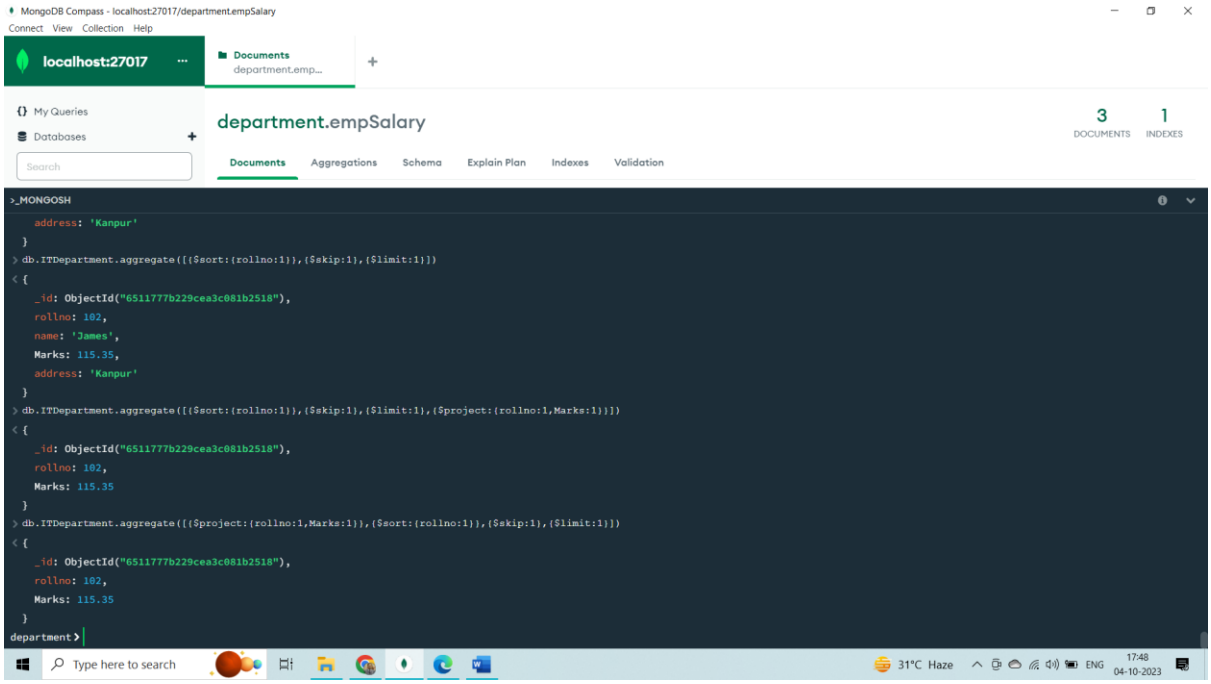
In this example, we are using the "sample_mflix" database loaded from our sample data in the Intro to Aggregations section.

```
db.movies.aggregate([ { $limit: 1 } ])
```

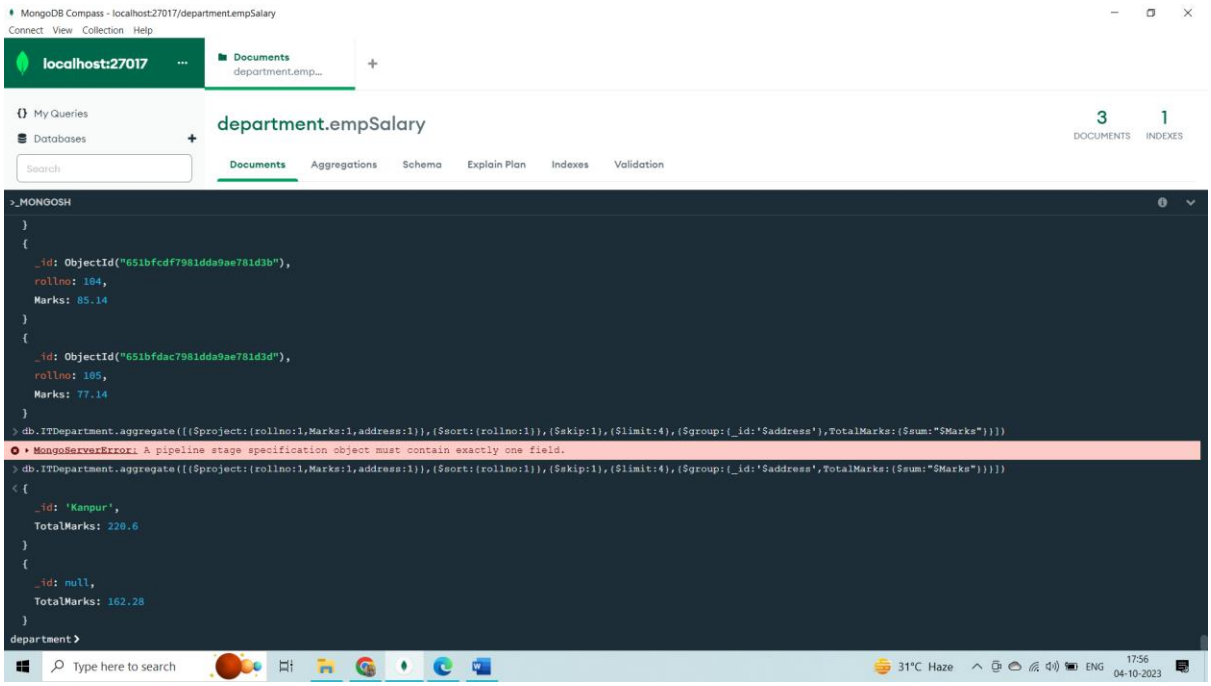
```
[
  {
    _id: ObjectId("573a1390f29313caabcd4135"),
    plot: 'Three men hammer on an anvil and pass a bottle of beer around.',
    genres: [ 'Short' ],
    runtime: 1,
    cast: [ 'Charles Kayser', 'John Ott' ],
    num_mflix_comments: 0,
    title: 'Blacksmith Scene',
    fullplot: 'A stationary camera looks at a large anvil with a blacksmith behind it an
    countries: [ 'USA' ],
    released: ISODate("1893-05-09T00:00:00.000Z"),
    directors: [ 'William K.L. Dickson' ],
    rated: 'UNRATED',
    awards: { wins: 1, nominations: 0, text: '1 win.' },
    lastupdated: '2015-08-26 00:03:50.133000000',
    year: 1893,
    imdb: { rating: 6.2, votes: 1189, id: 5 },
    type: 'movie',
    tomatoes: {
      viewer: { rating: 3, numReviews: 184, meter: 32 },
      lastUpdated: ISODate("2015-06-28T18:34:09.000Z")
    }
```

