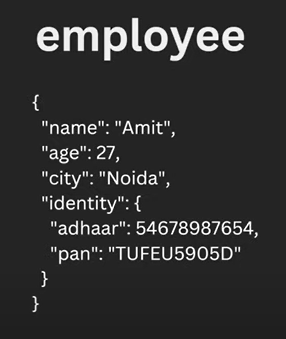
Mongo db = large database

Data stored in a json format

Json format :-



Schemas a re not present in the mongo db

Features :- flexibility incresed

Less relation

Data is stored together

* When we use same id for inserting a data it will create an error
* MongoDB work syncroniously so if error occur no further records get inserted

Working on port:- mongodb://localhost:27017

Commands:-[

**Show dbs** - showning all the dbs

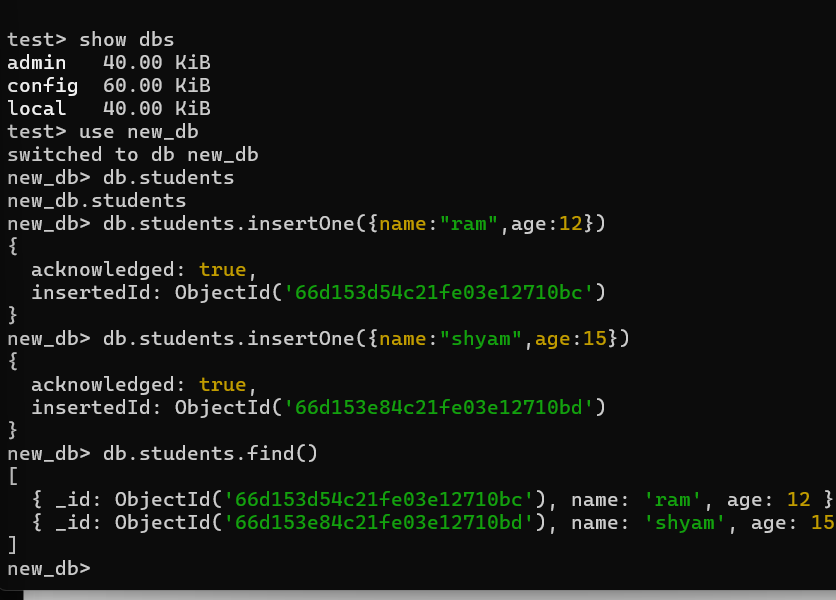
**Show collections:-** showing collections in the database

**Use (database name**) -create and switched to new database

**Db.(collection name)** – to create a collection inside the db only if not present in the database ortherwise it swich to collection ,

**Db.(collection name).insertOne({name:ram,id-12234,age-17})** – insert element into a database

**Db.student.find() –** see database or see all the element in the database bt sing **db**  refers current database and **.collection name** represents the collection in the database and **.find()**  represents the finding the all the elements in the database were it also include unique identifier **id** called

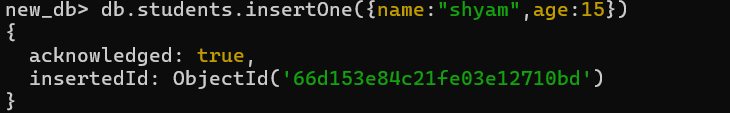


]

**CRUD in mongo db:**

**Create :** creating a document : we can insert document in two ways

1.**insertOne(data, options):**

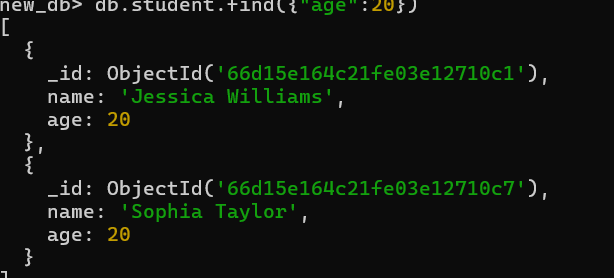
****

2.**insertMany(data,options):**

****

**Read:** reading document :

1.**find(filter,options):**

****

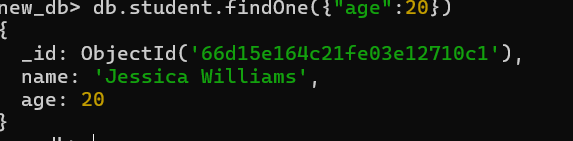
Find return value in list format []

**It matches all the elements and then returns the list of all the elements**

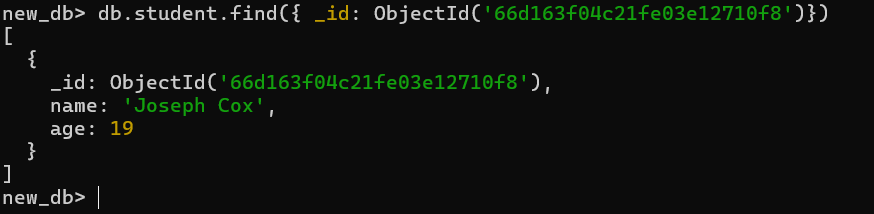
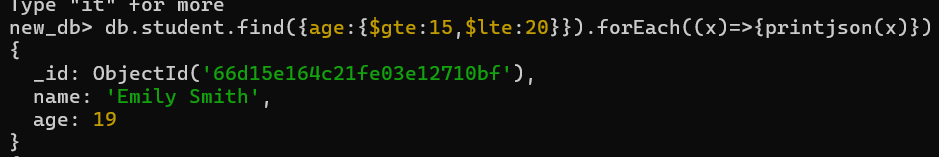
2.**findOne(filter,options):**

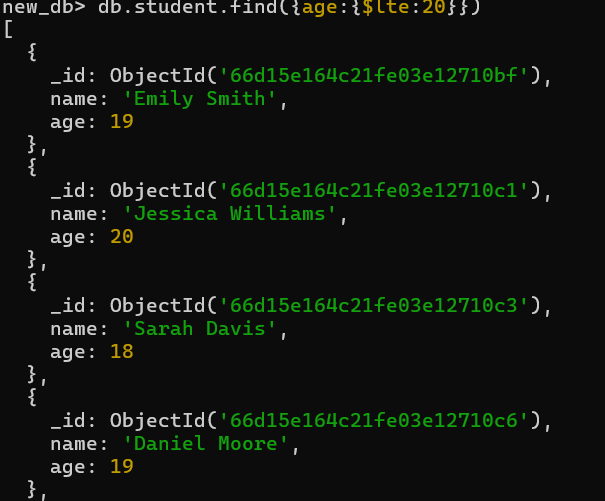
**It will shows the first data which matches the filter by order in the table**

**FindOne returns the first elemets in the whole list**

****

**Applying condition to fetch data :**

* **db.student.find({age:{$gte:20}}) returns the greater than equal =20**
* **db.student.find({age:{$lte:20}}) returns records less than or equal =20**
* **db.student.find({age:{$lt:20}}) returns records less than =20**
* **db.student.find({age:{$gt:20}}) returns records greater than =20**
* **db.student.find({age:{$gt:15,$lt:20}}) returns records greater than =15 and less than =20**
* **Db.student.find({\_id:ObjectId('66d163f04c21fe03e12710f8')})**
* ****
* **db.student.find({age:{$gte:20,$lte:25}}).forEach((x)=>{printjson(x)}),to display all the elemets with particular condition**
* **We can also find using default id `s provided by mongodb**
* **For eg.**
* ****

****

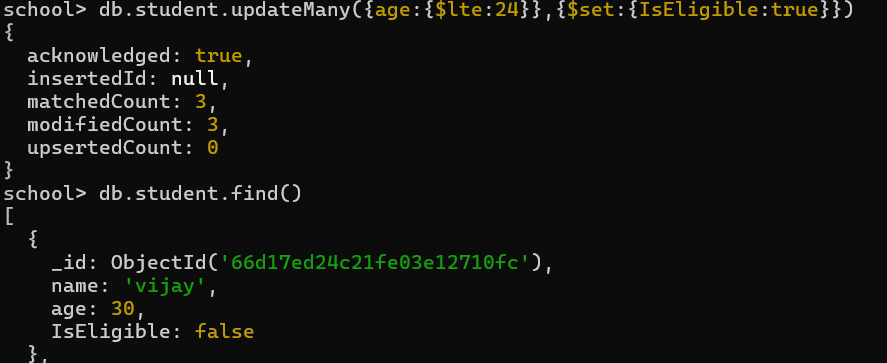
**Update:** update the value :

**While inserting value for key and key is not present in the database then database create that key and put it into particular record while using update function**

1.**updateOne({filter},{$set:{attribute:NewValue},options}):**

* ****

2.**updateMany(filter,data,options)**

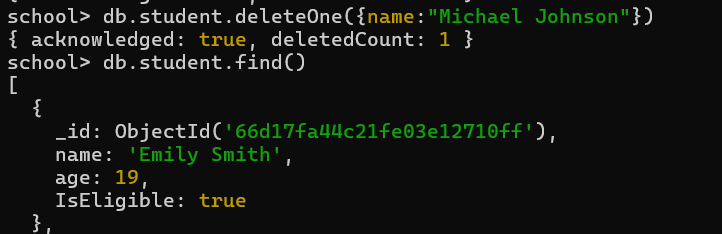
****

3.**replaceOne(filter,data,option)**

**Delete:** delete the element:

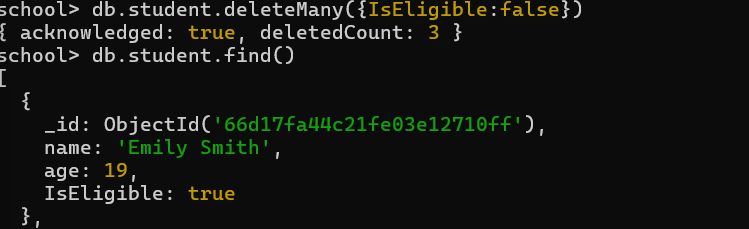
1.**deleteOne(filter, options)**

**Db.student.deleteOne({name:”Michael Johnson”})**

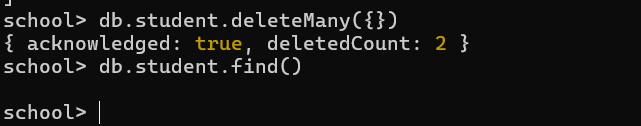
****

2.**deleteMany(filter, options):**

**Db.student.deleteMany({age:30}) or Db.student.deleteMany({condition})**

****

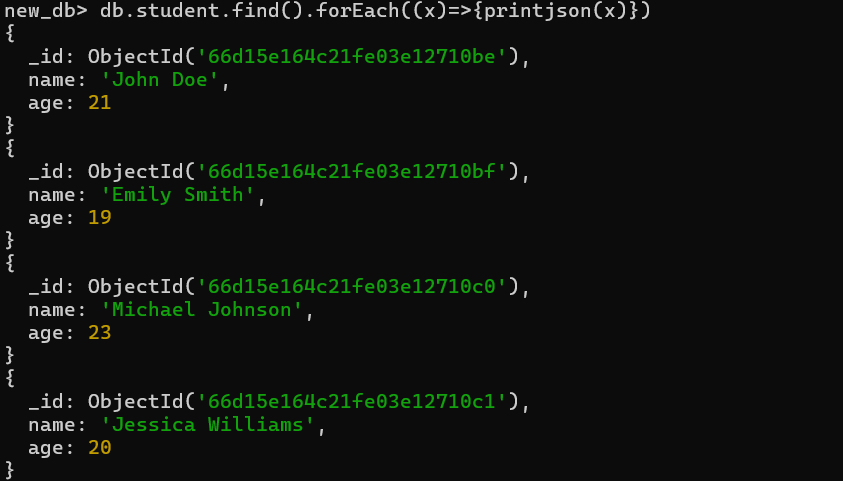
**3.deleteMany({}) , used to delete all the records from the table**

****

**Use db.[collection name],find().forEach((x)=>{printjson(x)}) , for finding all the dat in the collection if it is mor than 20 then d bitrate only 20 and we needto itrate it so use th**

**is command to itrate**

* **We can only apply this on cursor not on a single record**
* **It shows all the records**

****

**Use db.[collection name],find().count() .used t find the count of the data in the database{ number of the records }**

****

**Projections in mongoDB**

**Projection :-** **In MongoDB, projection is a feature that allows you to control which fields of a document are returned by a query. By default, MongoDB returns all fields in a document when you perform a query. However, with projection, you can specify which fields you want to include or exclude in the result set.**

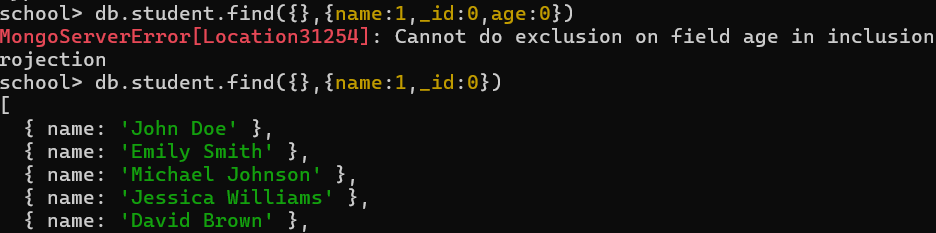
**\*only returns the particular fields in the records like only names ,only ids as per requirements**

**Db.collectionName.find({},{field to find:1}): db.student.find({},{name:1})**

**It shows only particular field and there value but in mongo db \_id is special attribute so we need to declare it so the database can exclude the id**

**Exclude the fileds by using this methods:**

**Db.student.find({},{name:1,\_id:0,~~age:0~~}) no need to declare regular attribute excluding syntax(~~age:0~~ ) it is automatic in the mongodb**

****

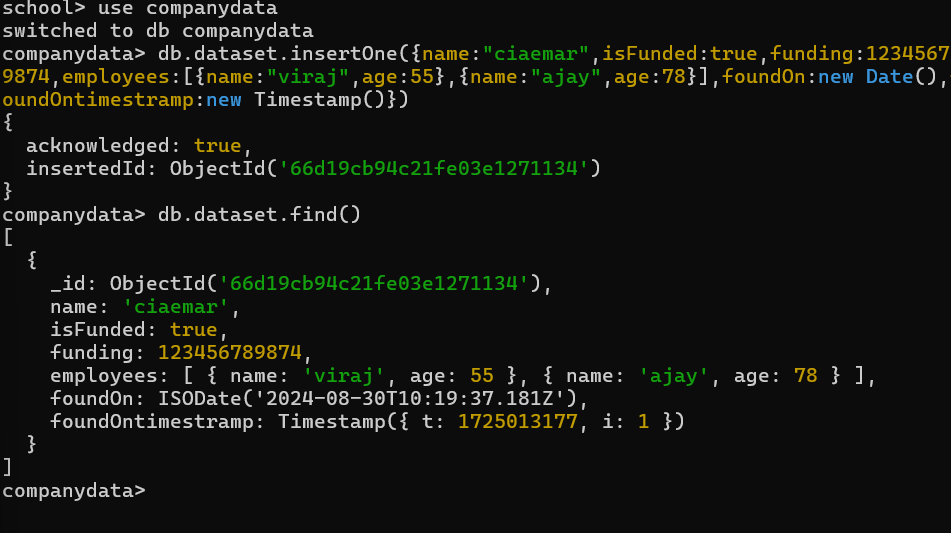
**Is mongodb is schemaless ?**

**MongoDB is schema-less, meaning it doesn't require a predefined schema. This allows for flexibility in storing documents with varying structures within the same collection. It's ideal for handling unstructured or evolving data, where fields can be added or removed without impacting the existing data.**

**We can add and remove data dynamically no need to define each field for everyone it is not mendetory it grow and shrink atomatically as per inserting elements fields**

**There is no schema in the mongo db**

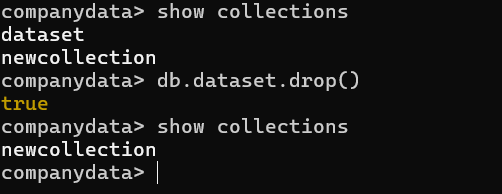
**DataTypes in MongoDB:**

* **Text -> STRING**
* **Boolean**
* **Number ->** 
  + - * **integer -> 32bits**
      * **Numberlong->64 bits**
      * **NumberDecimal**
* **ObjectID-> budefault created by mongodb**
* **Isodate -> date**
* **Timestramp -> date in milliseconds**
* **Arrays**
* **Nested document**
* ****

