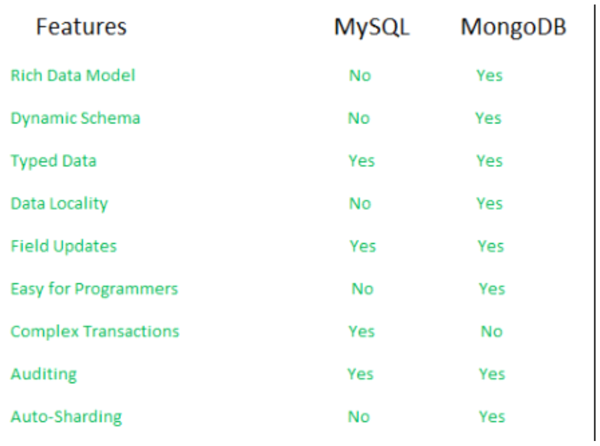
# MonogoDB

Monday, January 6, 2025 9:23 PM

What is MongoDB?

MongoDB is a cross-platform document-based database. Categorized as a NoSQL database, MongoDB avoids the conventional table-oriented relational database structure in support of the JSON-like documents with the dynamic schemas, making the data integration in specific kinds of applications quicker and simpler.



| Features | MySQL | MongoDB |
|---|---|---|
| Rich Data Model | No | Yes |
| Dynamic Schema | No | Yes |
| Typed Data | Yes | Yes |
| Data Locality | No | Yes |
| Field Updates | Yes | Yes |
| Easy for Programmers | No | Yes |
| Complex Transactions | Yes | No |
| Auditing | Yes | Yes |
| Auto-Sharding | No | Yes |

Document :

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

- Less relation
- Data stored together

Why MongoDB is the best NoSQL database?

MongoDB is the best NoSQL database due to the following features:

- High Performance
- High Availability
- Easily Scalable
- Rich Query Language
- Document Oriented

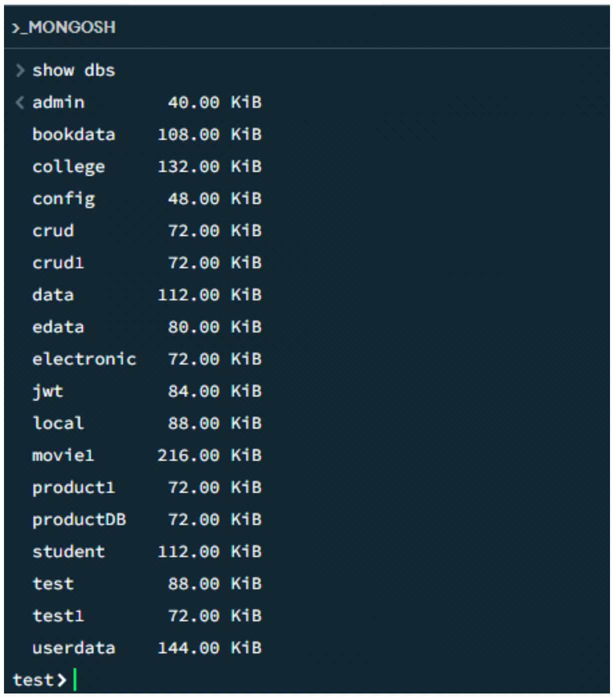
What is Document?
A record in MongoDB is a document, which is a data structure composed of field and value pairs.

What is collection?
The collection is a set of documents.

What is MongoDB Atlas?
MongoDB Atlas is a fully-managed cloud database that handles all the complexity of deploying, managing, and healing your deployments on the cloud service provider of your choice (AWS , Azure, and GCP).

How can I see the list of databases I created before?

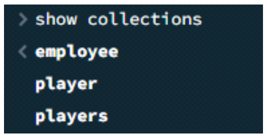
- show dbs



```
>_MONGOSH
> show dbs
< admin         40.00 KiB
  bookdata     108.00 KiB
  college      132.00 KiB
  config        48.00 KiB
  crud          72.00 KiB
  crud1         72.00 KiB
  data         112.00 KiB
  edata         80.00 KiB
  electronic    72.00 KiB
  jwt           84.00 KiB
  local         88.00 KiB
  movie1       216.00 KiB
  product1      72.00 KiB
  productDB     72.00 KiB
  student      112.00 KiB
  test          88.00 KiB
  test1         72.00 KiB
  userdata     144.00 KiB
test>
```

How can I see the list of collections I created before?

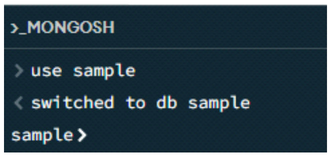
- show collections



```
> show collections
< employee
  player
  players
```

How to create database in MongoDB?