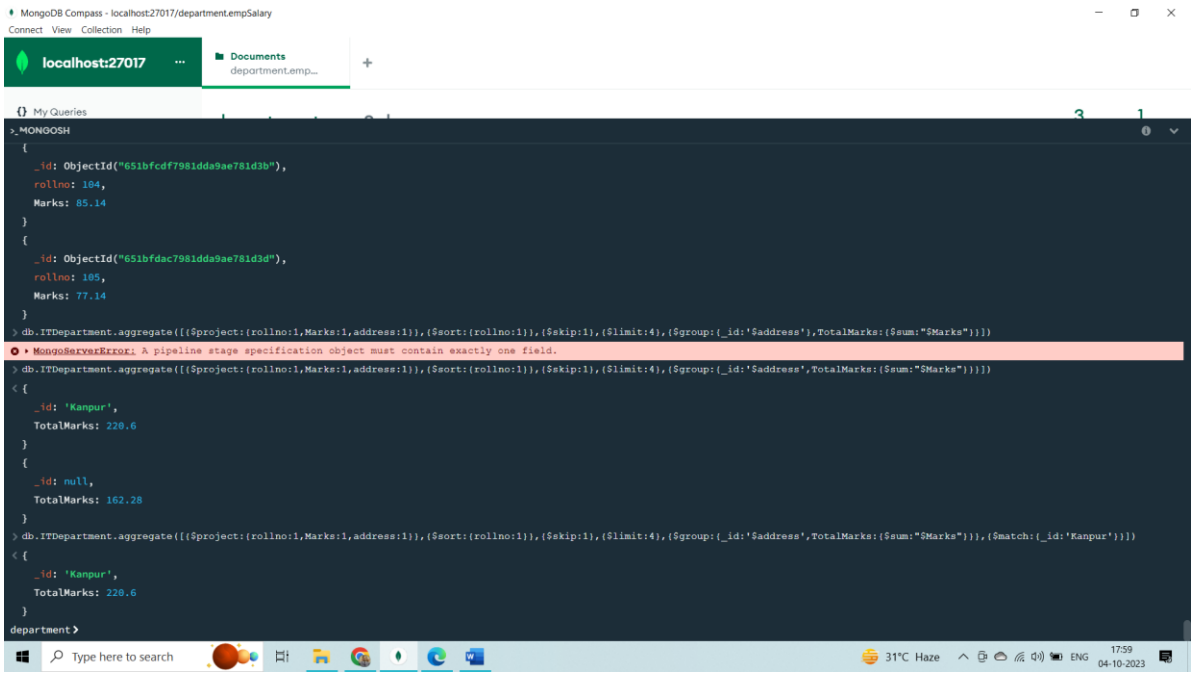
Limit show only 1st Record

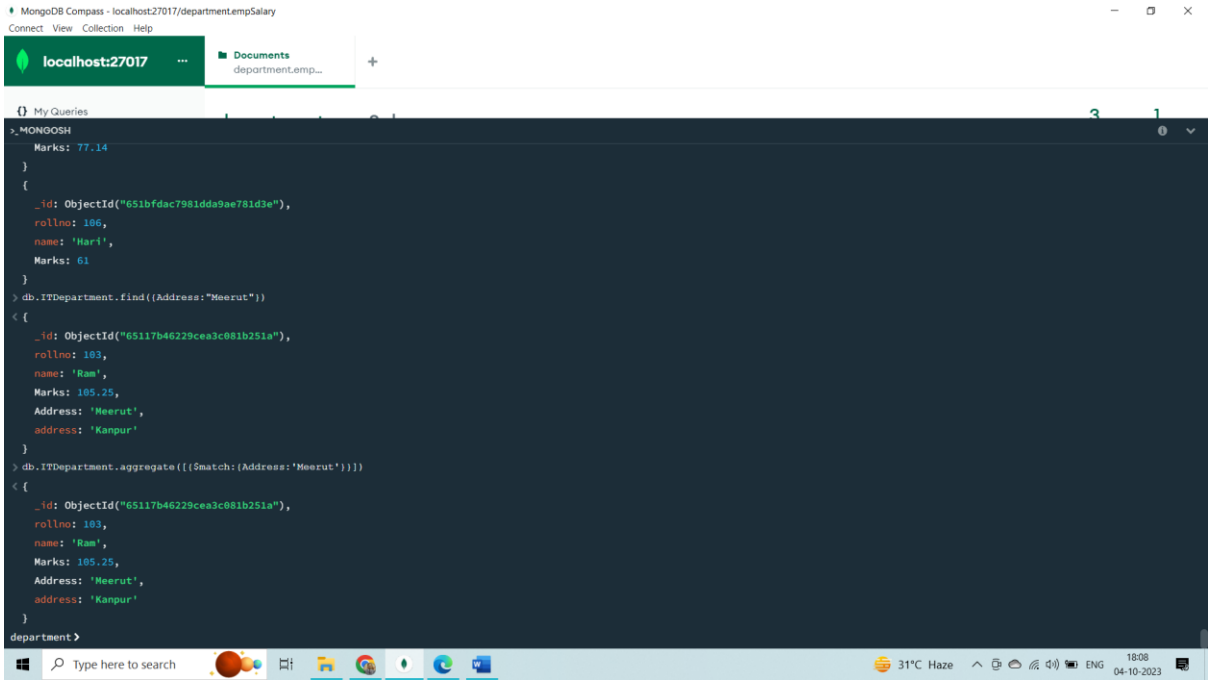



MongoDB Compass - localhost:27017/department.empSalary
Connect View Collection Help
localhost:27017
Documents
department.emp...
My Queries
Databases
Search
department.empSalary
3 DOCUMENTS
1 INDEXES
Documents Aggregations Schema Explain Plan Indexes Validation
>_MONGOSH
address: 'Kanpur'
}
db.ITDepartment.aggregate([{$sort:{rollno:1}},{$skip:1},{$limit:1}])
{
_id: ObjectId("6511777b229cea3c081b2518"),
rollno: 102,
name: 'James',
Marks: 115.35,
address: 'Kanpur'
}
db.ITDepartment.aggregate([{$sort:{rollno:1}},{$skip:1},{$limit:1},{$project:{rollno:1,Marks:1}}])
{
_id: ObjectId("6511777b229cea3c081b2518"),
rollno: 102,
Marks: 115.35
}
db.ITDepartment.aggregate([{$project:{rollno:1,Marks:1}},{$sort:{rollno:1}},{$skip:1},{$limit:1}])
{
_id: ObjectId("6511777b229cea3c081b2518"),
rollno: 102,
Marks: 115.35
}
department
Type here to search
31°C Haze
ENG
17:48
04-10-2023

With group


MongoDB Compass - localhost:27017/department.empSalary
Connect View Collection Help
localhost:27017
Documents
department.emp...
My Queries
Databases
Search
department.empSalary
3 DOCUMENTS
1 INDEXES
Documents Aggregations Schema Explain Plan Indexes Validation
>_MONGOSH
}
{
_id: ObjectId("651bfcdf7981dda9ae781d3b"),
rollno: 104,
Marks: 85.14
}
{
_id: ObjectId("651bfdac7981dda9ae781d3d"),
rollno: 105,
Marks: 77.14
}
db.ITDepartment.aggregate([{$project:{rollno:1,Marks:1,address:1}},{$sort:{rollno:1}},{$skip:1},{$limit:4},{$group:{_id:'$address'},TotalMarks:{$sum:"$Marks"}}])
MongoServerError: A pipeline stage specification object must contain exactly one field.
db.ITDepartment.aggregate([{$project:{rollno:1,Marks:1,address:1}},{$sort:{rollno:1}},{$skip:1},{$limit:4},{$group:{_id:'$address',TotalMarks:{$sum:"$Marks"}}}])
{
_id: 'Kanpur',
TotalMarks: 220.6
}
{
_id: null,
TotalMarks: 162.28
}
department
Type here to search
31°C Haze
ENG
17:56
04-10-2023





MongoDB Aggregation $project

Aggregation `$project`

This aggregation stage passes only the specified fields along to the next aggregation stage.

This is the same projection that is used with the `find()` method.

Example

In this example, we are using the "sample_restaurants" database loaded from our sample data in the Intro to Aggregations section.

```
db.restaurants.aggregate([
  {
    $project: {
      "name": 1,
      "cuisine": 1,
      "address": 1
    }
  },
  {
    $limit: 5
  }
])
```

```
[
    {
      _id: ObjectId("5eb3d668b31de5d588f4292a"),
      address: {
        building: '2780',
        coord: [ -73.98241999999999, 40.579505 ],
        street: 'Stillwell Avenue',
        zipcode: '11224'
      },
      cuisine: 'American',
      name: 'Riviera Caterer'
    },
    {
      _id: ObjectId("5eb3d668b31de5d588f4292b"),
      address: {
        building: '7114',
        coord: [ -73.9068506, 40.6199034 ],
        street: 'Avenue U',
        zipcode: '11234'
      },
      cuisine: 'Delicatessen',
      name: "Wilken'S Fine Food"
    },
    {
```

This will return the documents but only include the specified fields.

Notice that the `_id` field is also included. This field is always included unless specifically excluded.

We use a `1` to include a field and `0` to exclude a field.

Note: You cannot use both 0 and 1 in the same object. The only exception is the `_id` field. You should either specify the fields you would like to include or the fields you would like to exclude.

MongoDB Aggregation `$sort`

Aggregation `$sort`

This aggregation stage groups sorts all documents in the specified sort order.

Remember that the order of your stages matters. Each stage only acts upon the documents that previous stages provide.

Example

In this example, we are using the "sample_airbnb" database loaded from our sample data in the Intro to Aggregations section.

```
db.listingsAndReviews.aggregate([
  {
    $sort: { "accommodates": -1 }
  },
  {
    $project: {
      "name": 1,
      "accommodates": 1
    }
  },
  {
    $limit: 5
  }
])
```

```
[
  {
    _id: '19990097',
    name: 'House close to station & direct to opera house....',
    accommodates: 16
  },
  { _id: '19587001', name: 'Kaena O Kekai', accommodates: 16 },
  {
    _id: '20958766',
    name: 'Great Complex of the Cellars',
    accommodates: 16
  },
  {
    _id: '12509339',
    name: 'Barra da Tijuca beach house',
    accommodates: 16
  },
  {
    _id: '20455499',
    name: 'DOWNTOWN VIP MONTREAL ,HIGH END DECOR,GOOD VALUE..',
    accommodates: 16
  }
]
Atlas atlas-8iy36m-shard-0 [primary] sample_airbnb>
```

This will return the documents sorted in descending order by the `accommodates` field.

The sort order can be chosen by using `1` or `-1`. `1` is ascending and `-1` is descending.

MongoDB Aggregation `$match`

Aggregation `$match`

This aggregation stage behaves like a find. It will filter documents that match the query provided.

Using `$match` early in the pipeline can improve performance since it limits the number of documents the next stages must process.