**Delete database:**

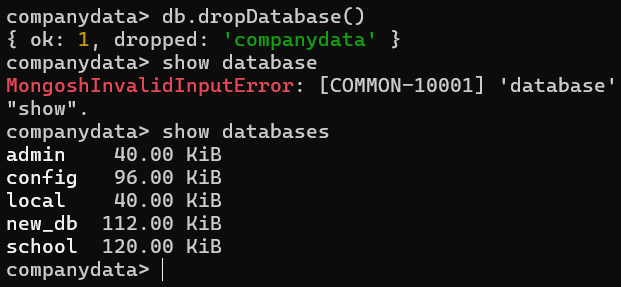
**For deleting collections :**

**Db.[collectionname].drop() ,use to drop the collection in the database**

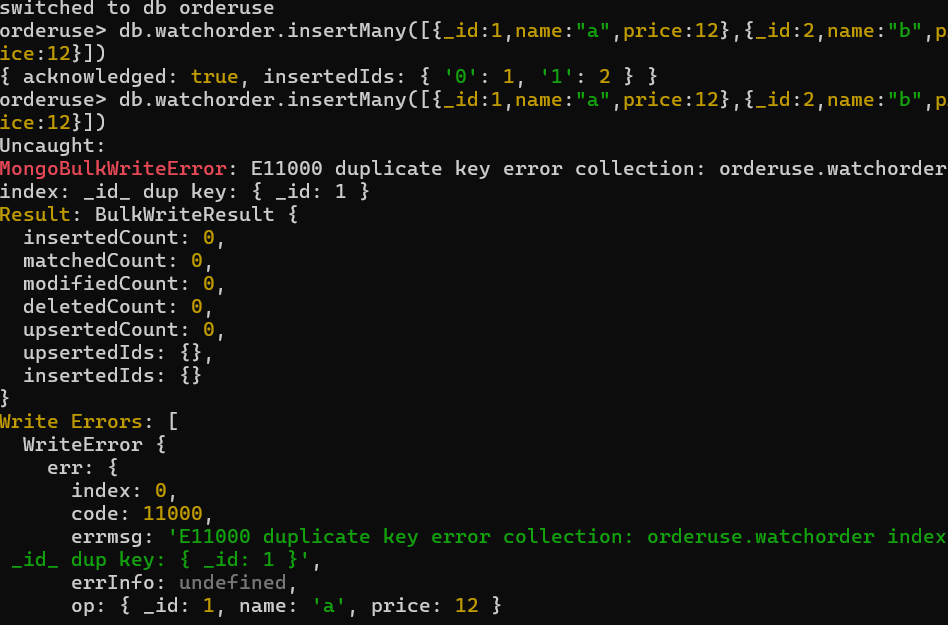
****

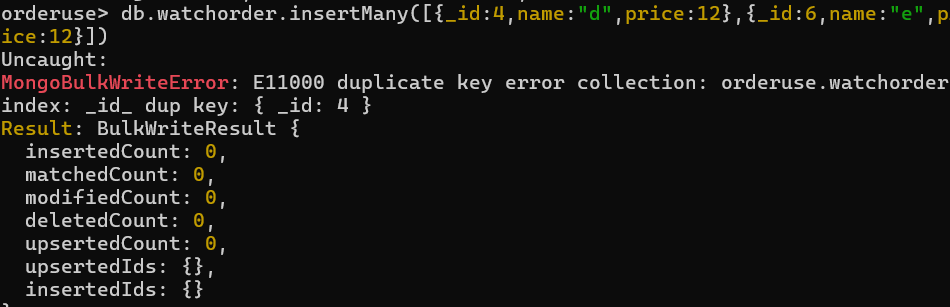
**for deleting database :**

**db.dropDatabase(),drop whole database including all the data inside the database**

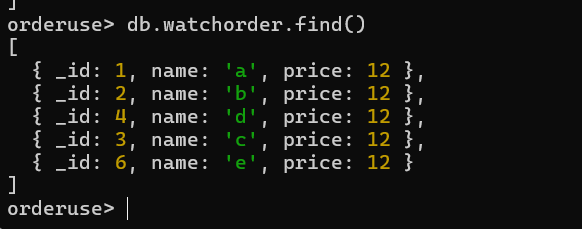
****

**Insert options({},{order:false})**

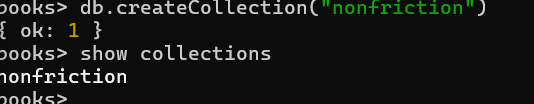
****

**Because of error next element also not get inserted **

**Use order db.[collection name].insertMany** **([{\_id:4,name:"d",price:12},{\_id:6,name:"e",price:12}],{ordered:false})**

****

Still we get error but because of {ordered:false} remaining element is pushed into database rather than remaining one

* **Schema validation in mongo db :**
* **Creating empty collection:**
* ****

**In MongoDB, validation allows you to enforce a specific schema on documents within a collection, even though MongoDB is schema-less by default. This is done using a schema validation rule that specifies the required structure, types, and constraints for fields.**

**db.createCollection("students", {**

**validator: { $jsonSchema: {**

**bsonType: "object",**

**required: ["name", "age"],**

**properties: {**

**name: {**

**bsonType: "string",**

**description: "must be a string and is required"**

**},**

**age: {**

**bsonType: "int",**

**minimum: 18,**

**description: "must be an integer >= 18 and is required"**

**}**

**}**

**}}**

**});**

**db.createCollection("nonfictional", {**

**validator: { -> it is field used to to add a validator in a database while adding data for example require field while inserting a data**

**$jsonSchema: { -> define a jsonschema abd required field**

**required: ["name", "price"], -> enter a require field if not enterd while data insertion then error occure**

**properties: { -> set properties like type and description**

**name: {**

**bsonType: "string",**

**description: "must be a string and is required"**

**},**

**price: {**

**bsonType: "number",**

**description: "must be a number and is required"**

**}**

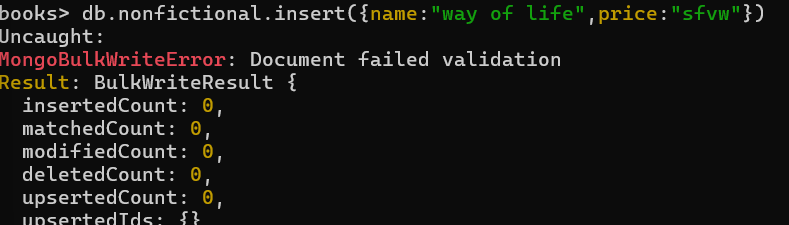
**}**

**}**

**},**

**validationAction: "error" -> action if data is not fulfilled or valid**

**})**

****

db.runCommand({

collMod: "nonfictional", -> for modify the database scemas

validator: {

$jsonSchema: {

required: ["name", "price", "author"],

properties: {

name: {

bsonType: "string",

description: "must be a string and is required"

},

price: {

bsonType: "number",

description: "must be a number and is required"

},

author: {

bsonType: "object",

required: ["firstName", "lastName"],

properties: {

firstName: {

bsonType: "string",

description: "must be a string and is required"

},

lastName: {

bsonType: "string",

description: "must be a string and is required"

}

}

}

}

}

},

validationAction: "error"

})

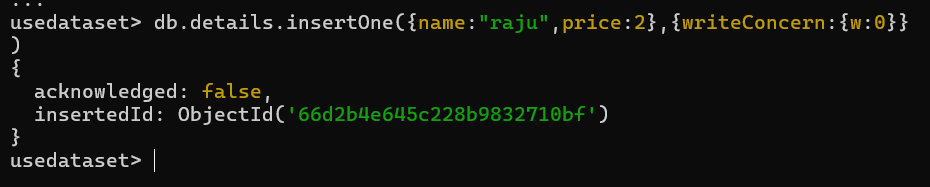
###### Write concern specification

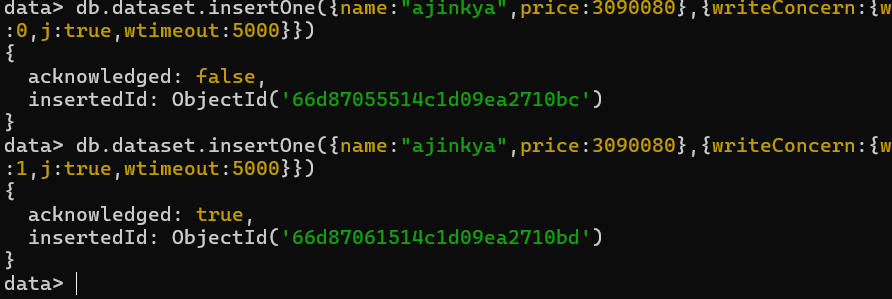
{w:<value>, j:<boolean>,wtimeout<number>}

**W :is a acknowledgement 1 for yes 0 for no**

**J: j is a jurnal it write the current oprations inside the gernal in case of pwercut it used to retrive the last state of the database and continue opration by default j is false**

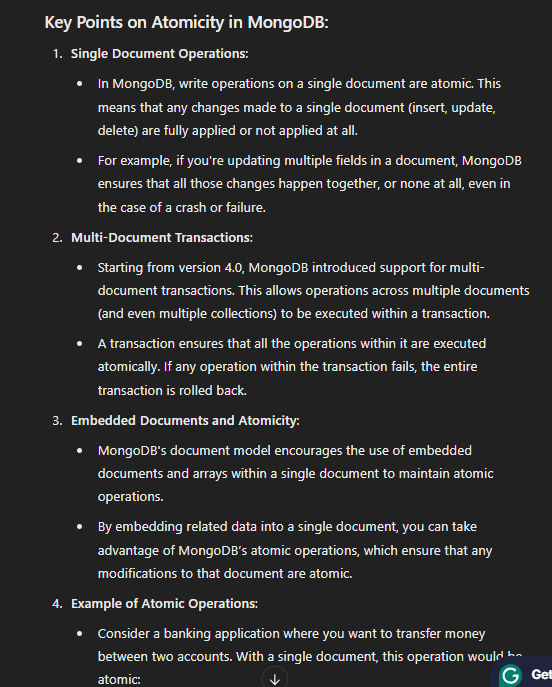
**Wtimeout:set timeout for opration discontinue opration after pertiular amount of time**

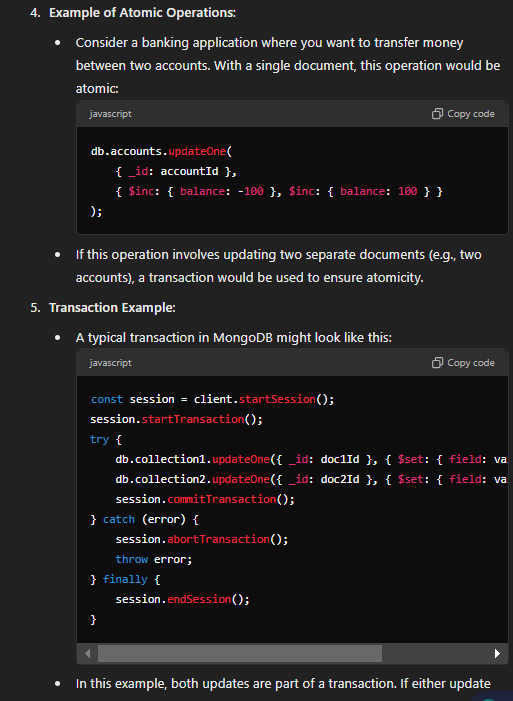
****

****

**Automicity in mongo db:**

Atomicity in MongoDB refers to the guarantee that a series of operations within a single unit of work are treated as a single operation, meaning they either all succeed or all fail. In other words, if any part of the operation fails, the entire operation is rolled back, leaving the database in its original state. This ensures data consistency.





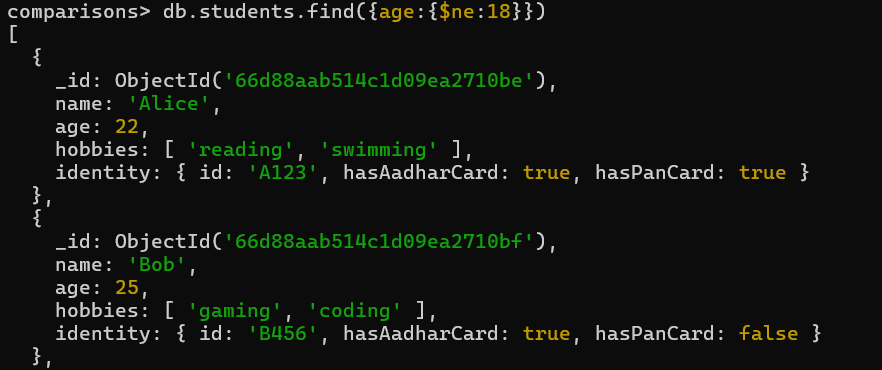
**Comparision oprators in mongodb:**

****

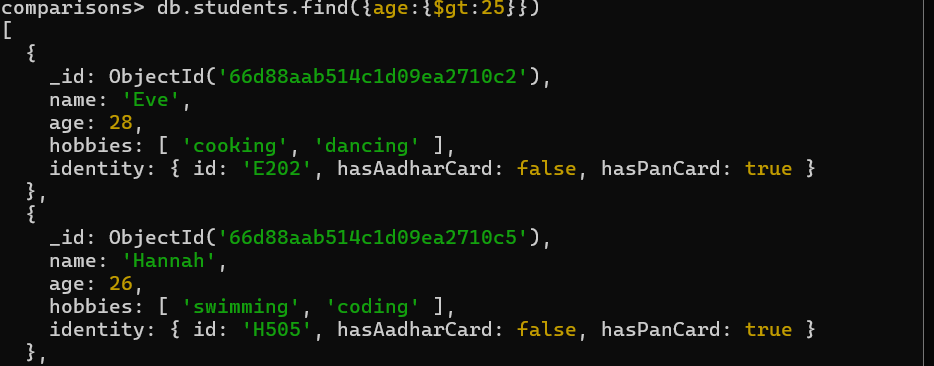
**1. $eq (Equal)**

* **Description: Matches documents where the value of a field equals the specified value.**
* ****

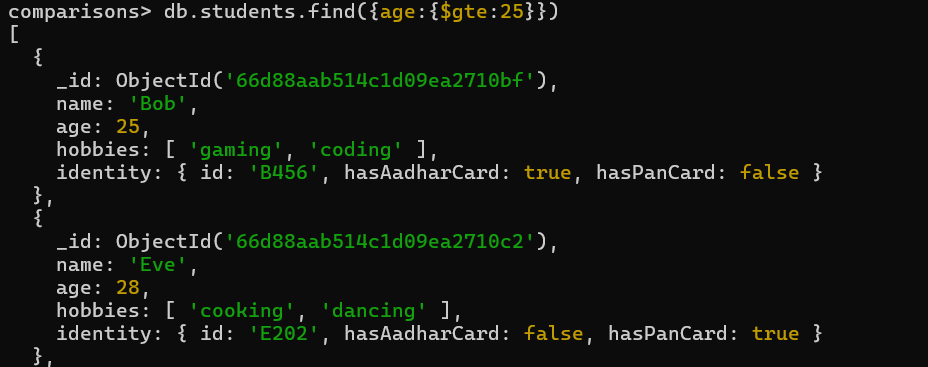
**2. $ne (Not Equal)**

* **Description: Matches documents where the value of a field does not equal the specified value.**
* ****

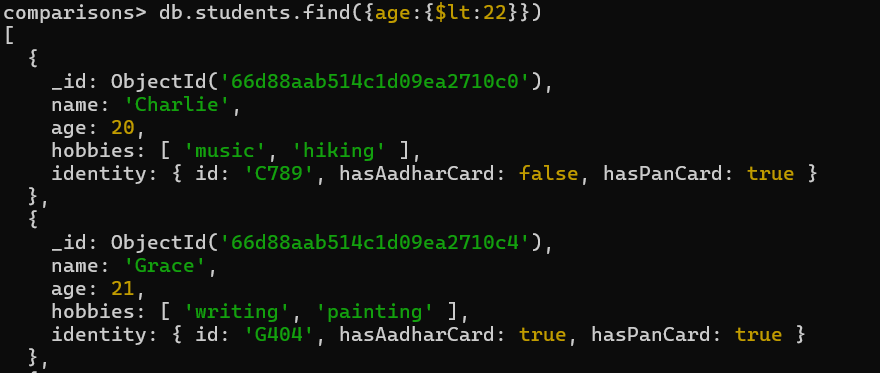
**3. $gt (Greater Than)**

* **Description: Matches documents where the value of a field is greater than the specified value.**
* ****