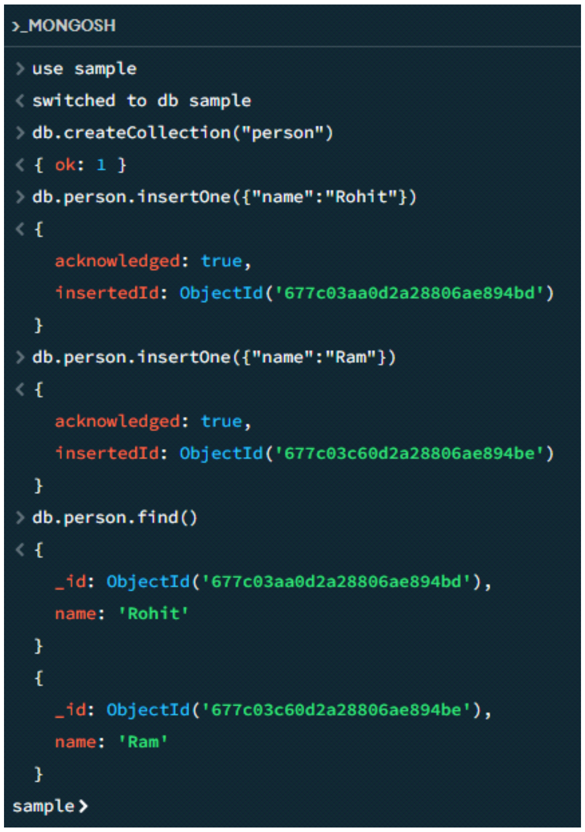
- use db_name



```
>_MONGOSH
> use sample
< switched to db sample
sample>
```

How to create collection?

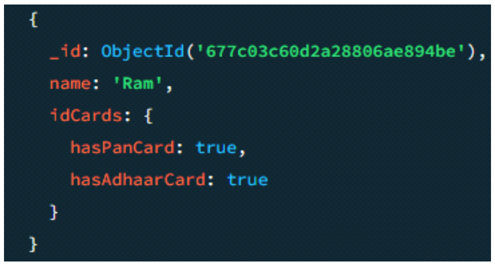
- db.createCollection(<collection_name>,<option>)
- More Details :- https://www.mongodb.com/docs/manual/reference/method/db.createCollection/
- db.collection_name.insertOne({"field":"value"})



```
>_MONGOSH
> use sample
< switched to db sample
> db.createCollection("person")
< { ok: 1 }
> db.person.insertOne({"name":"Rohit"})
< {
    acknowledged: true,
    insertedId: ObjectId('677c03aa0d2a28806ae894bd')
  }
> db.person.insertOne({"name":"Ram"})
< {
    acknowledged: true,
    insertedId: ObjectId('677c03c60d2a28806ae894be')
  }
> db.person.find()
< {
    _id: ObjectId('677c03aa0d2a28806ae894bd'),
    name: 'Rohit'
  }
  {
    _id: ObjectId('677c03c60d2a28806ae894be'),
    name: 'Ram'
  }
sample>
```

What is embedded document?

- Nested document



```
  {
    _id: ObjectId('677c03c60d2a28806ae894be'),
    name: 'Ram',
    idCards: {
      hasPanCard: true,
      hasAdhaarCard: true
    }
  }
```

CRUD :- Create,Read,Update,Delete

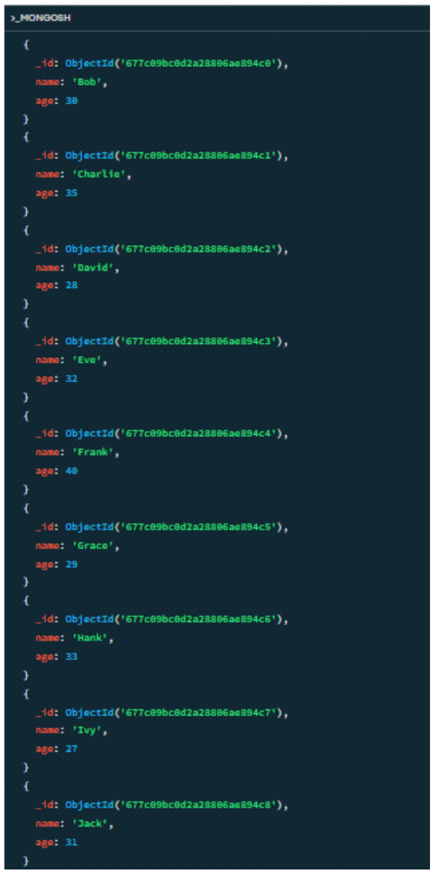
1. Create :
- insertOne(data,option)
- insertMany(data,option)

2. Read :
- find(filter,options)
- findOne(filter,options)

3. Delete :
- deleteOne(filter,option)
- deleteMany(filter,option)

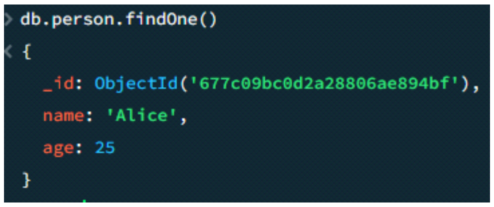
4. Update :
- updateOne(filter,data,option)
- updateMany(filter,data,option)
- replaceOne(filter,data,option)

☐ Find()