Example

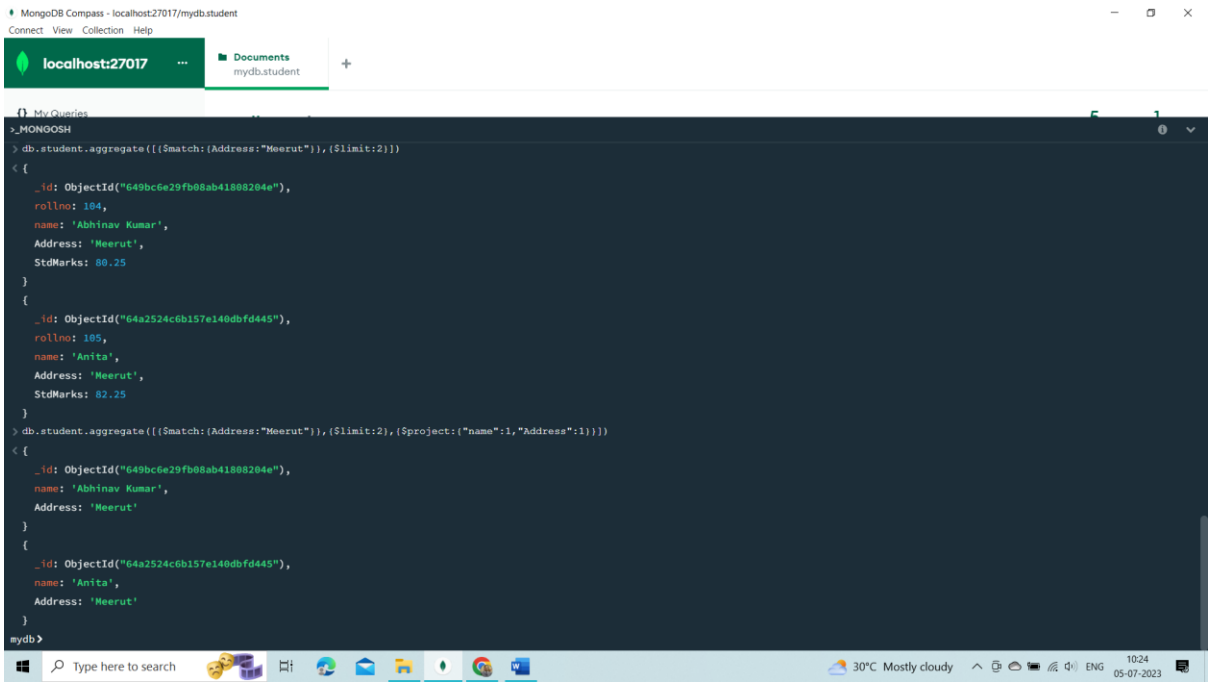
In this example, we are using the "sample_airbnb" database loaded from our sample data in the Intro to Aggregations section.

```
db.listingsAndReviews.aggregate([

  { $match : { property_type : "House" } },

  { $limit: 2 },
```

```
  { $project: {
    "name": 1,
    "bedrooms": 1,
    "price": 1
  }}
])
```

```
[
  {
    _id: '16253247',
    name: 'Gorgeous Remodeled Modern Home w/ Beach Across St.',
    bedrooms: 2,
    price: 194.00
  },
  {
    _id: '18616850',
    name: 'The Paddington Cottage | Sydney Eastern Suburbs',
    bedrooms: 3,
    price: 350.00
  }
]
Atlas atlas-8iy36m-shard-0 [primary] sample_airbnb>
```

This will only return documents that have the `property_type` of "House".



MongoDB Aggregation $addFields

Aggregation $addFields

This aggregation stage adds new fields to documents.

Example

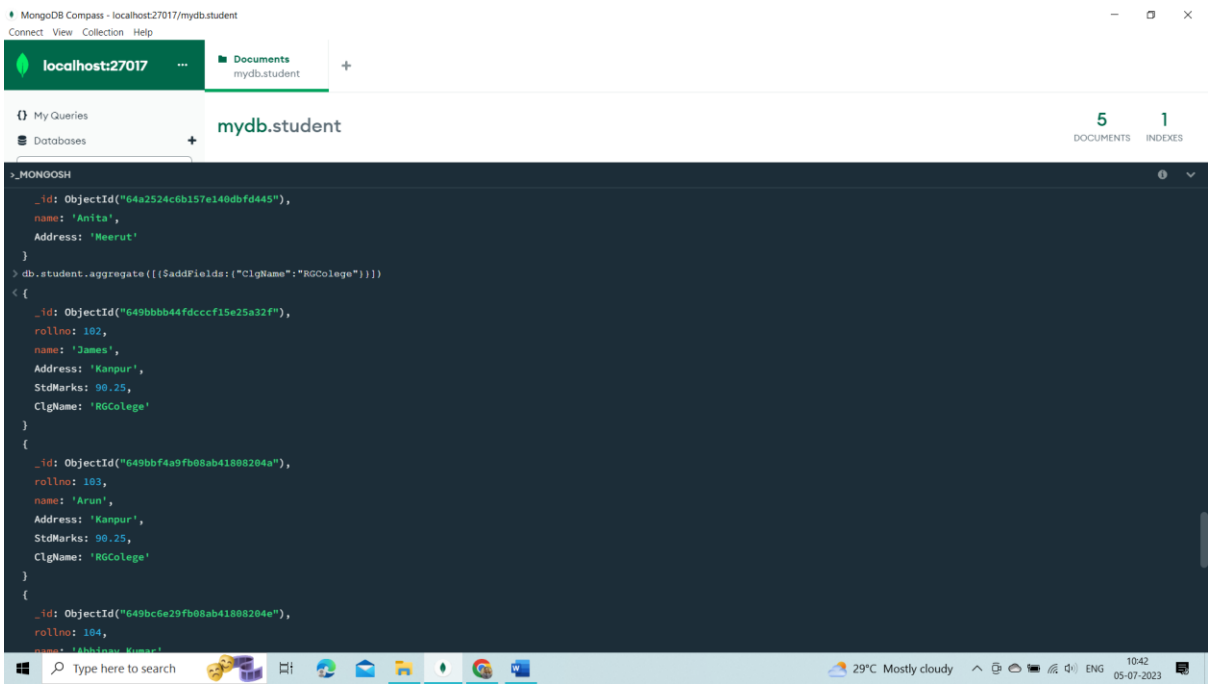
In this example, we are using the "sample_restaurants" database loaded from our sample data in the Intro to Aggregations section.

```
db.restaurants.aggregate([
  {
    $addFields: {
      avgGrade: { $avg: "$grades.score" }
    }
  },
  {
    $project: {
      "name": 1,
      "avgGrade": 1
    }
  },
```

```
  {
    $limit: 5
  }
])
```

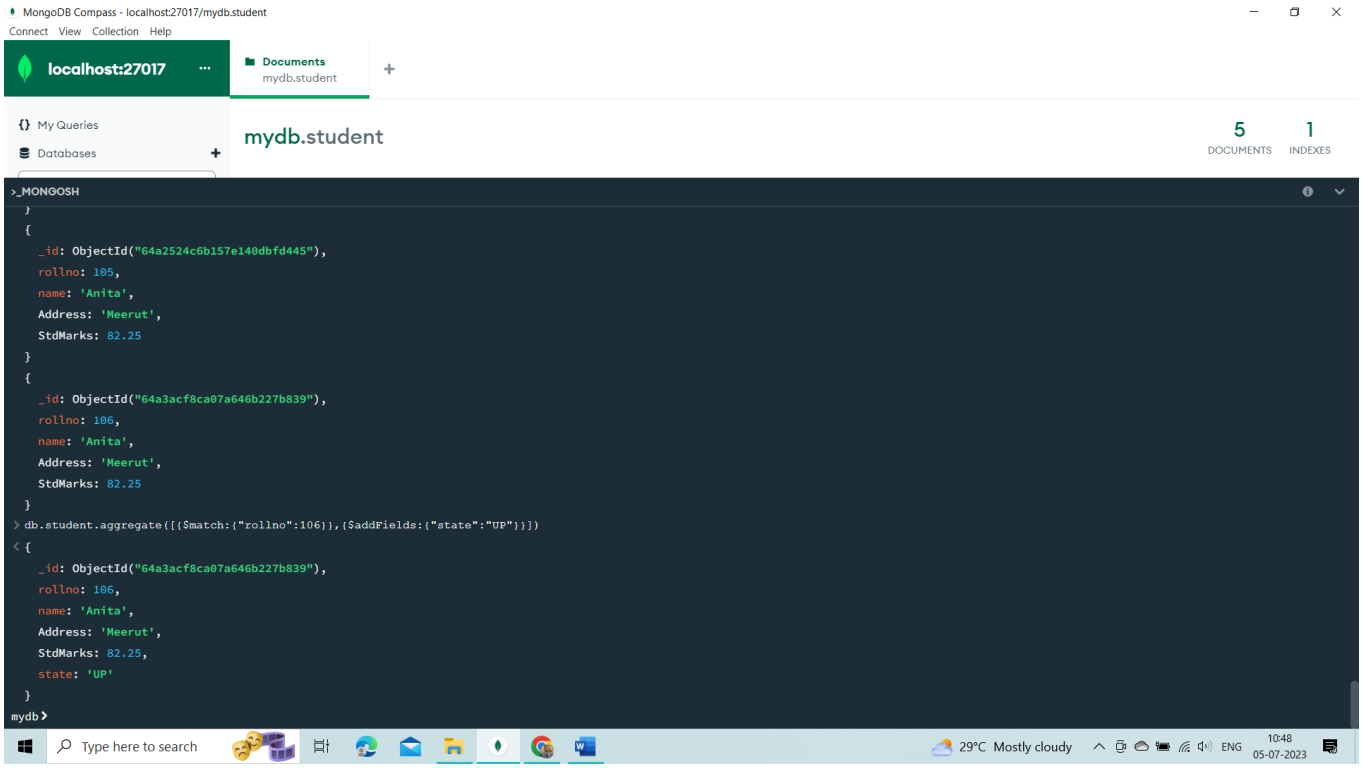
```
[
  {
    _id: ObjectId("5eb3d668b31de5d588f4292a"),
    name: 'Riviera Caterer',
    avgGrade: 9
  },
  {
    _id: ObjectId("5eb3d668b31de5d588f4292b"),
    name: "Wilken'S Fine Food",
    avgGrade: 10
  },
  {
    _id: ObjectId("5eb3d668b31de5d588f4292c"),
    name: 'Kosher Island',
    avgGrade: 10.5
  },
  {
    _id: ObjectId("5eb3d668b31de5d588f4292d"),
    name: "Wendy'S",
    avgGrade: 13.75
  },
  {
    _id: ObjectId("5eb3d668b31de5d588f4292e"),
    name: 'Morris Park Bake Shop',
```

Add Fields to all record



This will return the documents along with a new field, `avgGrade`, which will contain the average of each restaurants `grades.score`.