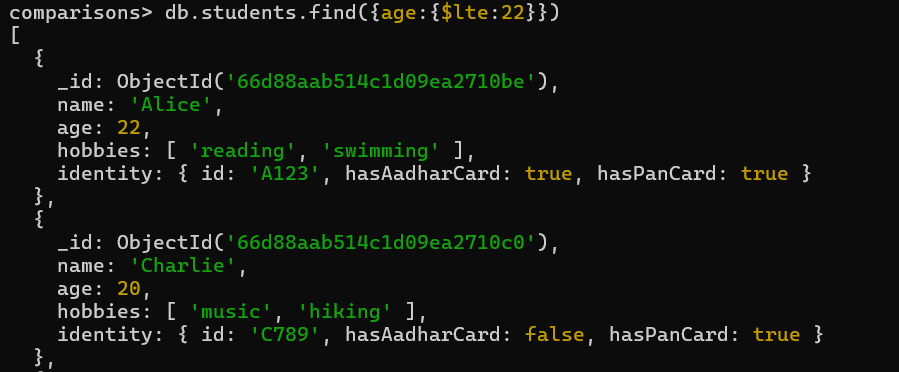
**4. $gte (Greater Than or Equal)**

* **Description: Matches documents where the value of a field is greater than or equal to the specified value.**
* ****

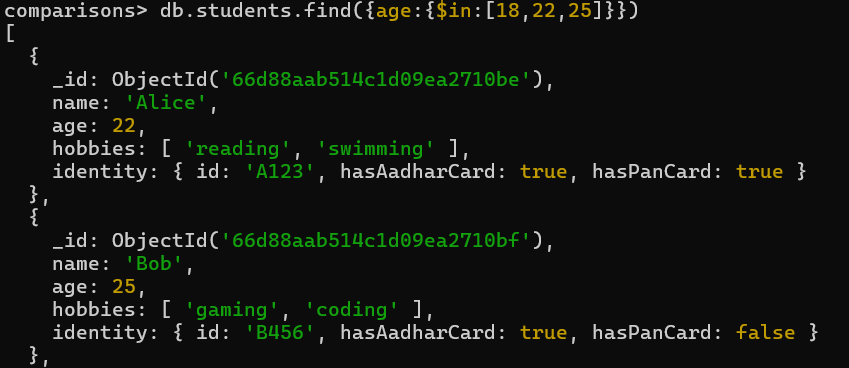
**5. $lt (Less Than)**

* **Description: Matches documents where the value of a field is less than the specified value.**
* ****

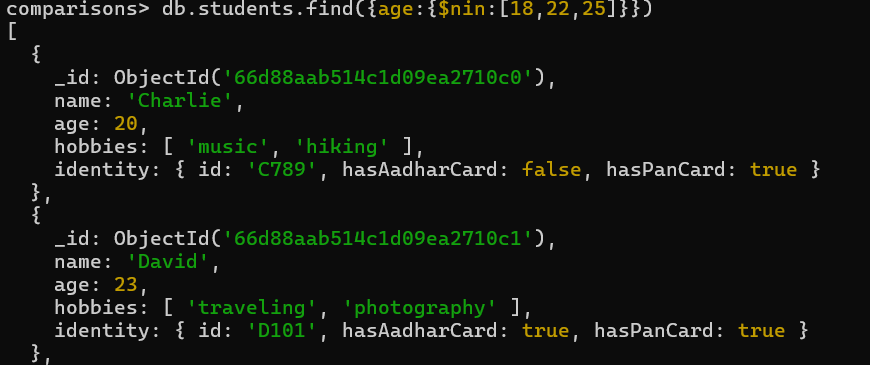
**6. $lte (Less Than or Equal)**

* **Description: Matches documents where the value of a field is less than or equal to the specified value.**
* ****

**7. $in (In)**

* **Description: Matches documents where the value of a field equals any value in the specified array.**
* ****

**8. $nin (Not In)**

* **Description: Matches documents where the value of a field does not equal any value in the specified array.**
* ****

**9. $exists (Exists)**

* **Description: Matches documents where the specified field exists (or does not exist).**

**10. $type (Type)**

* **Description: Matches documents where the value of a field is of the specified BSON type.**

**Combining oprators :**

**db.collection.find({**

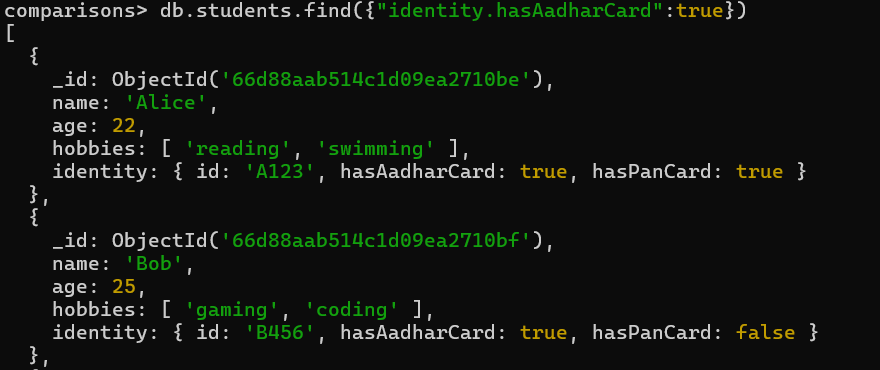
**$and: [**

**{ age: { $gte: 18 } },**

**{ age: { $lte: 30 } }**

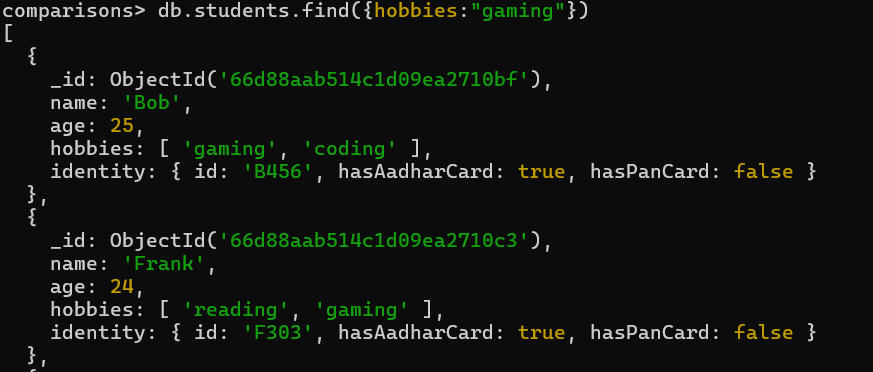
**]**

**})**

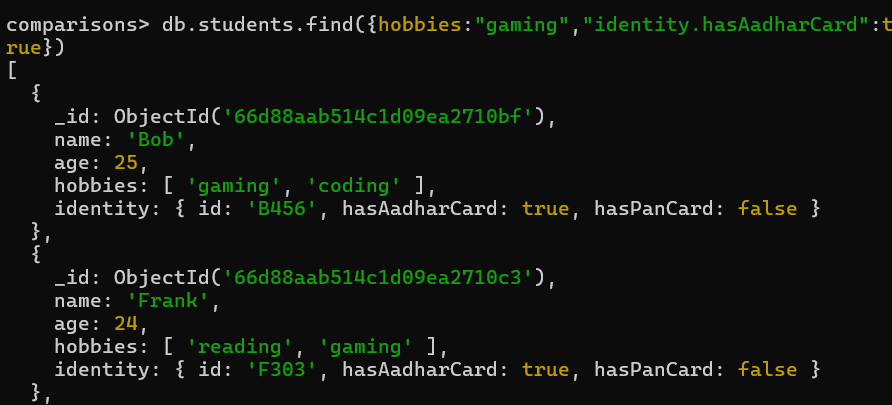
****

**Showing all the candidate with having aadhar card**

**For finding the particular value in list of particular attribute**

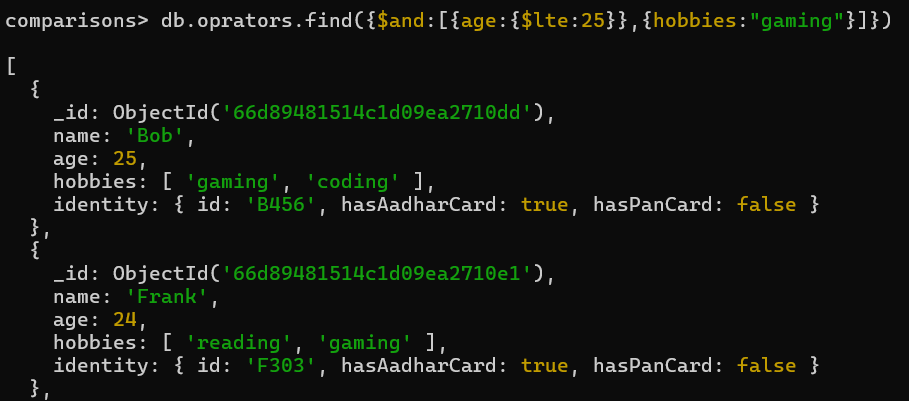
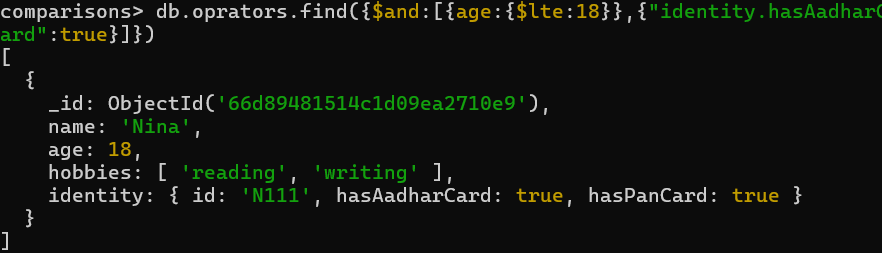
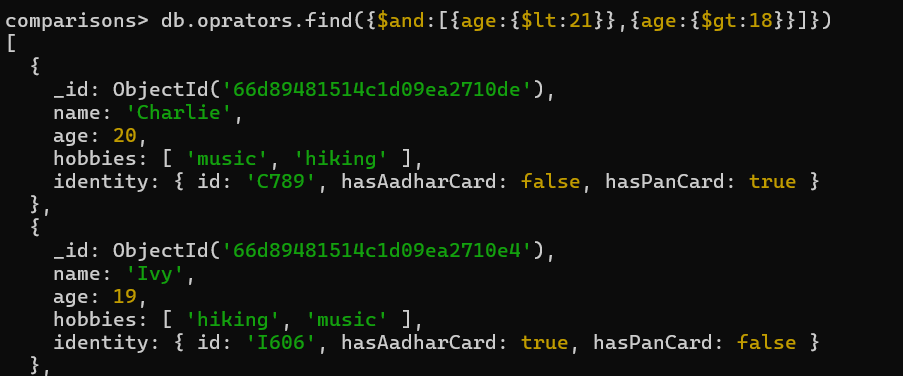
* **Db.collection.find({hobbies:gaming}) so it will retrive all the students with hobby gaming**
* ****

**Combining a both queries**

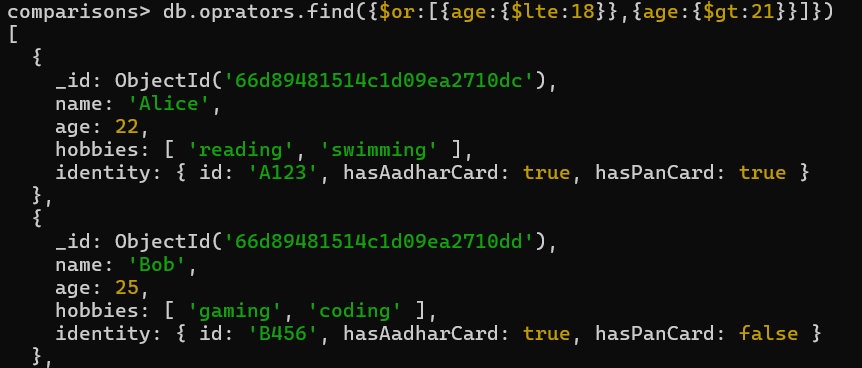
****

**Oprators:**

**1. $and**

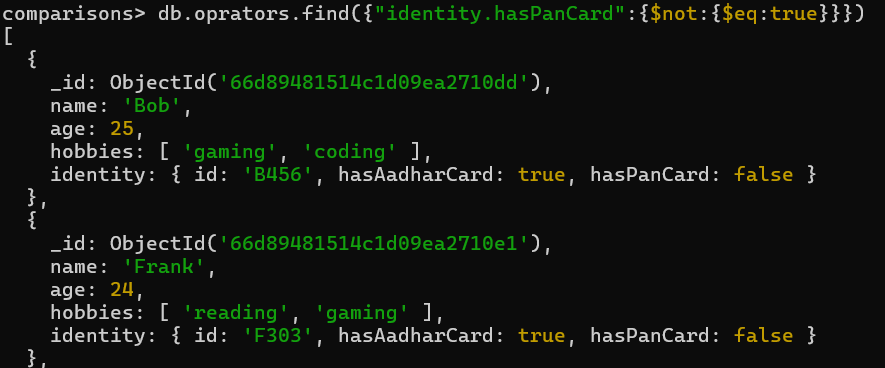
* **Description: Combines multiple conditions, and all conditions must be true for a document to match.**
* **Must both conditions are true**
* ****
* ****
* **\*Condition in json if we directly use two keys similar keys as a codition then last condition will apply as a filter**
* ****

**2. $or**

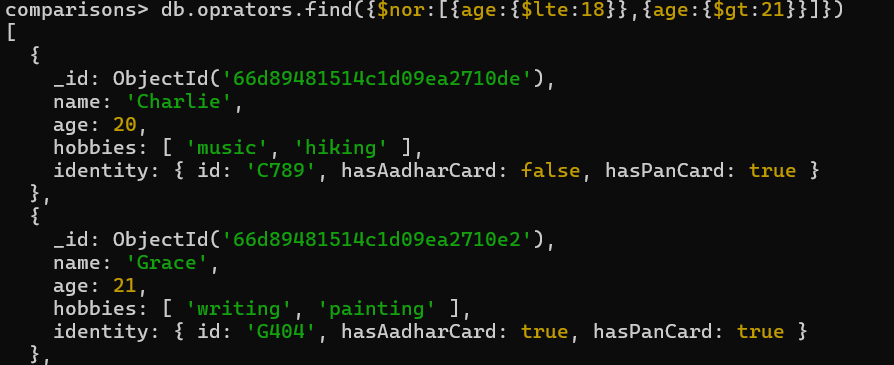
* **Description: Combines multiple conditions, and at least one of the conditions must be true for a document to match.**
* ****
* **Db.oprators.find({$or[{age:{lte:18}},{age:{gte:21}}]})**

**3. $not**

* **Description: Inverts the effect of a query expression. It matches documents where the specified condition is false.**

****

**4. $nor**

* **Description: Combines multiple conditions, and none of the conditions must be true for a document to match.**
* **Rather than all the documents those matches the condition**
* ****
* **elements query oprators in mongodb**

**5. $exists**

* **Description: Checks for the existence of a field within documents.**

**2. $type**

* **Checks the BSON data type of a field.**

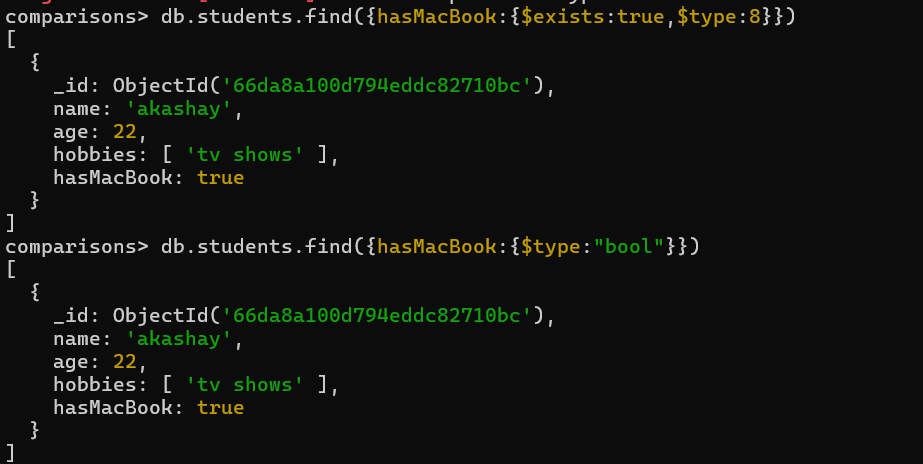
**Syntax:**

**{ field: { $type: <BSON type> } }**

**You can use either type names (like "string", "int") or type numbers (like 2 for string, 16 for int).**

**Common BSON types:**

* **1 - Double**
* **2 - String**
* **3 - Object**
* **4 - Array**
* **8 - Boolean**
* **10 - Null**
* **16 - Int32**
* **18 - Int64**