How to add Field to select record



MongoDB Aggregation $count

Aggregation $count

This aggregation stage counts the total amount of documents passed from the previous stage.

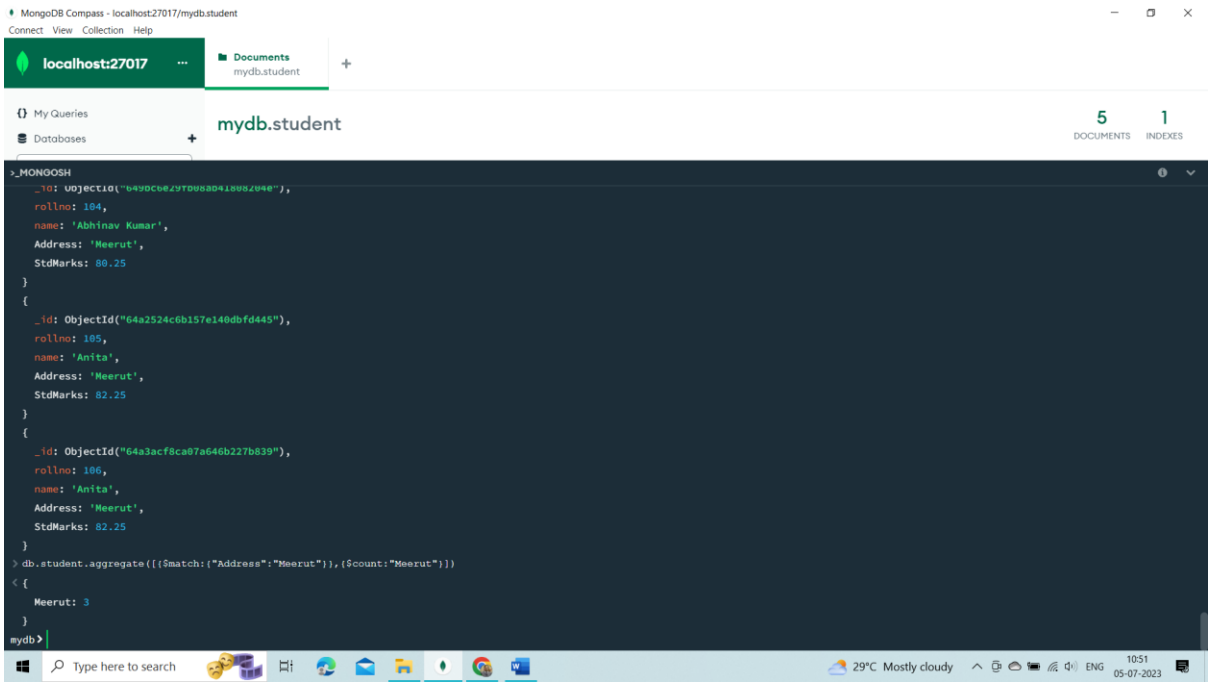
Example

In this example, we are using the "sample_restaurants" database loaded from our sample data in the Intro to Aggregations section.

```
db.restaurants.aggregate([
  {
    $match: { "cuisine": "Chinese" }
  },
  {
    $count: "totalChinese"
  }
])
```

```
[ { totalChinese: 2418 } ]
Atlas atlas-8iy36m-shard-0 [primary] sample_restaurants>
```

Counting



--

This will return the number of documents at the `$count` stage as a field called "totalChinese".

```
nployee> db.getCollection('Empinfo').find({})

{
  _id: ObjectId("63c0df6c1c3d68b027325a32"),
  empid: 101,
  title: 'IT Company Employee',
  salary: 41000
},
{
  _id: ObjectId("63c0e07a1c3d68b027325a34"),
  empid: 102,
  title: 'Sales Company Employee',
  salary: 20000
}

nployee> db.getCollection('Empinfo').find({},{empid:1,salary:1})

{
  _id: ObjectId("63c0df6c1c3d68b027325a32"),
  empid: 101,
  salary: 41000
},
{
  _id: ObjectId("63c0e07a1c3d68b027325a34"),
  empid: 102,
nployee>
```

$filter Aggregation

Json Data (name - collection)

[

{ "_id": 1,

"collapsed": [10, 20, 30, 40]

},

{ "_id": 2,

"collapsed": [11, 22, 33, 44] }

]

Query

db.collection.aggregate([

{

"$project": {

"evenArray": {

"$filter": {

```
        "input": "$collapsed",
        "as": "colsp",
        "cond": {
          "$eq": [ "$_id",  2 ]
        }    } }    }
  }
])
```

Output

```
[ {
   "_id": 1,
   "evenArray": []
 },
 {   "_id": 2,
   "evenArray": [     11,     22,     33,     44    ]
 }
]
```

Aggregation `$lookup`

This aggregation stage performs a left outer join to a collection in the same database.

There are four required fields:

- `from`: The collection to use for lookup in the same database
- `localField`: The field in the primary collection that can be used as a unique identifier in the `from` collection.
- `foreignField`: The field in the `from` collection that can be used as a unique identifier in the primary collection.

- `as`: The name of the new field that will contain the matching documents from the `from` collection.

Lookup complete Exercise use for join

1) Create Student Collections

```
db.students .insertMany([
{"id" : 1 , "pupil" : "John" , "std" : 6, "ht" : 153 , "wt" : 43},
{"id" : 2 , "pupil" : "Jack" , "std" : 6, "ht" : 164 , "wt" : 54},
])
```

2) Create sport Collections

```
db.sports .insertMany([
{"id" : 1, "sport" : "Basketball" , "winner" : "John"},
{"id" : 2, "sport" : "TT", "winner" : "Jack"},
{"id" : 3, "sport" : "Tennis" , "winner" : "John"},
])
```

The following query joins matches from sports collection to students collection based on the field value pupil and winner in students and sports respectively

Using Lookup

```
db.students.aggregate([
{
$lookup:
{
from : "sports",
localField : "pupil",
foreignField : "winner",
as : "games"
} } ] )
```

Result

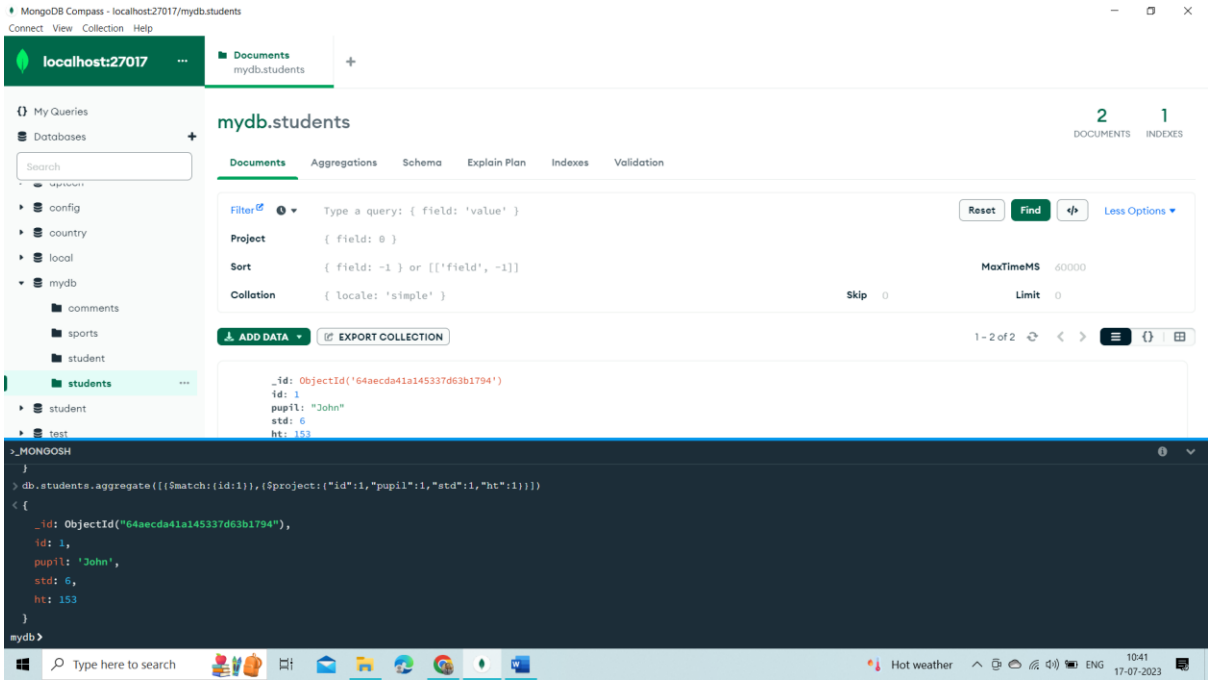
```
{
_id: ObjectId("64aecda41a145337d63b1794"),
id: 1,
pupil: 'John',
std: 6,
ht: 153,
```

```
wt: 43,
games: [
{
_id: ObjectId("64aece5d1a145337d63b1796"),
id: 1,
sport: 'Basketball',
winner: 'John'
},
{
_id: ObjectId("64aece5d1a145337d63b1798"),
id: 3,
sport: 'Tennis',
winner: 'John'
}
]
}
{
_id: ObjectId("64aecda41a145337d63b1795"),
id: 2,
pupil: 'Jack',
std: 6,
ht: 164,
wt: 54,
games: [
{
_id: ObjectId("64aece5d1a145337d63b1797"),
id: 2,
sport: 'TT',
winner: 'Jack'
}
]
}
```

Using Limit

```
db.students.aggregate([
{
$lookup:
{
from : "sports",
localField : "pupil",
foreignField : "winner",
as : "games"
} },
{ $limit: 1

 }])
```

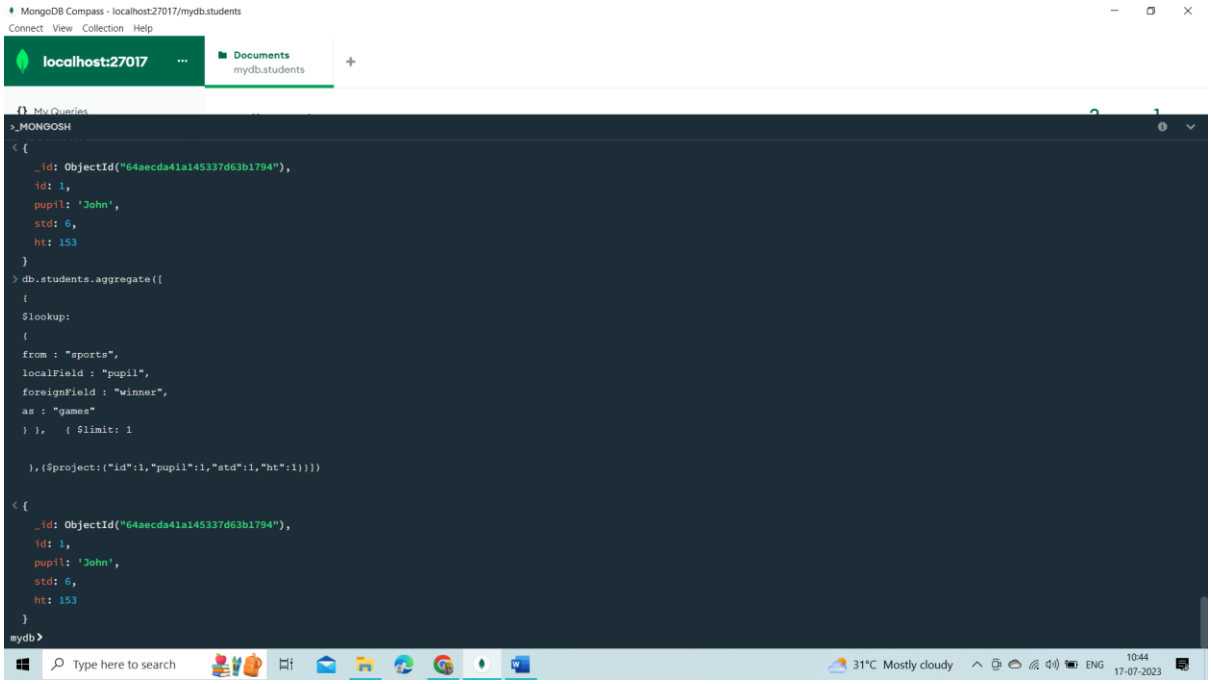
Match with project



Lookup with Limit and Project

```
db.students.aggregate([
{
$lookup:
{
from : "sports",
localField : "pupil",
foreignField : "winner",
as : "games"
} },
 {
   $limit: 1
 },
{
$project:
        {"id":1,
        "pupil":1,
        "std":1,
        "ht":1
         }
}])
```

Output



Second Exercise

Create two sample collections, `inventory` and `orders`:

```
db.inventory.insertMany( [
{ prodId: 100, price: 20, quantity: 125 },
{ prodId: 101, price: 10, quantity: 234 },
{ prodId: 102, price: 15, quantity: 432 },
{ prodId: 103, price: 17, quantity: 320 }
] )
```

```
db.orders.insertMany( [
{ orderId: 201, custid: 301, prodId: 100, numPurchased: 20 },
{ orderId: 202, custid: 302, prodId: 101, numPurchased: 10 },
{ orderId: 203, custid: 303, prodId: 102, numPurchased: 5 },
{ orderId: 204, custid: 303, prodId: 103, numPurchased: 15 },
{ orderId: 205, custid: 303, prodId: 103, numPurchased: 20 },
{ orderId: 206, custid: 302, prodId: 102, numPurchased: 1 },
{ orderId: 207, custid: 302, prodId: 101, numPurchased: 5 },
{ orderId: 208, custid: 301, prodId: 100, numPurchased: 10 },
{ orderId: 209, custid: 303, prodId: 103, numPurchased: 30 }
] )
```

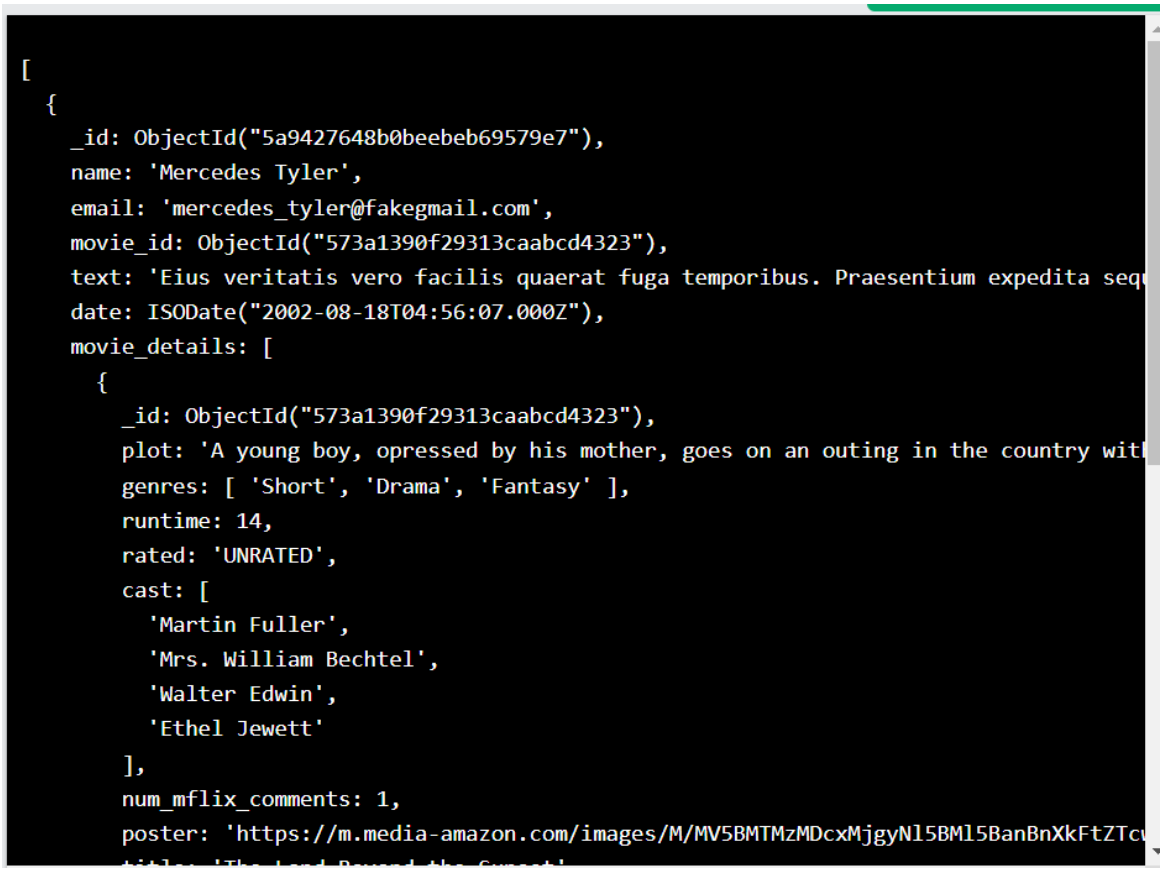
```
db.createView( "sales", "orders", [
    {
        $lookup:
            {
                from: "inventory",
                localField: "prodId",
```

```
                foreignField: "prodId",
                as: "inventoryDocs"
            }
    },
    {
        $project:
            {
              _id: 0,
              prodId: 1,
              orderId: 1,
              numPurchased: 1,
              price: "$inventoryDocs.price"
            }
    },
        { $unwind: "$price" }
] )
```

Example

In this example, we are using the "sample_mflix" database loaded from our sample data in the Intro to Aggregations section.

```
db.comments.aggregate([
  {
    $lookup: {
      from: "movies",
      localField: "movie_id",
      foreignField: "_id",
      as: "movie_details",
    },
  },
  {
    $limit: 1
  }
])
```



This will return the movie data along with each comment.