****

**3. $size**

* **Matches arrays if their length equals the specified size.**

**Syntax:**

**{ field: { $size: <number> } }**

**db.collection.find({ tags: { $size: 3 } })**

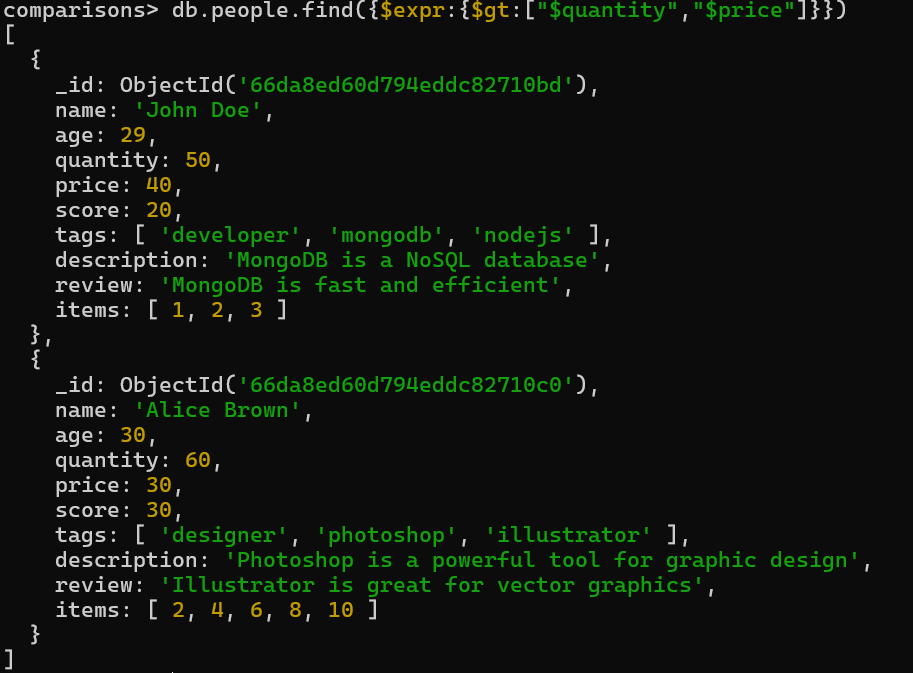
**evaluaton query oprations**

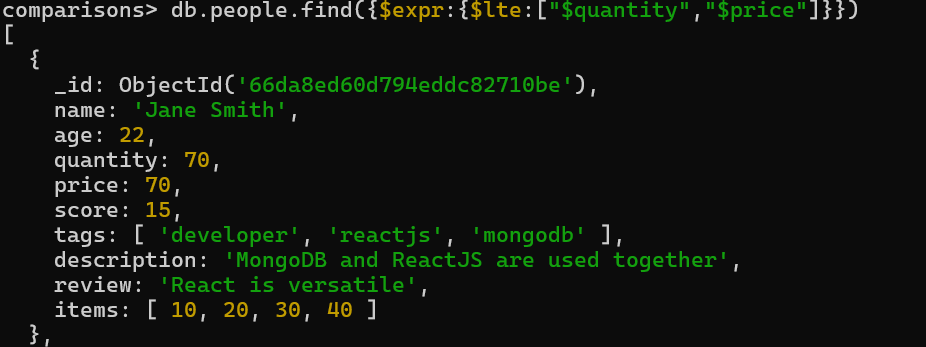
**In MongoDB, evaluation query operators are used to execute expressions or conditions on data. These operators are often used when you need to apply complex conditions or custom logic during a query. The main evaluation operators in MongoDB are:**

**1. $expr**

* **Allows the use of aggregation expressions within the query language.**
* **You can compare fields, apply arithmetic operations, or evaluate other conditions using this operator.**

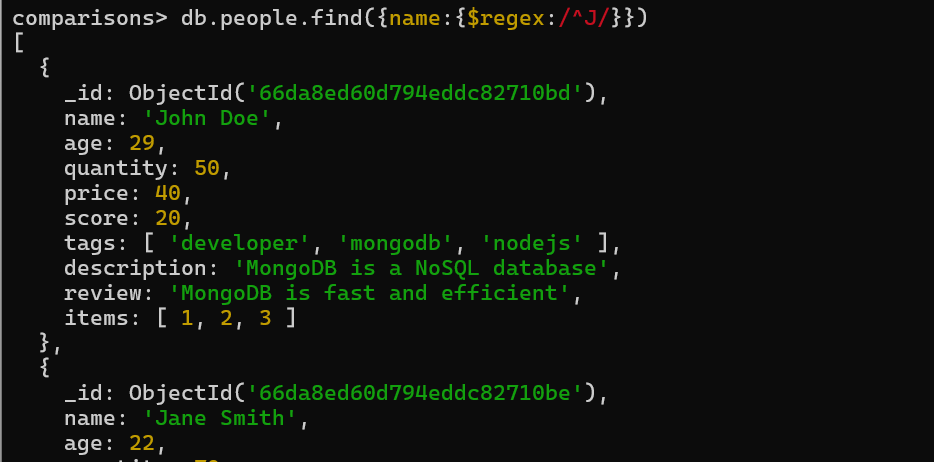
**Syntax:**

****

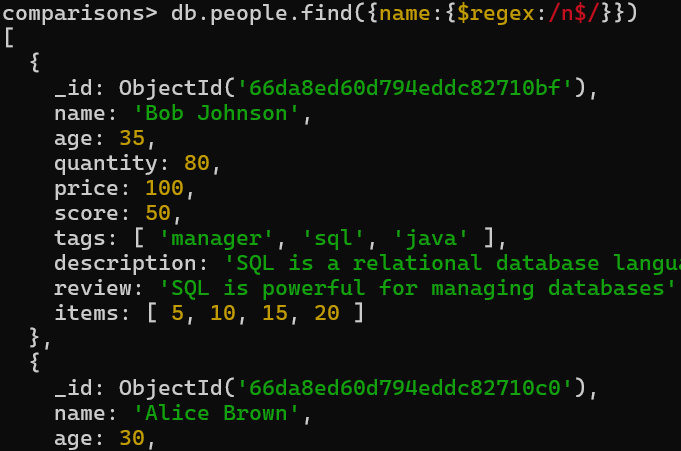


**$regex Example:** Find documents where the description contains the word "MongoDB":

* The ^ anchor ensures that the string starts with "john".
* With the i option, it will match both "John" and "john" (or any other case variations).

1. db.collection.find({ name: { $regex: /john/i } })
2. 

* The $ anchor ensures that the string ends with "example.com". The \. is used to escape the dot (.), which is a special character in regex.

1. db.collection.find({ email: { $regex: /example\.com$/ } })
2. 

* Find documents where the bio field contains "JavaScript" in a multi-line string:

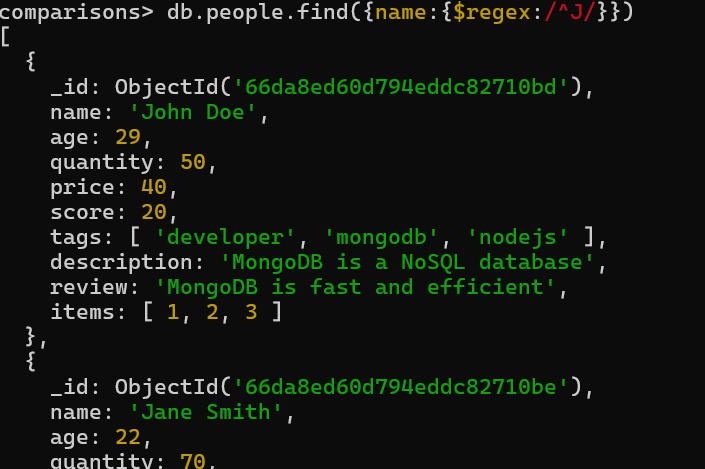
1. db.collection.find({ bio: { $regex: /JavaScript/m } })

**validation purpose regex:**

**numric value: db.collection.find({ name: { $regex: /^[0-9]+$/ } })**

**character validation db.collection.find({ email: { $regex: /^[a-zA-Z0-9\_]+$/ } })**

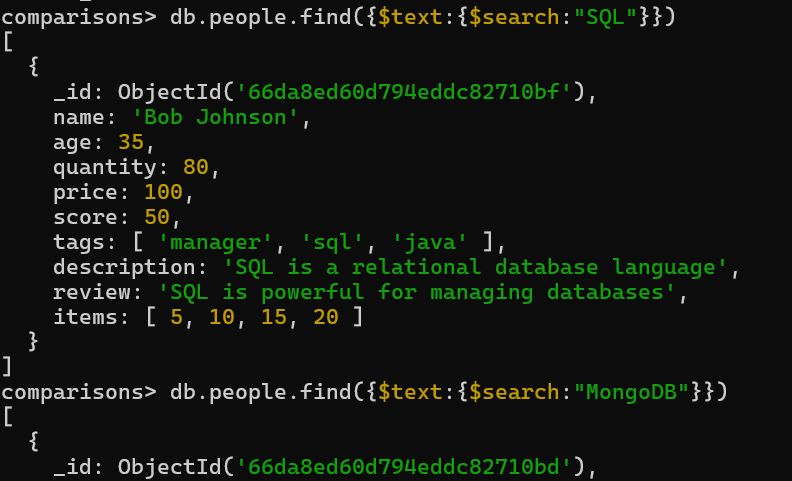
db.people.find({ description: { $regex: /MongoDB/ } })



**$text Example:** Assuming a text index on description and review, find documents that mention "React":

db.people.createIndex({ description: "text", review: "text" })

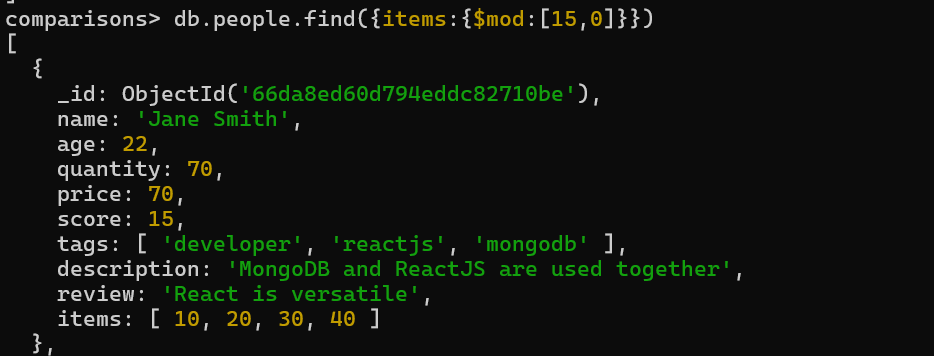
db.people.find({ $text: { $search: "React" } })

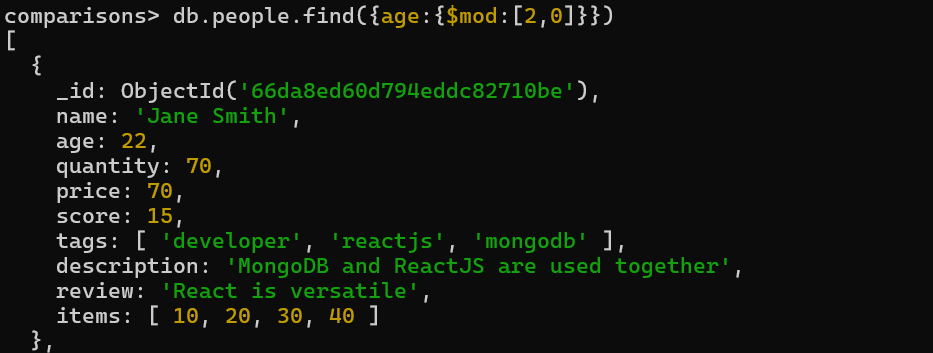


**2. $mod**

* Performs a modulus operation and matches documents where a field's value is divisible by a given number.

{ field: { $mod: [divisor, remainder] } }

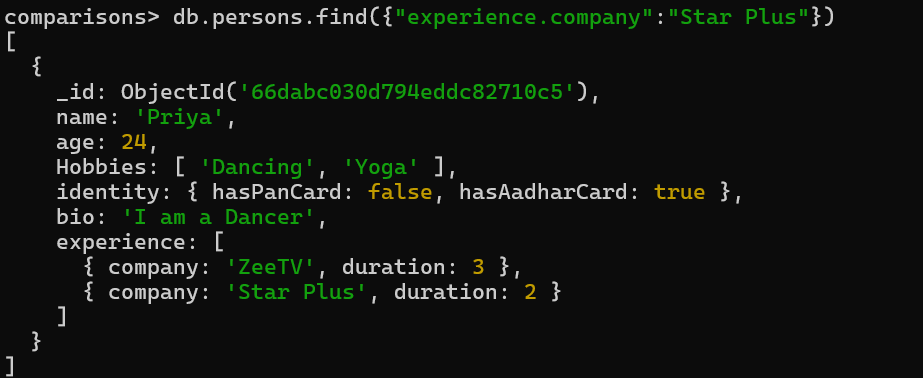




Random questions to solve:

How to search in the nested document:

Db.collection.find(“attribute.innerattribute”:”value”)

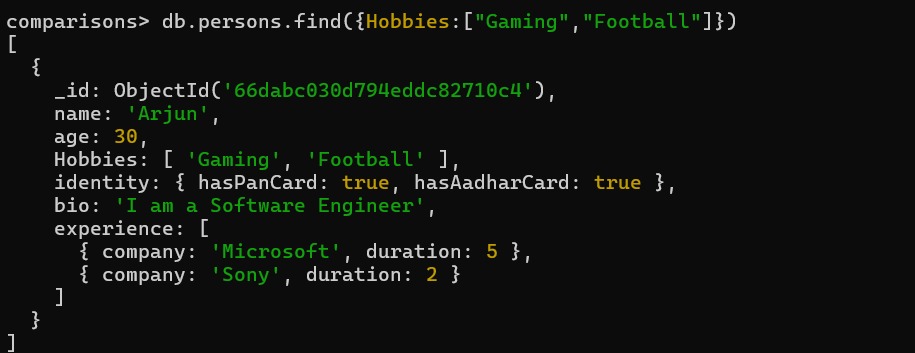


Finding the attribute with pweticular size:

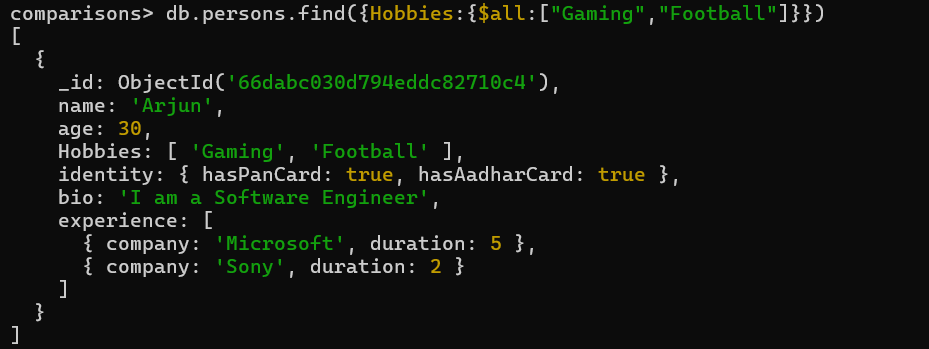


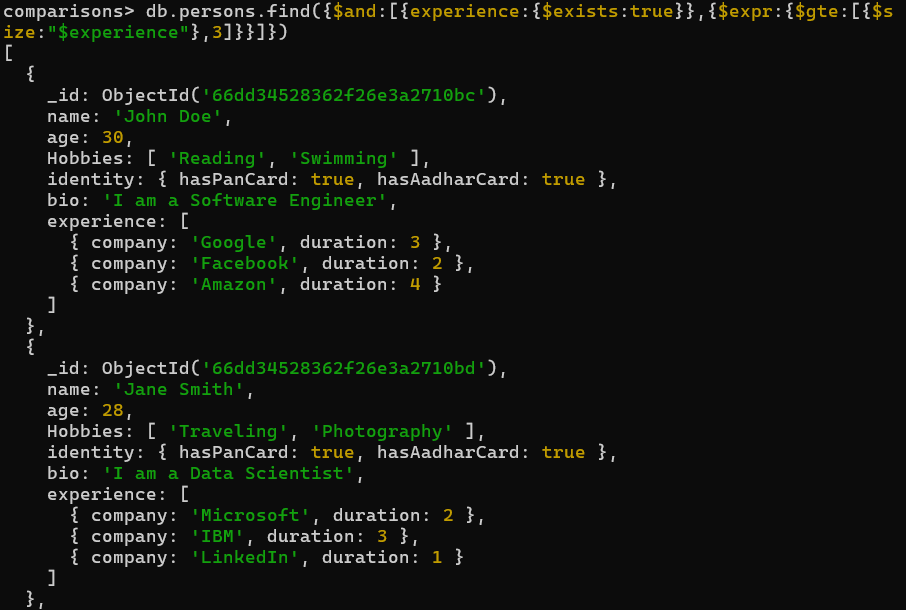


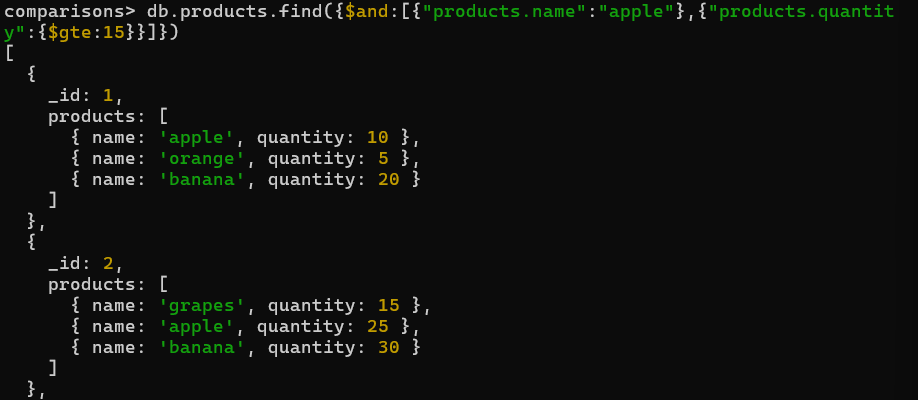
Finding perticular instance like in hobbies for perticular hobbies



We use this query for searching all the instences in the attributes even if they are not following same order but present in the attribute ::We use $all keyword







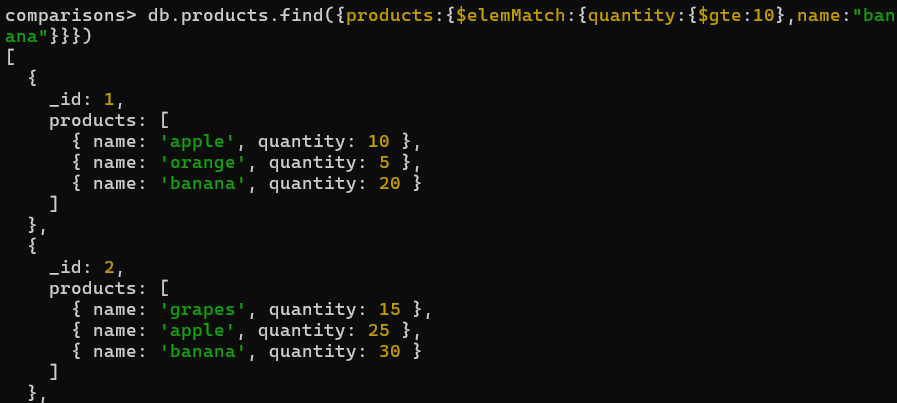
Finding elements with specific Attribute (regular ,method) in basic method there is A problem when we serch for particular element and value if similar value found then it will show the result which is not true for

Eg:as per query only records will be shown which contain name:apple and respective quantity is 15 but in this exam[le it shows elemnt event if quantity is not matching cause it only watch as a product prespective e=required elements in product data and requide quantity in product not by each element respective



Finding elemets with “elemMatch”

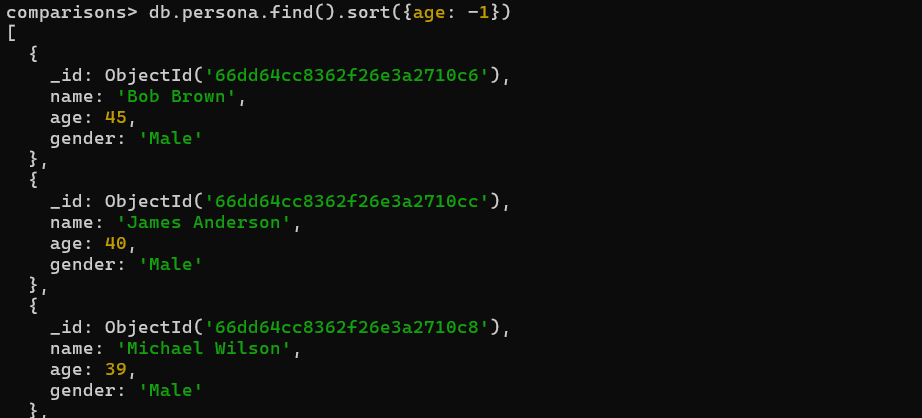
Search by element and with respective elements



**Sorting in mongodb**

**We can sort a dataset with use of sort opration it takes a json file and create a sorted dataset 1 for assending and -1 for desending**

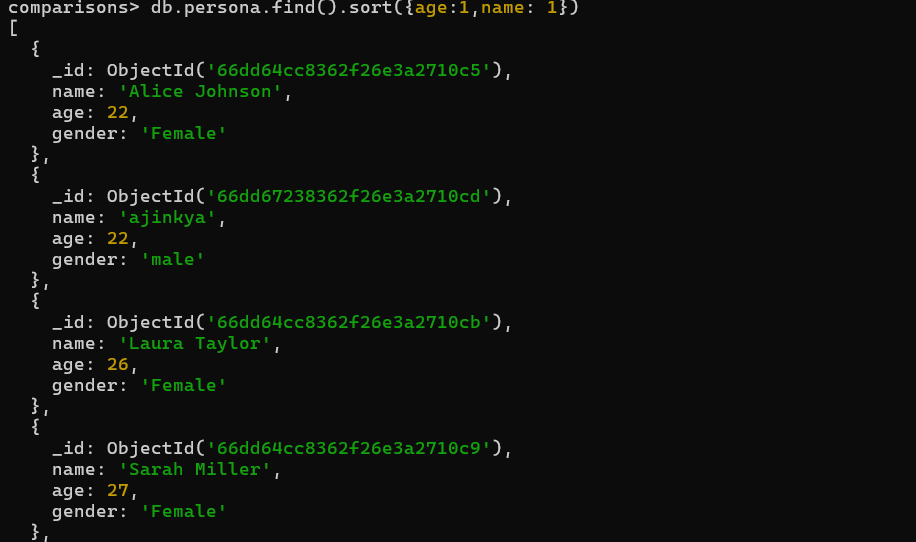
****



For name same syntax is used



Nested sorting for example if two persons having a same age then sort with name again between those peoples

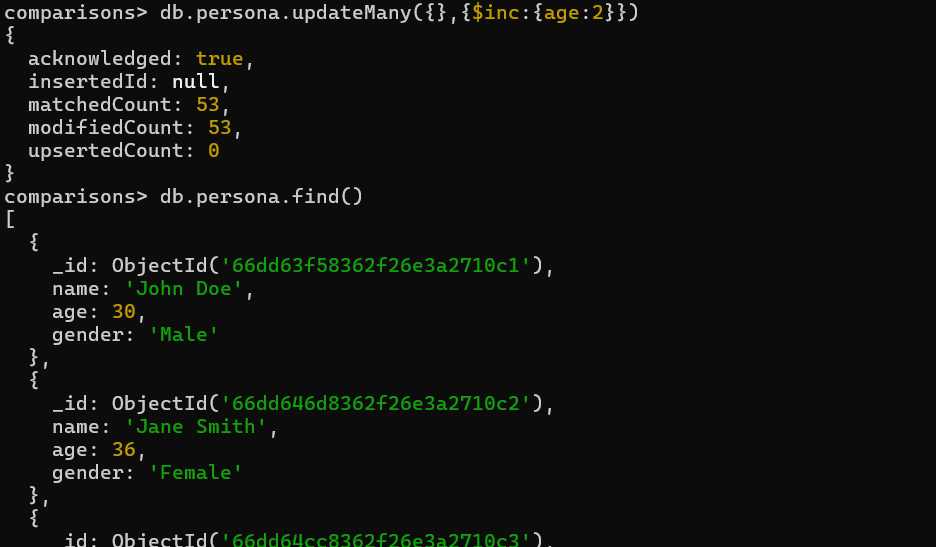


To show all the element we use forEach loop

Syntax: (😊 db.collectionname.find().sort({attribute:1/-1}).forEach(x =>printjson(x)))

Advanced updates

$inc oprator

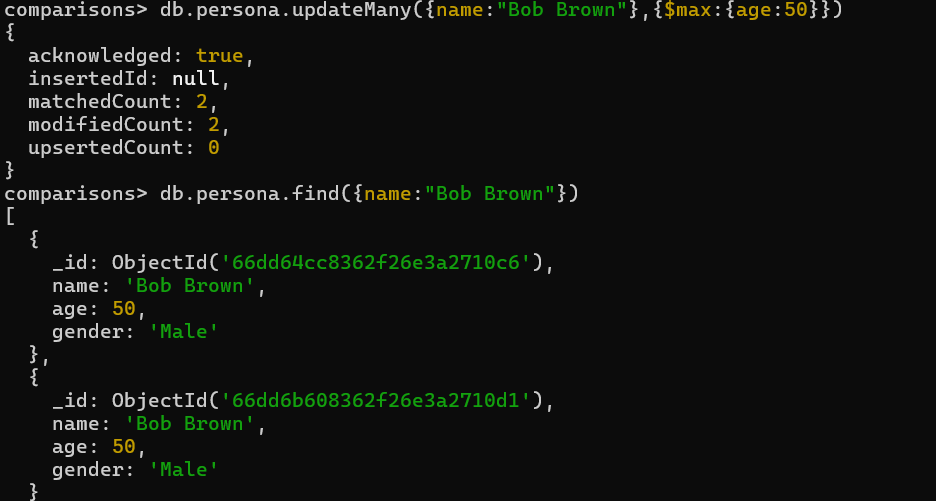


Used to increment a value of attribute

Syntax is ( db.persona.updateMany({},{$inc:{age:2}}))

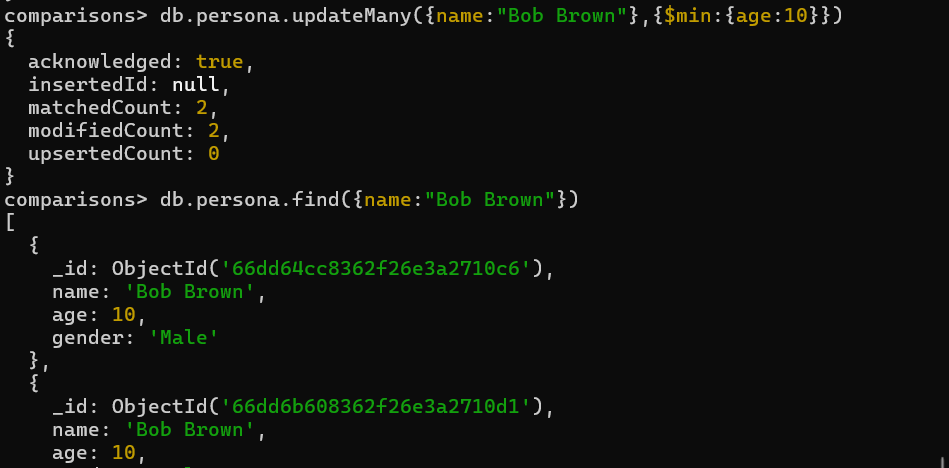
$Min and $max oprator :

$max :is used to increase the value of attribute it will only work if current value is less than applied value if the current value is :50 and we use max : 30 then it will not be applied it shows original value in that case

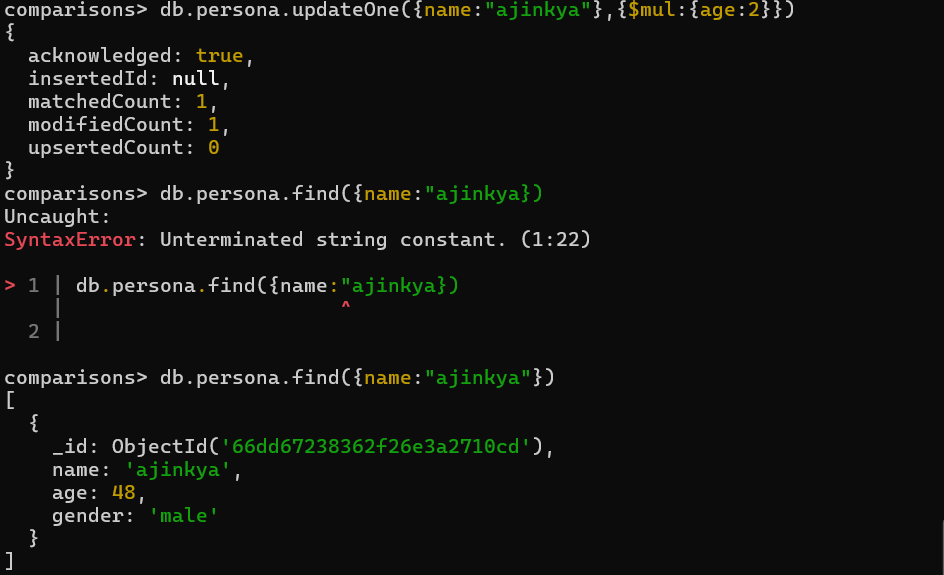


$min :

This is same as a max but viceversa if updated value is max than current value then it still remains same no change for eg :current value is 23 and updated value is 34 then min will not consider that value



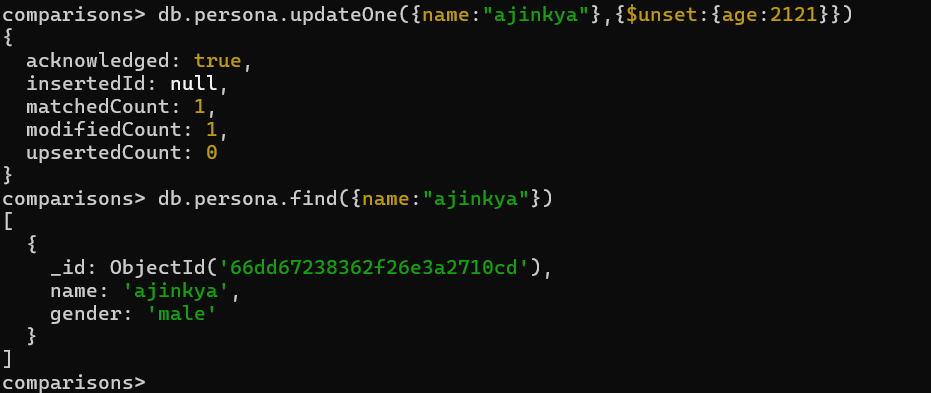
$mul: multiply the values in this case value of age is multiply by 2 cause we set age :2



$unset:

Used to remove the field from the particular table or particular dataset it removes the particular attribute from the table or particular dataset