This is the full set of aggregation stages:

- `$match` – Filter documents
- `$geoNear` – Sort documents based on geographic proximity
- `$project` – Reshape documents (remove or rename keys or add new data based on calculations on the existing data)
- `$lookup` – Coming in 3.2 – Left-outer joins
- `$unwind` – Expand documents (for example create multiple documents where each contains one element from an array from the original document)
- `$group` – Summarize documents
- `$sample` – Randomly selects a subset of documents
- `$sort` – Order documents

- `$skip` – Jump over a number of documents
- `$limit` – Limit number of documents
- `$redact` – Restrict sensitive content from documents
- `$out` – *Coming in 3.2** – store the results in a new collection

MongoDB Aggregation $out

Aggregation $out

This aggregation stage writes the returned documents from the aggregation pipeline to a collection.

The $out stage must be the last stage of the aggregation pipeline.

Example

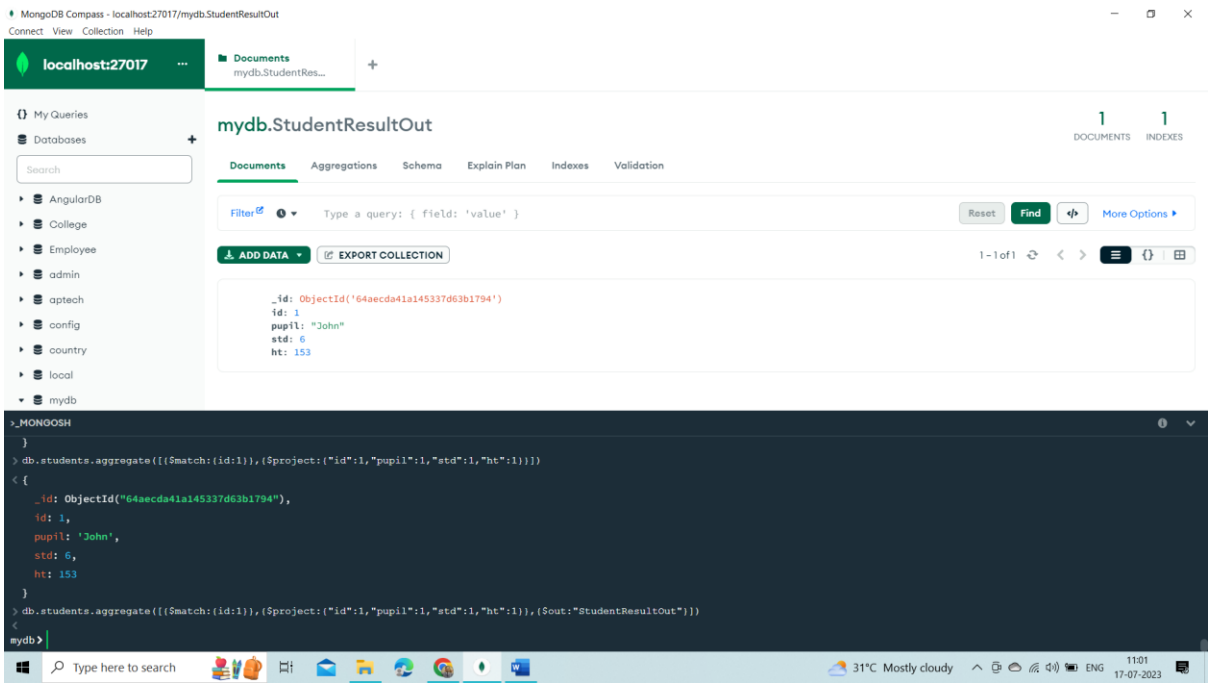
In this example, we are using the "sample_airbnb" database loaded from our sample data in the Intro to Aggregations section.

```
db.listingsAndReviews.aggregate([
  {
    $group: {
      _id: "$property_type",
      properties: {
        $push: {
          name: "$name",
          accommodates: "$accommodates",
          price: "$price",
        },
      },
    },
  },
  { $out: "properties_by_type" },
])
```

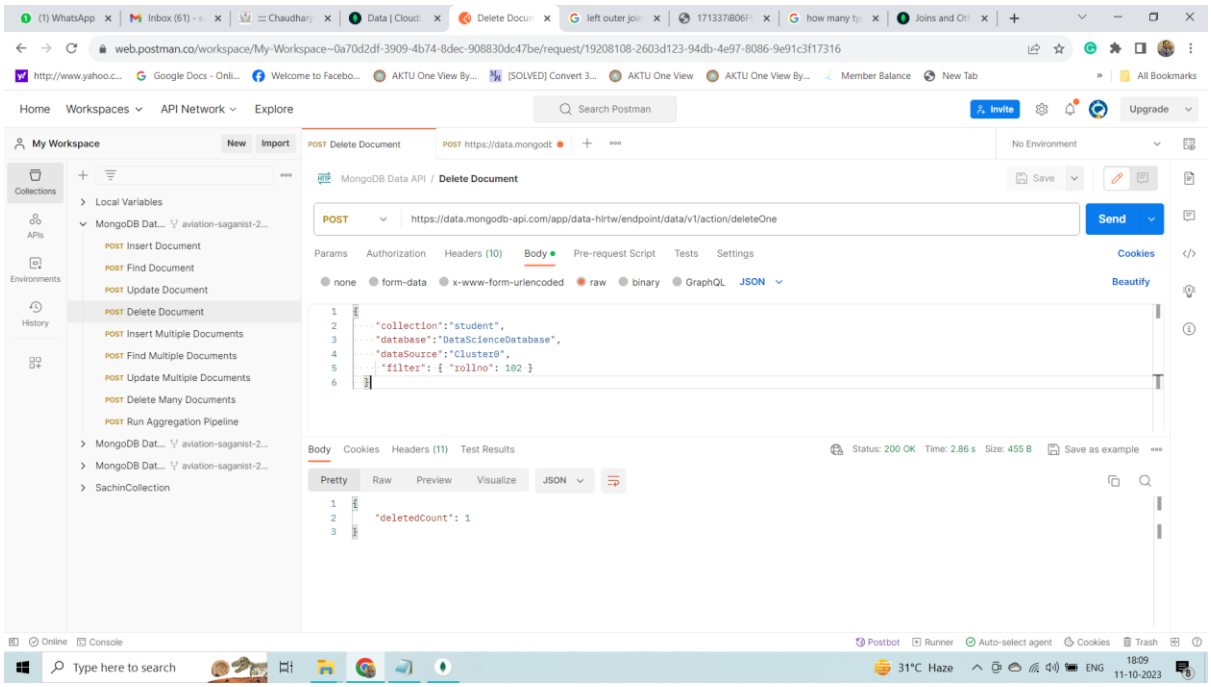
```
Results saved to collection: properties_by_type
```