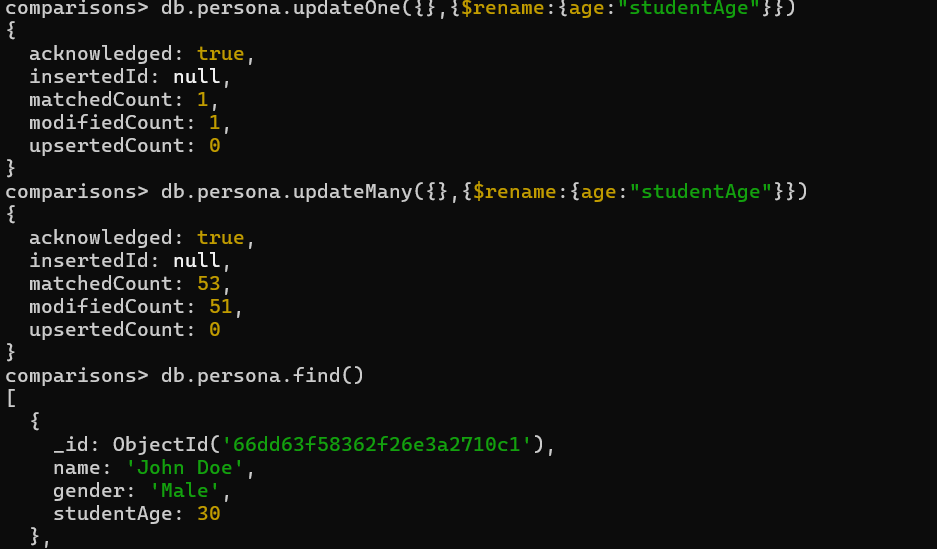
* In current scenario it removes the age attribute from the table or value



$rename:

Rename is used to rename the attribute or values

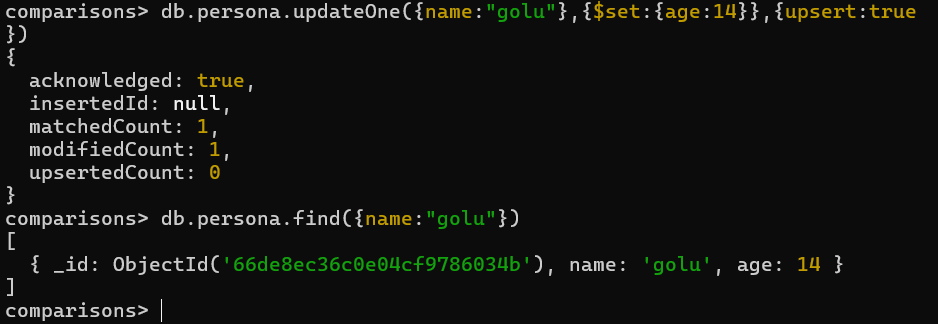
We can change attribute name or their value for eg :we change age:studentAge

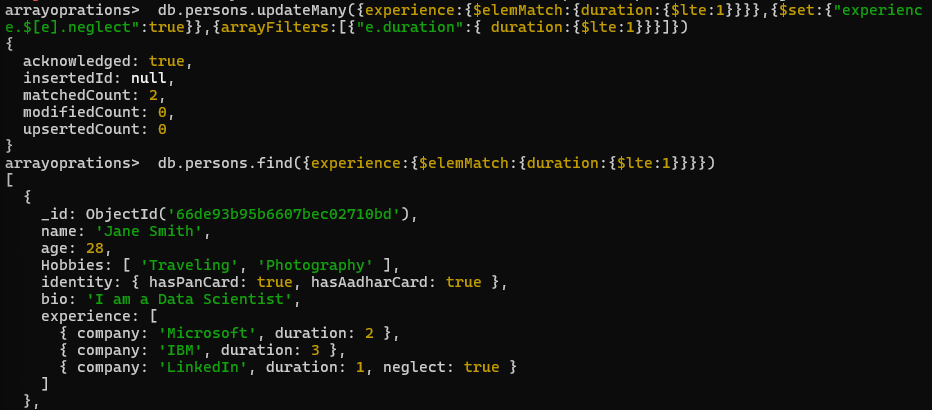


$upsert:true:

It is used to cheak the current dataset is in the list or not if not then insert this dataset as a new element or new dataset

* In current scenario we search for dataset named golu but it is not in the database so it create that dataset
* We use upsert:true for creating an element



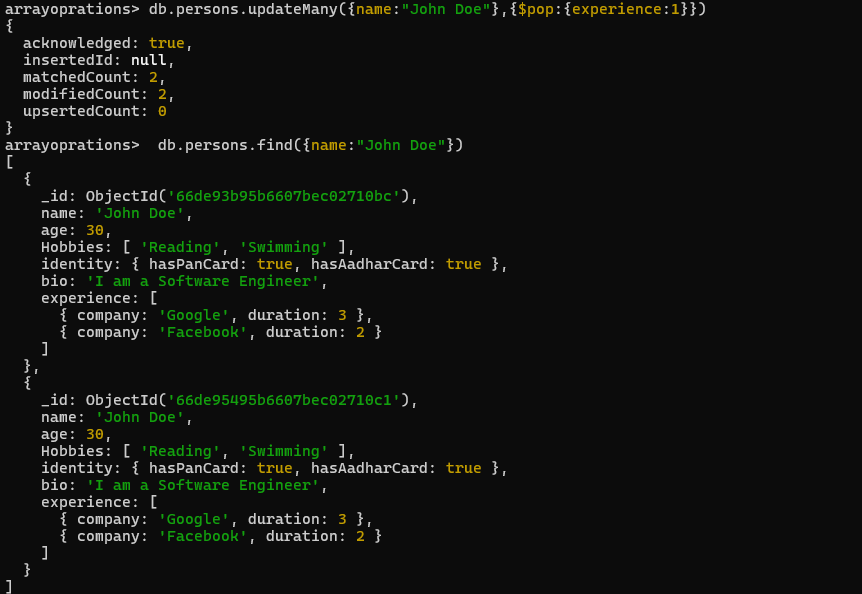


$Push oprations in the array used to push the data into a array or in a dataset 

$Pull: pull oprations used to remove the element from the dataset



$pop : pop the last element from the data attribute or in the array or datasets



Indexing in the mongodb :

When a query is executed mongoDb can use the index to quickly locate the documents that match the query by searching through the B-tree

Indexing in MongoDB is a mechanism that improves the efficiency of query operations. By default, MongoDB creates an index on the \_id field, but you can create additional indexes on other fields to optimize queries.

**Single Field Index**:

* This is the most common type of index. It is created on a single field of the document, like name or age.

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

**Compound Index**:

* A compound index is created on multiple fields to support more complex queries.

db.collection.createIndex({ name: 1, age: -1 });

**Multikey Index**:

* Used when the field contains an array. MongoDB creates separate index entries for each element in the array.

db.collection.createIndex({ tags: 1 });

Managing Indexes:

Listing indexes

db.collection.getIndexes();

Dropping an index

db.collection.dropIndex({ name: 1 });

Explain() :

Syntax is :db.collectionName.find({field:”value”}).explain()

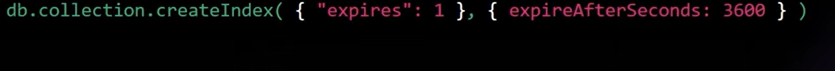
 index attributes unique identify while sortiong



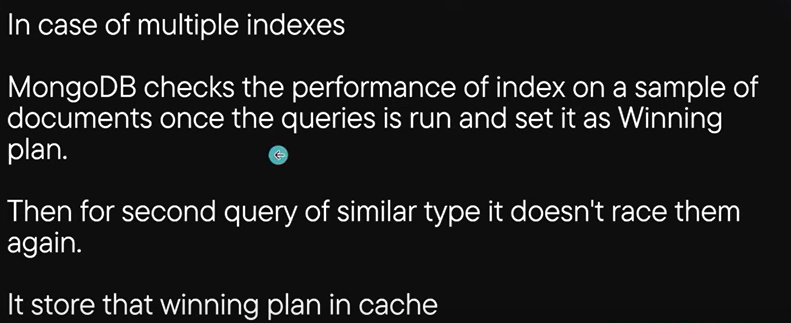
Partial index is the type of expression were only document who matches the second condition documents are sorted

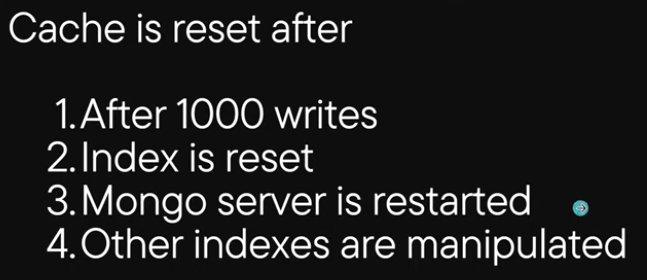


Expires field is only work on date field and single field index







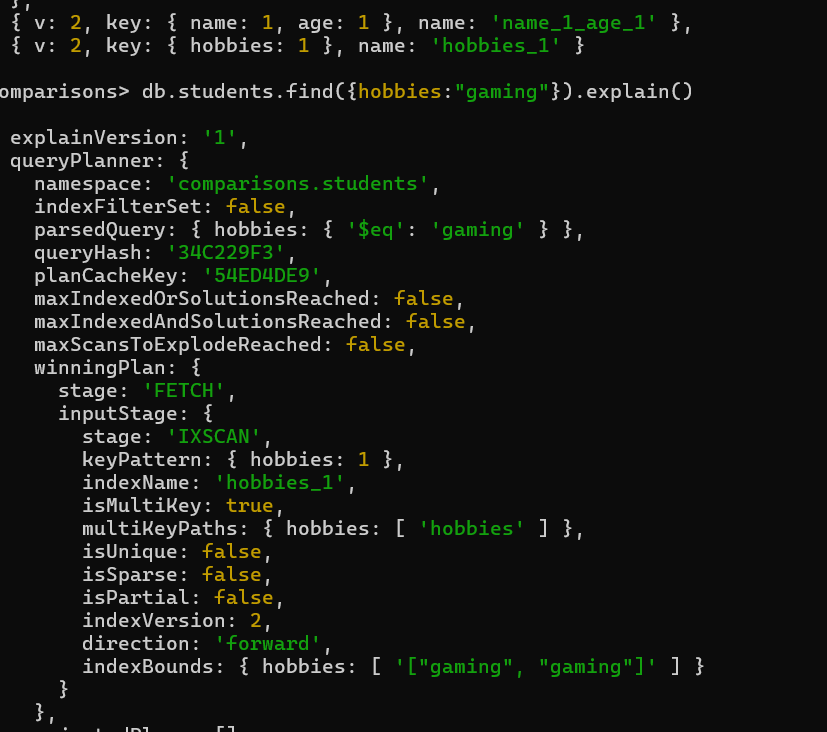


Multikey index:

MongoDB will crate a seprate index entry for each value in each array

So it quickly look up documents that matches specific value

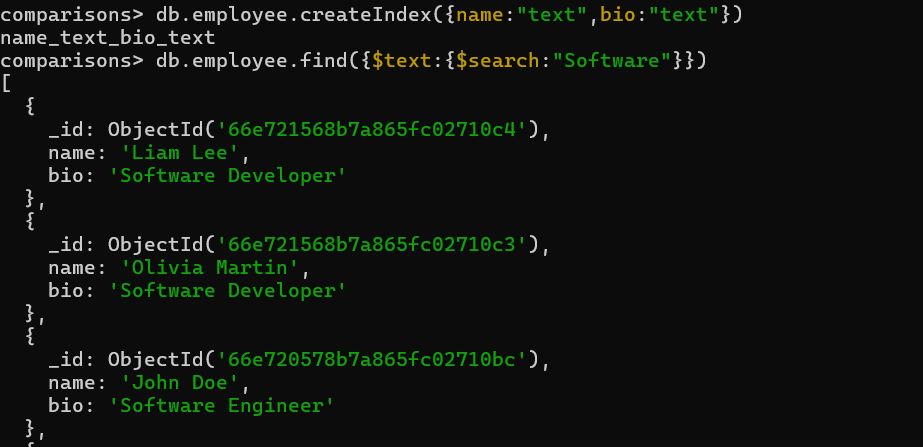
It takes lot of space cause it createsa entry for EACH VALUE in the array

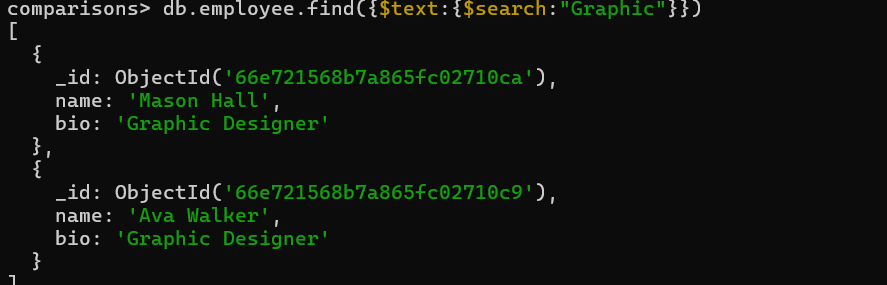


**Text index:**

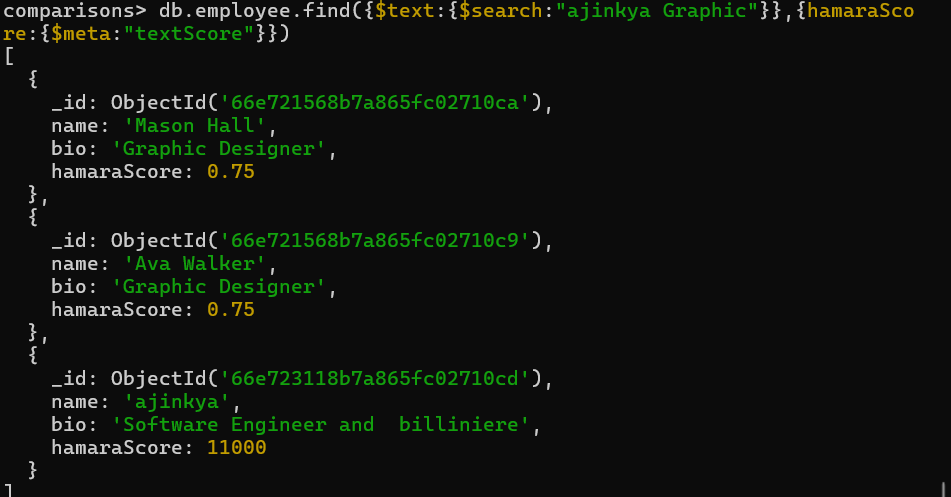
**Search all the records with similar keyword in the text field we need to create a index first to search in text index**

**We can only create a single text index in one index**

****

****

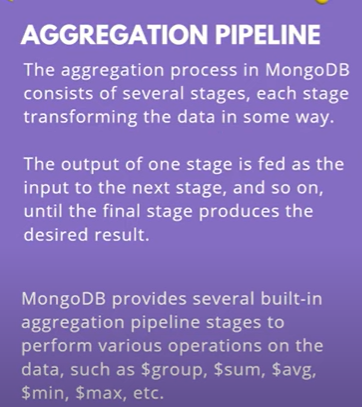
****



Aggregation in mongodb:

Query used :(db.collection.aggregate(pipeline,options))

Aggregation in MongoDB is a powerful framework used to process and analyze data by transforming and filtering documents in a collection. It allows you to perform operations like grouping, filtering, sorting, reshaping, and calculating aggregated values (e.g., sum, average) on data. MongoDB provides an aggregation pipeline where you can pass documents through multiple stages, each performing a specific operation.