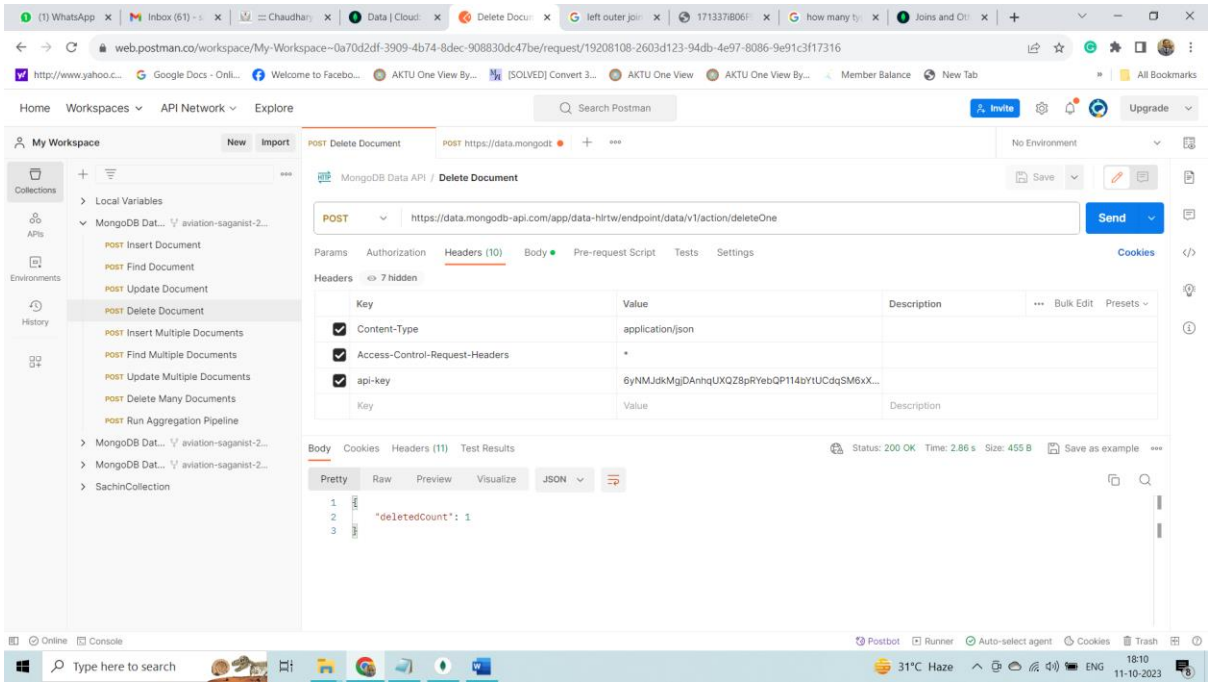
Save Result into out

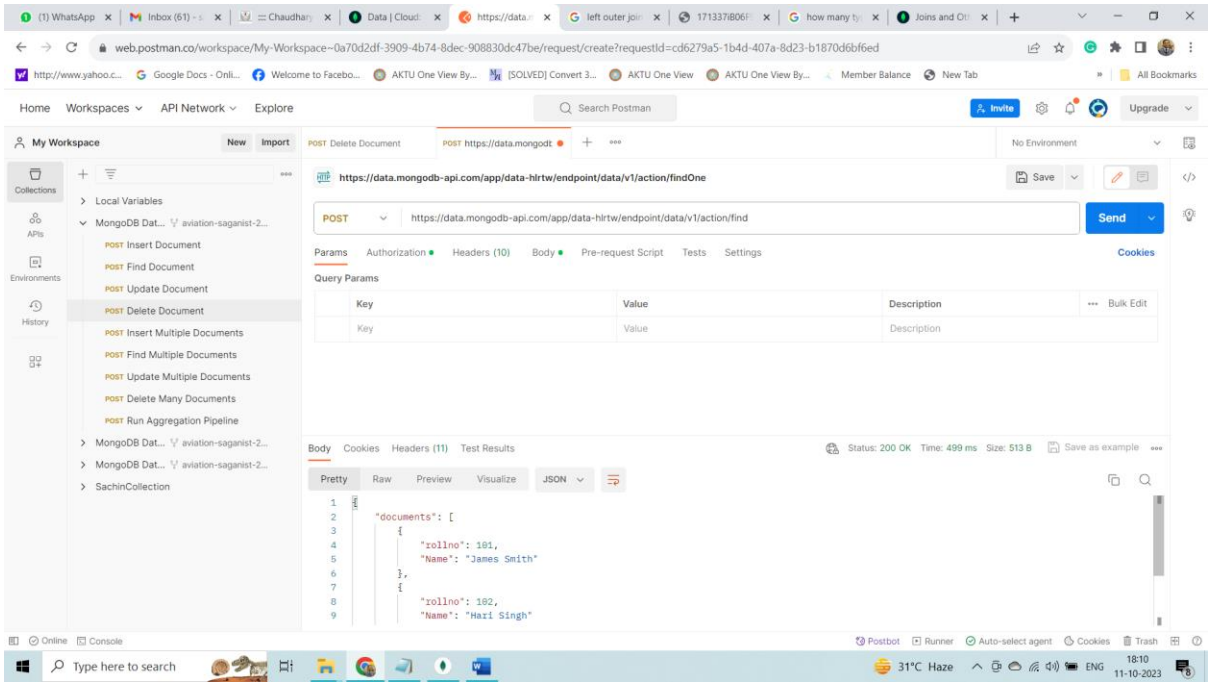
```
db.students.aggregate([
{
$match:{id:1}
},
{
$project:{"id":1,"pupil":1,"std":1,"ht":1}
},
{
$out:"StudentResultOut"}
])
```

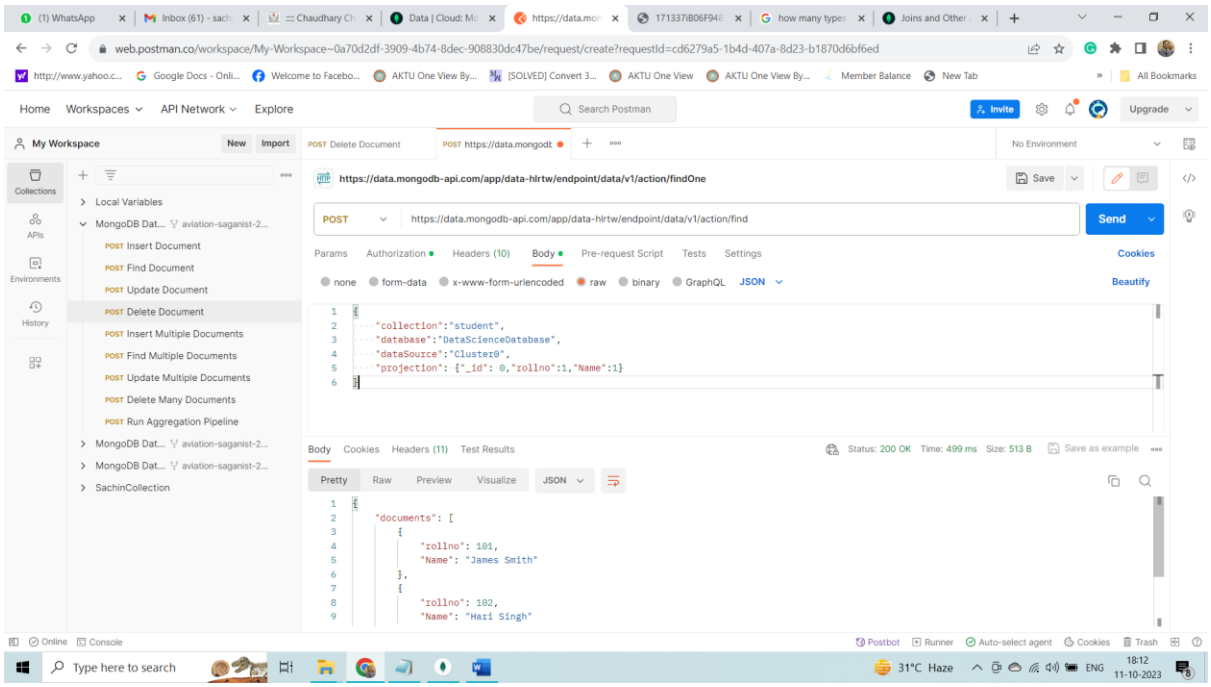


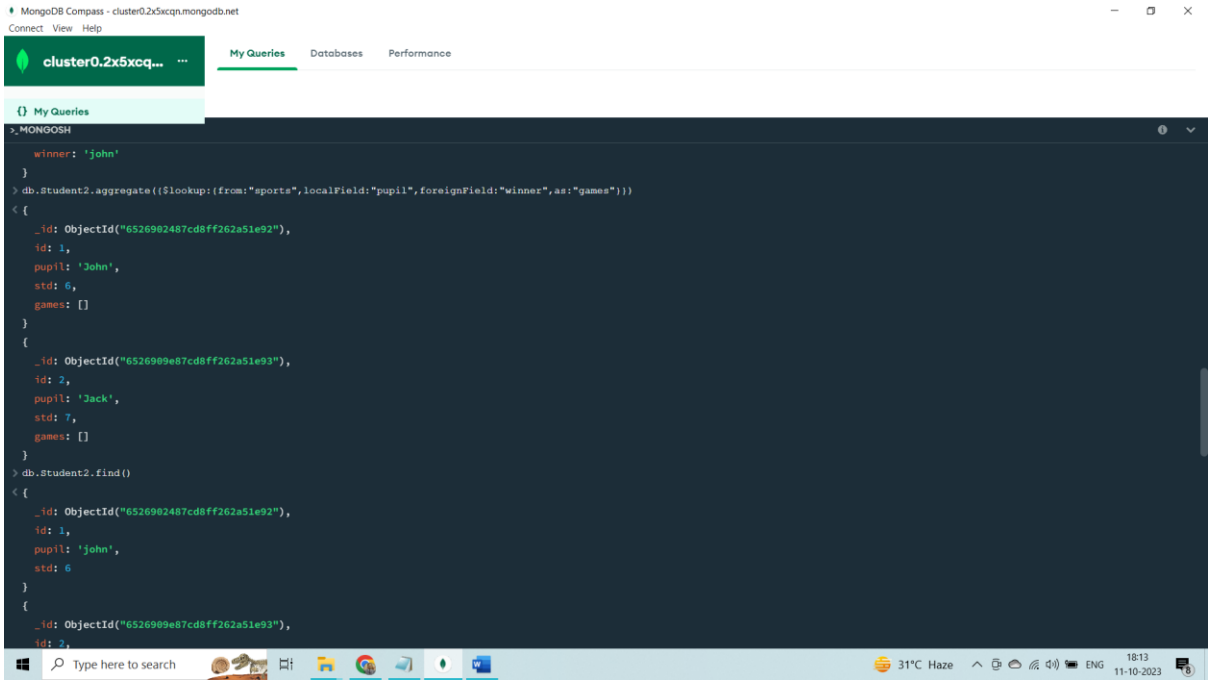
The first stage will group properties by the `property_type` and include the `name`, `accommodates`, and `price` fields for each. The `$out` stage will create a new collection called `properties_by_type` in the current database and write the resulting documents into that collection.

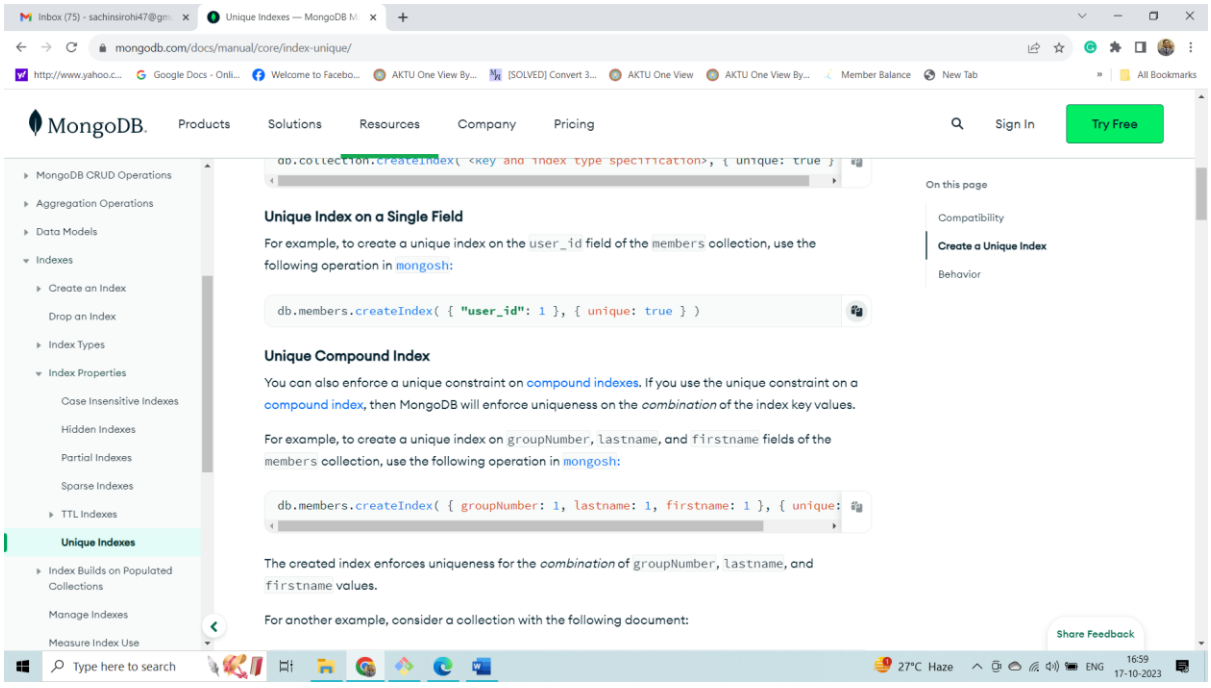

web.postman.co/workspace/My-Workspace~0a70d2df-3909-4b74-8dec-908830dc47be/request/19208108-2603d123-94db-4e97-8086-9e91c3f17316
Home
Workspaces
API Network
Explore
Search Postman
Invite
Upgrade
My Workspace
New
Import
POST Delete Document
POST https://data.mongodt
No Environment
MongoDB Data API / Delete Document
Save
POST
https://data.mongodb-api.com/app/data-hlrtw/endpoint/data/v1/action/deleteOne
Send
Params
Authorization
Headers (10)
Body
Pre-request Script
Tests
Settings
Cookies
none
form-data
x-www-form-urlencoded
raw
binary
GraphQL
JSON
Beautify
{
"collection":"student",
"database":"DataScienceDatabase",
"dataSource":"Cluster0",
"filter": { "rollno": 102 }
}
Body
Cookies
Headers (11)
Test Results
Status: 200 OK
Time: 2.86 s
Size: 455 B
Save as example
Pretty
Raw
Preview
Visualize
JSON
{
"deletedCount": 1
}
Collections
APIs
Environments
History
Local Variables
MongoDB Dat... aviation-saganist-2...
POST Insert Document
POST Find Document
POST Update Document
POST Delete Document
POST Insert Multiple Documents
POST Find Multiple Documents
POST Update Multiple Documents
POST Delete Many Documents
POST Run Aggregation Pipeline
MongoDB Dat... aviation-saganist-2...
MongoDB Dat... aviation-saganist-2...
SachinCollection
Online
Console
Postbot
Runner
Auto-select agent
Cookies
Trash
Type here to search
31°C Haze
ENG
18:09
11-10-2023


web.postman.co/workspace/My-Workspace~0a70d2df-3909-4b74-8dec-908830dc47be/request/19208108-2603d123-94db-4e97-8086-9e91c3f17316
Home
Workspaces
API Network
Explore
Search Postman
Invite
Upgrade
My Workspace
New
Import
POST Delete Document
POST https://data.mongodt
No Environment
MongoDB Data API / Delete Document
Save
POST
https://data.mongodb-api.com/app/data-hlrtw/endpoint/data/v1/action/deleteOne
Send
Params
Authorization
Headers (10)
Body
Pre-request Script
Tests
Settings
Cookies
Headers
7 hidden
Key
Value
Description
Bulk Edit
Presets
Content-Type
application/json
Access-Control-Request-Headers
*
api-key
6yNMJdkMgjDAnhqUXQZ8pRYebQP114bYtUCdqSM6xX...
Key
Value
Description
Body
Cookies
Headers (11)
Test Results
Status: 200 OK
Time: 2.86 s
Size: 455 B
Save as example
Pretty
Raw
Preview
Visualize
JSON
{
"deletedCount": 1
}
Collections
APIs
Environments
History
Local Variables
MongoDB Dat... aviation-saganist-2...
POST Insert Document
POST Find Document
POST Update Document
POST Delete Document
POST Insert Multiple Documents
POST Find Multiple Documents
POST Update Multiple Documents
POST Delete Many Documents
POST Run Aggregation Pipeline
MongoDB Dat... aviation-saganist-2...
MongoDB Dat... aviation-saganist-2...
SachinCollection
Online
Console
Postbot
Runner
Auto-select agent
Cookies
Trash
Type here to search
31°C Haze
ENG
18:10
11-10-2023


https://data.mongodb-api.com/app/data-hlrtw/endpoint/data/v1/action/findOne
POST
https://data.mongodb-api.com/app/data-hlrtw/endpoint/data/v1/action/find
Send
Params
Authorization
Headers (10)
Body
Pre-request Script
Tests
Settings
Cookies
Query Params
Key
Value
Description
Bulk Edit
Body
Cookies
Headers (11)
Test Results
Status: 200 OK
Time: 499 ms
Size: 513 B
Pretty
Raw
Preview
Visualize
JSON
{
"documents": [
{
"rollno": 101,
"Name": "James Smith"
},
{
"rollno": 102,
"Name": "Hari Singh"
Type here to search
31°C Haze
ENG
18:10
11-10-2023


https://data.mongodb-api.com/app/data-hlrtw/endpoint/data/v1/action/findOne
POST
https://data.mongodb-api.com/app/data-hlrtw/endpoint/data/v1/action/find
Send
Params
Authorization
Headers (10)
Body
Pre-request Script
Tests
Settings
Cookies
none
form-data
x-www-form-urlencoded
raw
binary
GraphQL
JSON
Beautify
{
"collection":"student",
"database":"DataScienceDatabase",
"dataSource":"Cluster0",
"projection": {"_id": 0,"rollno":1,"Name":1}
}
Body
Cookies
Headers (11)
Test Results
Status: 200 OK
Time: 499 ms
Size: 513 B
Pretty
Raw
Preview
Visualize
JSON
{
"documents": [
{
"rollno": 101,
"Name": "James Smith"
},
{
"rollno": 102,
"Name": "Hari Singh"
Type here to search
31°C Haze
ENG
18:12
11-10-2023


MongoDB Compass - cluster0.2x5xcqn.mongodb.net
Connect View Help
cluster0.2x5xcq...
My Queries
Databases
Performance
My Queries
>_MONGOSH
winner: 'john'
}
db.Student2.aggregate({$lookup:{from:"sports",localField:"pupil",foreignField:"winner",as:"games"}})
{
_id: ObjectId("6526902487cd8ff262a51e92"),
id: 1,
pupil: 'John',
std: 6,
games: []
}
{
_id: ObjectId("6526909e87cd8ff262a51e93"),
id: 2,
pupil: 'Jack',
std: 7,
games: []
}
db.Student2.find()
{
_id: ObjectId("6526902487cd8ff262a51e92"),
id: 1,
pupil: 'john',
std: 6
}
{
_id: ObjectId("6526909e87cd8ff262a51e93"),
id: 2,
Type here to search
31°C Haze
ENG
18:13
11-10-2023


Inbox (75) - sachinsirohi47@gm
Unique Indexes — MongoDB M
mongodb.com/docs/manual/core/index-unique/
MongoDB
Products
Solutions
Resources
Company
Pricing
Sign In
Try Free
MongoDB CRUD Operations
Aggregation Operations
Data Models
Indexes
Create an Index
Drop an Index
Index Types
Index Properties
Case Insensitive Indexes
Hidden Indexes
Partial Indexes
Sparse Indexes
TTL Indexes
Unique Indexes
Index Builds on Populated Collections
Manage Indexes
Measure Index Use
Unique Index on a Single Field
For example, to create a unique index on the user_id field of the members collection, use the following operation in mongosh:
db.members.createIndex({ "user_id": 1 }, { unique: true })
Unique Compound Index
You can also enforce a unique constraint on compound indexes. If you use the unique constraint on a compound index, then MongoDB will enforce uniqueness on the combination of the index key values.
For example, to create a unique index on groupNumber, lastname, and firstname fields of the members collection, use the following operation in mongosh:
db.members.createIndex({ groupNumber: 1, lastname: 1, firstname: 1 }, { unique:
The created index enforces uniqueness for the combination of groupNumber, lastname, and firstname values.
For another example, consider a collection with the following document:
On this page
Compatibility
Create a Unique Index
Behavior
Share Feedback
Type here to search
27°C Haze
ENG
16:59
17-10-2023