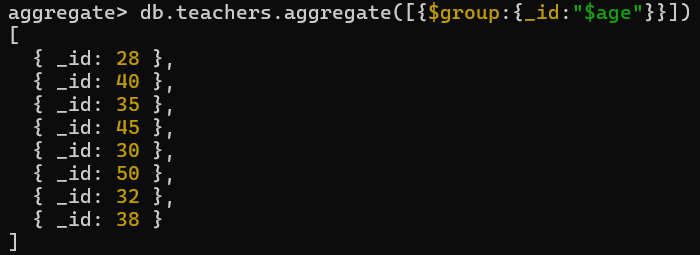
>**match function in the aggregate function**



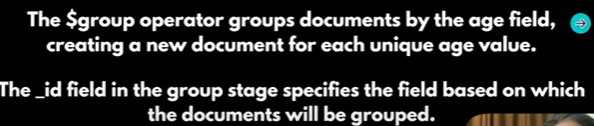
>group oprator

Query:db.teachers.aggregate([{$group:{\_id:”age”}}])

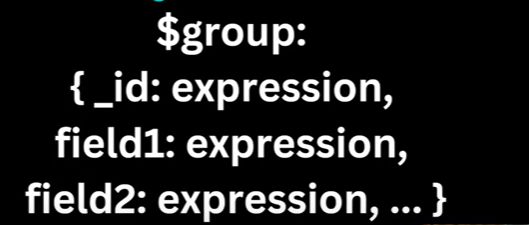
This query create a group of teachers with same age for each age group



This is the group of the teachers in each id there is a number of teachers with same age basically it is a groups of same aged people



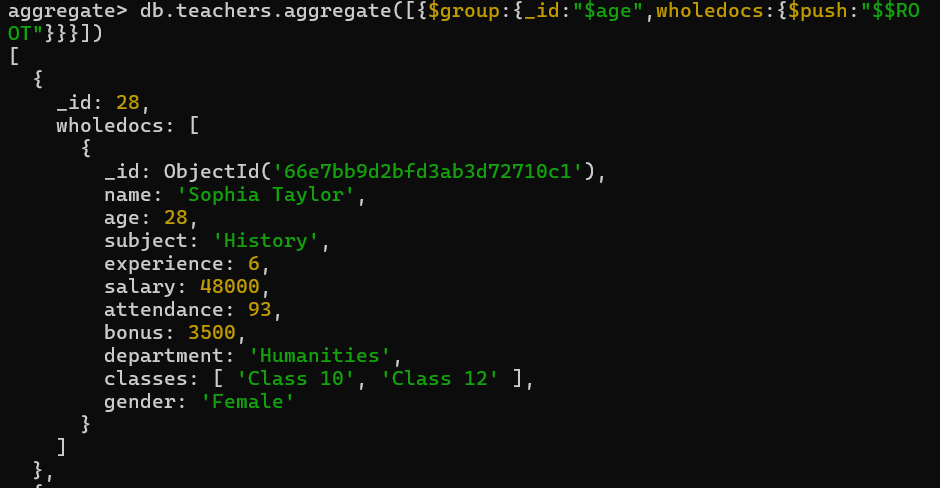
Group in the aggregation

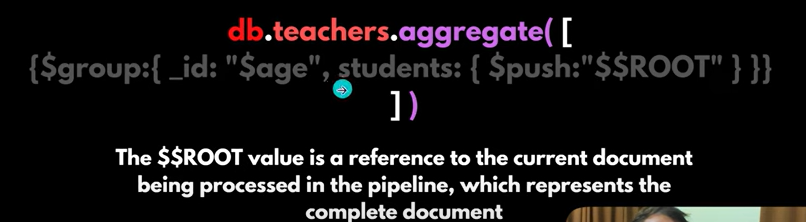


SCANNING WHOLE DOCUMENTS :

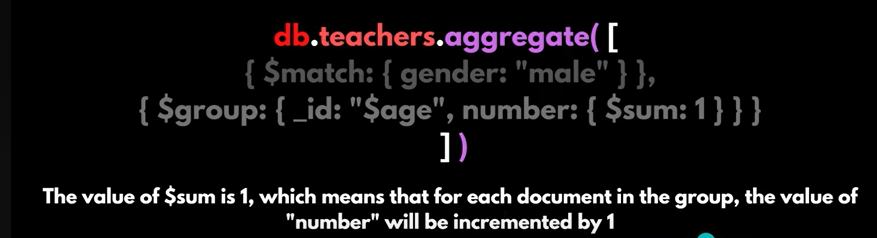
IN MONGO DB WHOLE DOCUMET IS REFFERED AS A **$$ROOT**

FOR EG

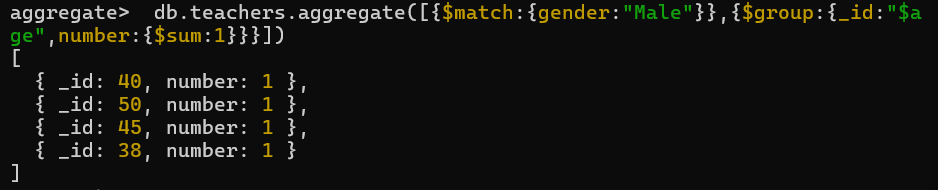
IN THIS QUERY WE AGGGREGATE THE TEACHERS BY ID AND THEN SHOWS ALL THE DATA OF THAT PERSON BY PUSHING ROOT IN THE NEW VARIABLE NAMED WHOLEDOCUMENT 



**$match opration:**

****

In this example in first braces we match the possibilities and in second braces we create a group using specific attribute

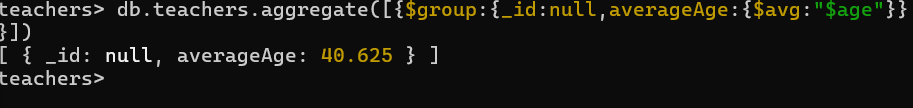
::: first curly braces we sort documents by condition match document by match:{gender:”MALE”},and second curly braces we group them with group command and then use opration **$sum** to find the count of the documents in the whole group

From remaining all

Mongo db remaining

Read me retrive all deleted data later

Aggregate function it return the average age in the teachers group



Unwid in mongodb:

In MongoDB, the $unwind function is used in aggregation pipelines to deconstruct an array field from the input documents and output one document for each element of the array. It essentially "unwinds" the array, creating a new document for each element in the array, while preserving the other fields.

{

$unwind: {

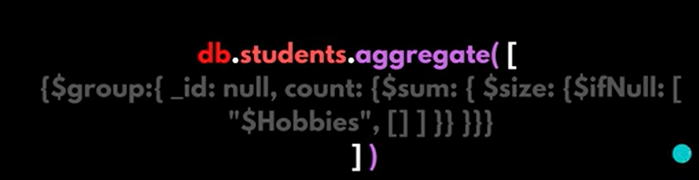
path: "$arrayField",

preserveNullAndEmptyArrays: true // Optional

}

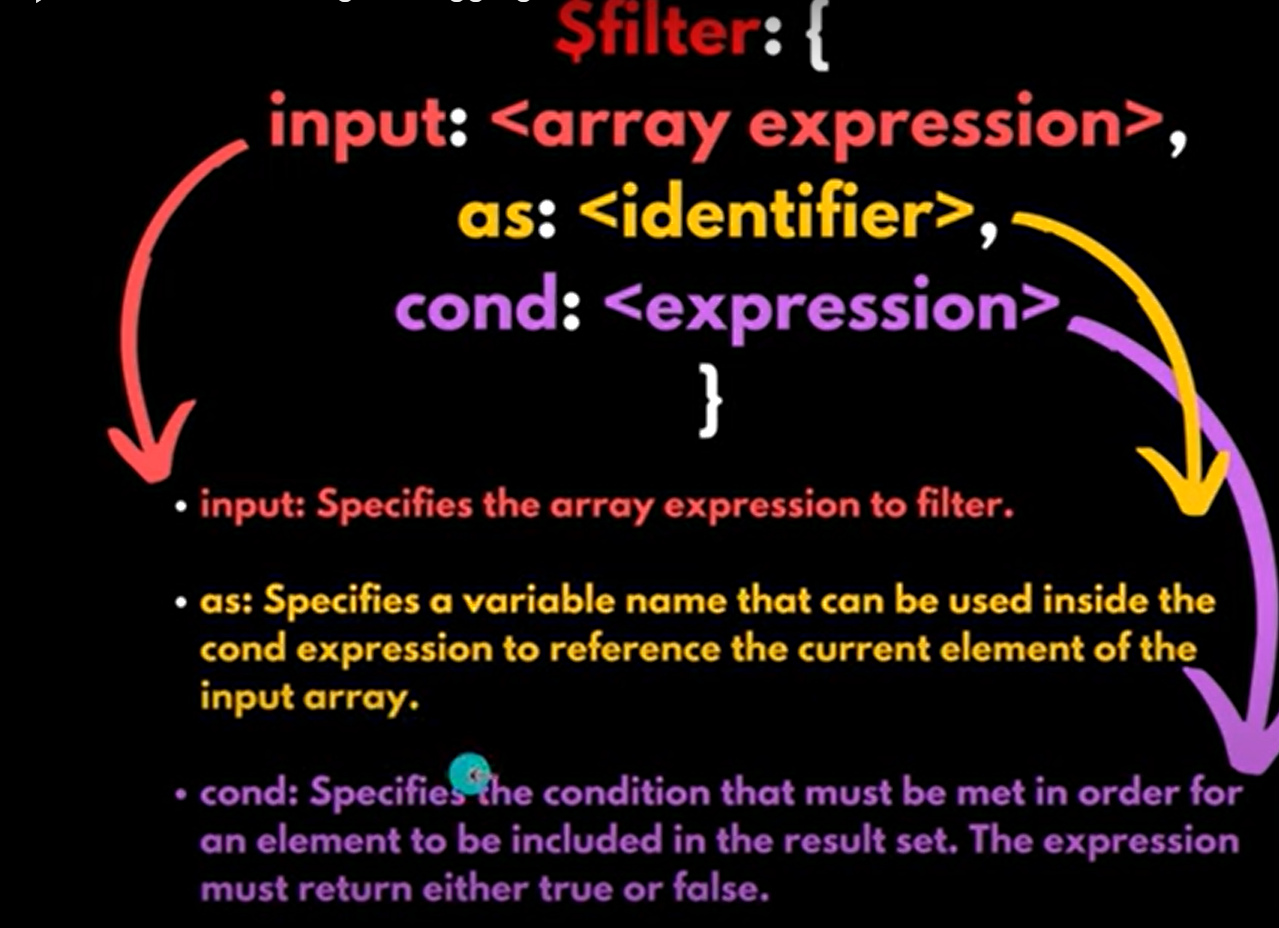
}

Grouping the students by age and then we count the size of hobbies first we count the size of each record in the collections then sum all the records and then display AS A count

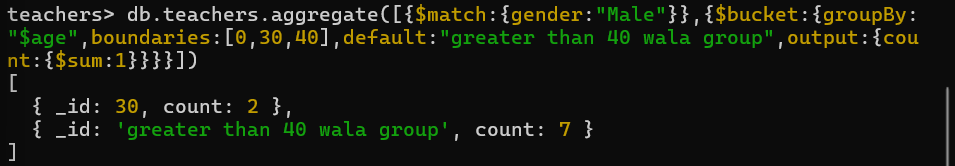




Grouping all the hobbies in the 



In MongoDB, the $bucket operator is used in aggregation pipelines to group documents into buckets (or ranges) based on a specified field's values. It is especially useful when you want to categorize data into ranges (for example, age groups, score ranges, etc.).



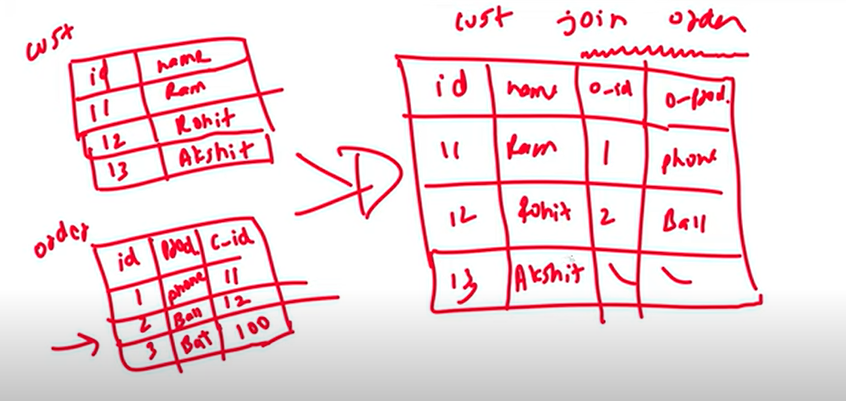


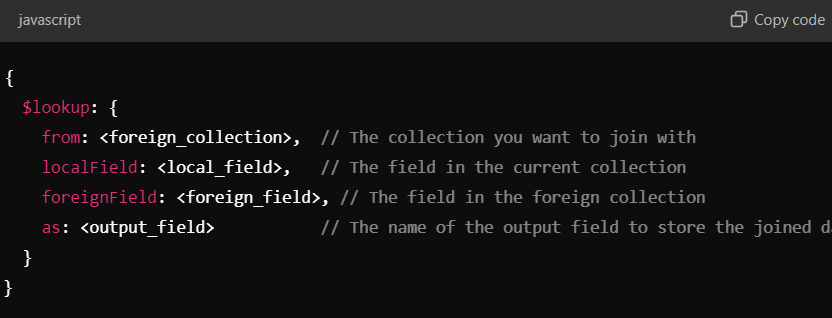
It includes all the names of teachers in a bucketlist and also tell the count of male students in each buckets

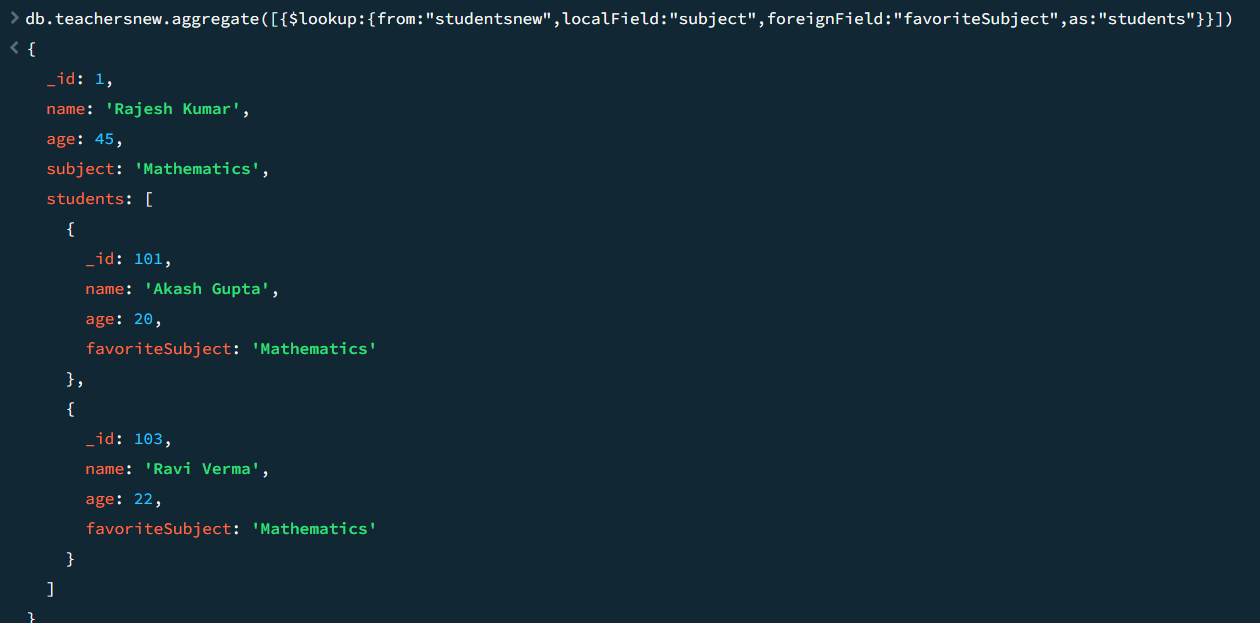
In MongoDB, **joins** between collections are achieved using the $lookup aggregation stage. MongoDB is a NoSQL database and does not inherently support joins like SQL databases do, but the $lookup stage allows you to perform joins between collections.

**$lookup in MongoDB**

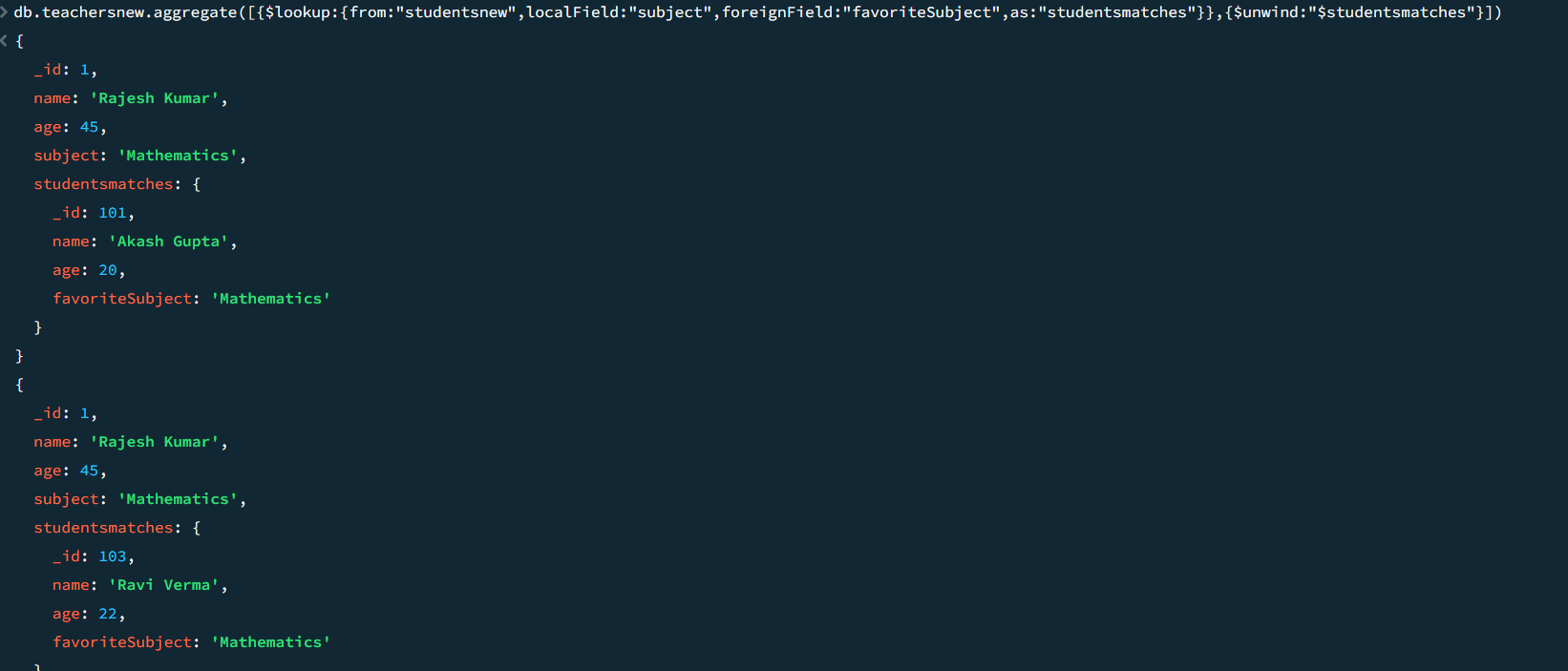
The $lookup stage allows you to join data from one collection to another during an aggregation query. You can combine documents from the local collection with documents from another collection based on a specified field.



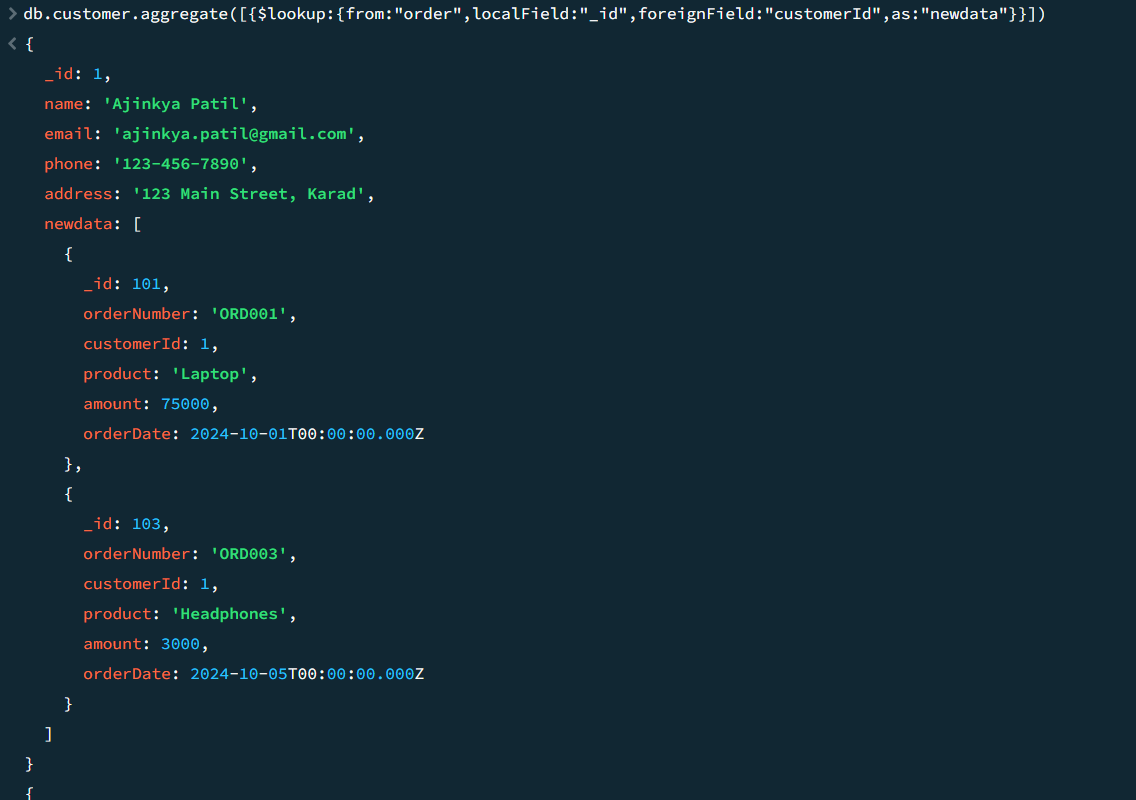




In this case we use lookup and connect collection studentsnew and we use subject field to match the fields and foreignfield is favorite in studentsnew is compared and as students it will display



We use unwind for for creating a diffrent instance of same matchs it removes nested loops



Joining the order with respective customer through respective \_id

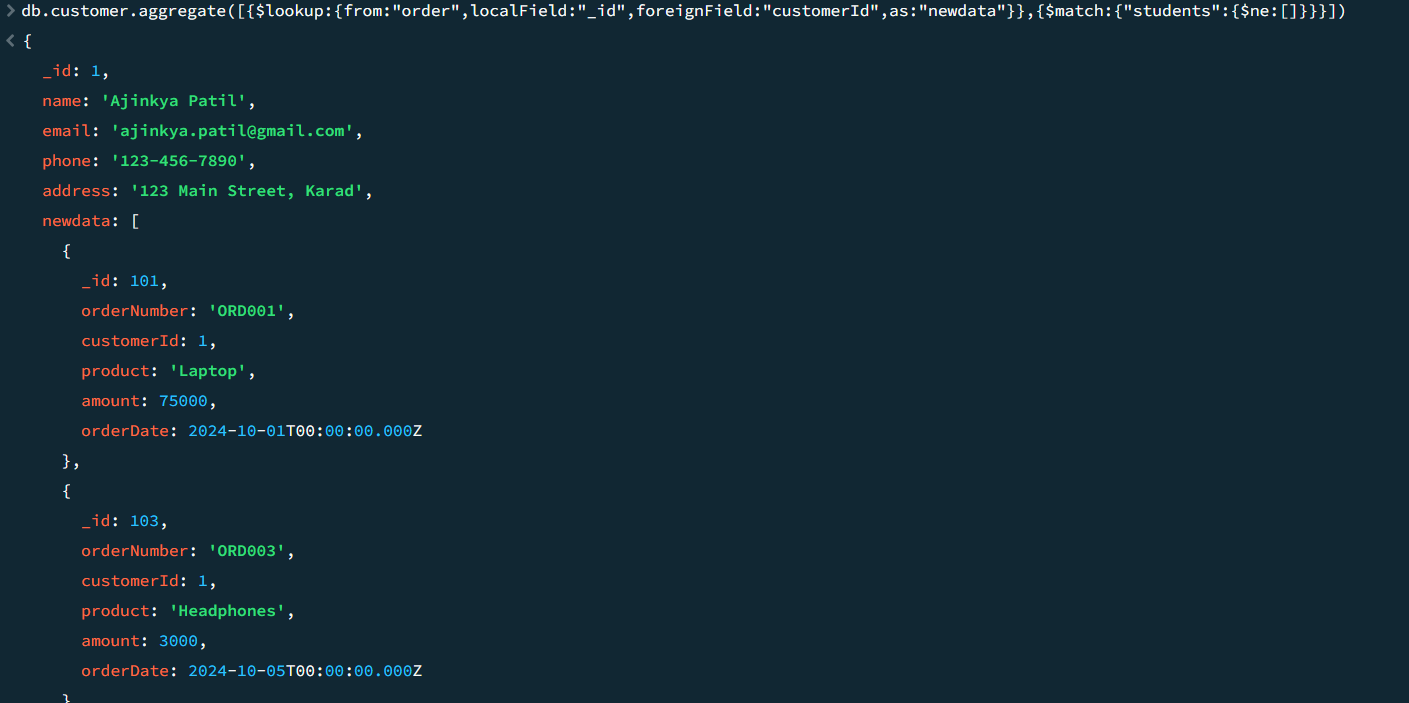
**Inner join in mongo db**

**2. Inner Join (Simulating using $lookup + $match)**

**To simulate an Inner Join, you can use a $match stage after the $lookup to filter out documents where there is no match. This will exclude documents from the local collection if no corresponding documents from the foreign collection exist.**

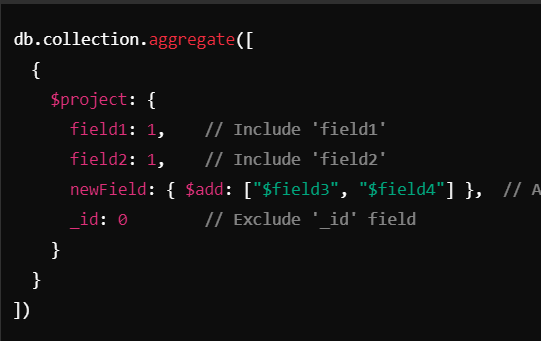
**Inner Join Example:**

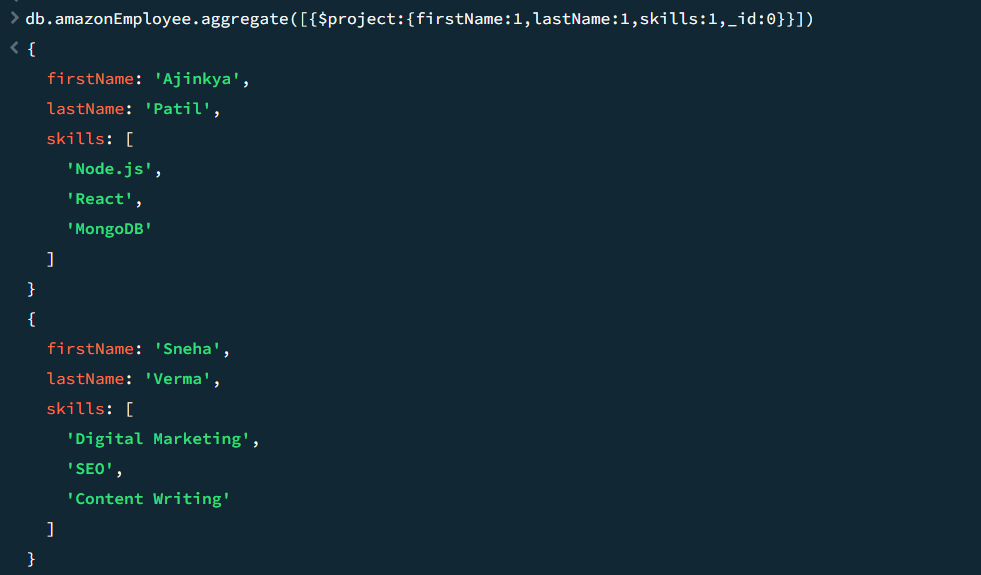
**Continuing with the same collections, we’ll only return teachers who have at least one matching student.**

****

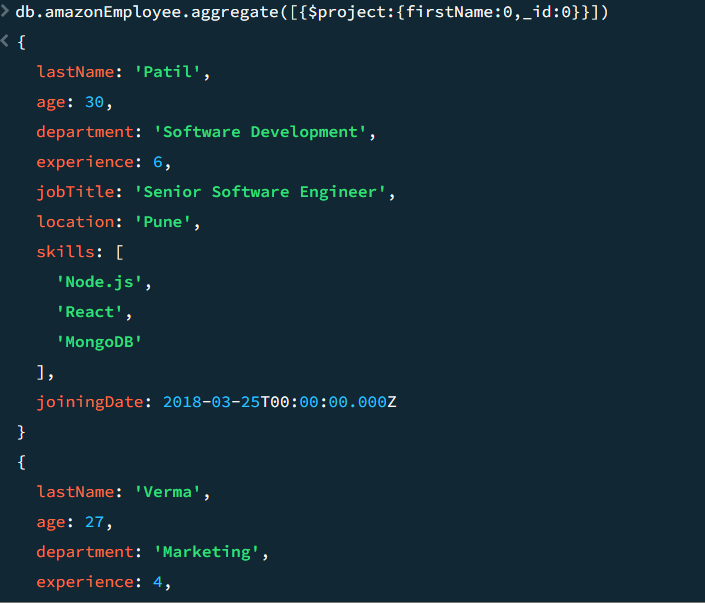
**In MongoDB, the $project stage is used in aggregation pipelines to reshape the documents by including, excluding, or adding computed fields. It is commonly used to control which fields are passed to the next stage of the pipeline or returned as the output. You can use the $project stage to:**

* **Include or exclude specific fields.**
* **Create new fields or modify existing ones.**
* **Perform computations like adding, subtracting, concatenating, etc.**

****



Basically in the project we include only required fields in the output rather than all the fields it only shows field in the documents that required rather than all fields use {1} for add the field in output default it is ‘’0 for all the field rather than {\_id}



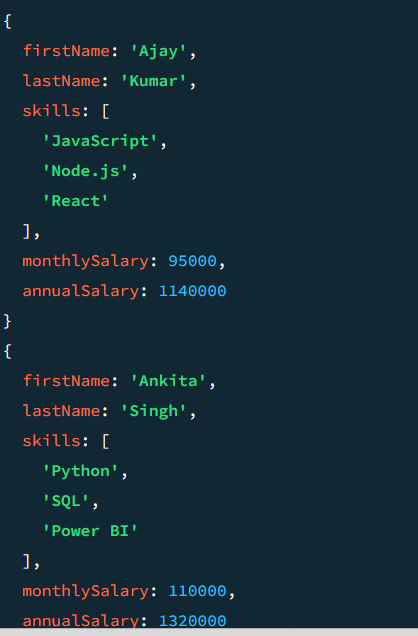
If we put xyz:{0} then it will remove that field only and include all the remaining field in the output



We can rename the field also in the project by providing [new\_name:”$old\_name ”]and it will change the name

But it will not change it MAIN RECORD IT ONLY OCCURE IN CURRENT OUTPUT





In this case we find a yearly salary by providing appropriate expression like [annualsalary :{$multiply:[12,”$salary”]}]

**Capped collections in mongodb:**

In MongoDB, a **capped collection** is a type of collection that maintains a fixed size. This means that when the collection reaches its size limit, it automatically overwrites the oldest documents with new ones in a circular, FIFO (First In, First Out) manner. Capped collections are optimized for high-throughput operations and are ideal for scenarios where you want to store recent, frequently updated data, such as logs or sensor data.

**Syntax:** **// Create a capped collection with a maximum size of 100 KB and a maximum of 3 documents**

**db.createCollection("logs", { capped: true, size: 100000, max: 3 })**

****

****

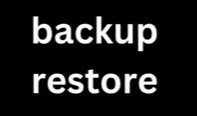
**Authentication in the mongo db :** **Authentication in MongoDB is the process of verifying the identity of a user or client attempting to connect to a MongoDB instance. MongoDB provides multiple mechanisms to enforce authentication, ensuring that only authorized users can access databases or perform operations.**

**User authorities in mongodb and roles in mongodb**

****

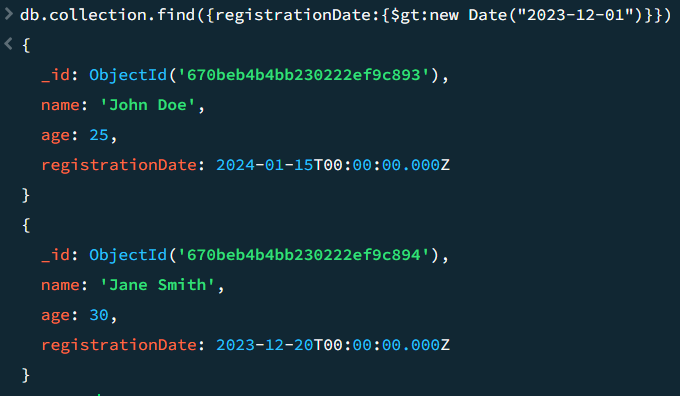
****

****

****

**MongoDB Replication & Sharding: Scaling Your Database for Performance**

****



Done done ….