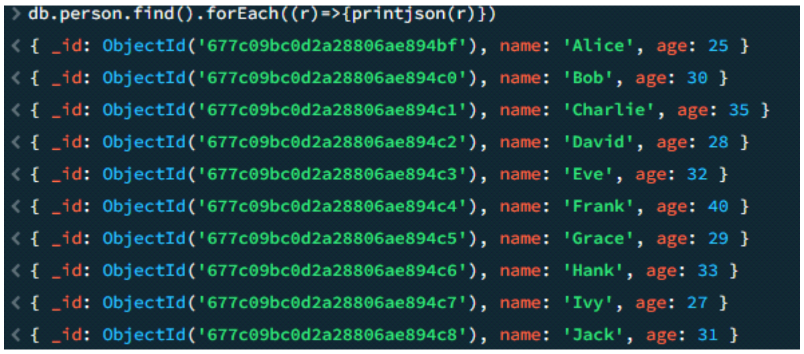
```
>_MONGOSH
  {
    _id: ObjectId('677c09bc0d2a28806ae894c0'),
    name: 'Bob',
    age: 30
  }
  {
    _id: ObjectId('677c09bc0d2a28806ae894c1'),
    name: 'Charlie',
    age: 35
  }
  {
    _id: ObjectId('677c09bc0d2a28806ae894c2'),
    name: 'David',
    age: 28
  }
  {
    _id: ObjectId('677c09bc0d2a28806ae894c3'),
    name: 'Eve',
    age: 32
  }
  {
    _id: ObjectId('677c09bc0d2a28806ae894c4'),
    name: 'Frank',
    age: 40
  }
  {
    _id: ObjectId('677c09bc0d2a28806ae894c5'),
    name: 'Grace',
    age: 29
  }
  {
    _id: ObjectId('677c09bc0d2a28806ae894c6'),
    name: 'Hank',
    age: 33
  }
  {
    _id: ObjectId('677c09bc0d2a28806ae894c7'),
    name: 'Ivy',
    age: 27
  }
  {
    _id: ObjectId('677c09bc0d2a28806ae894c8'),
    name: 'Jack',
    age: 31
  }
```

- findOne()



```
> db.person.findOne()
< {
    _id: ObjectId('677c09bc0d2a28806ae894bf'),
    name: 'Alice',
    age: 25
  }
```

Foreach :



```
> db.person.find().forEach((r)=>{printjson(r)})
< { _id: ObjectId('677c09bc0d2a28806ae894bf'), name: 'Alice', age: 25 }
< { _id: ObjectId('677c09bc0d2a28806ae894c0'), name: 'Bob', age: 30 }
< { _id: ObjectId('677c09bc0d2a28806ae894c1'), name: 'Charlie', age: 35 }
< { _id: ObjectId('677c09bc0d2a28806ae894c2'), name: 'David', age: 28 }
< { _id: ObjectId('677c09bc0d2a28806ae894c3'), name: 'Eve', age: 32 }
< { _id: ObjectId('677c09bc0d2a28806ae894c4'), name: 'Frank', age: 40 }
< { _id: ObjectId('677c09bc0d2a28806ae894c5'), name: 'Grace', age: 29 }
< { _id: ObjectId('677c09bc0d2a28806ae894c6'), name: 'Hank', age: 33 }
< { _id: ObjectId('677c09bc0d2a28806ae894c7'), name: 'Ivy', age: 27 }
< { _id: ObjectId('677c09bc0d2a28806ae894c8'), name: 'Jack', age: 31 }
```

Find no of document :
db.collection_name.find().count()

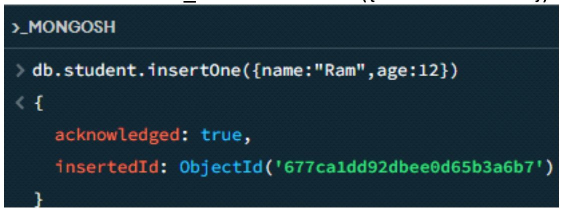
Find limited record
db.collection_name.find().limit()

Convert in array:

db.collection_name.find().toArray()

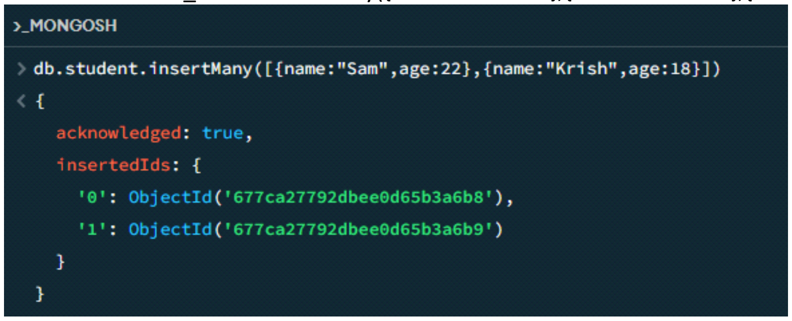
- Insert :

1. db.collection_name.insertOne({"field1":"value"})



```
>_MONGOSH
> db.student.insertOne({name:"Ram",age:12})
< {
    acknowledged: true,
    insertedId: ObjectId('677ca1dd92dbee0d65b3a6b7')
  }
```

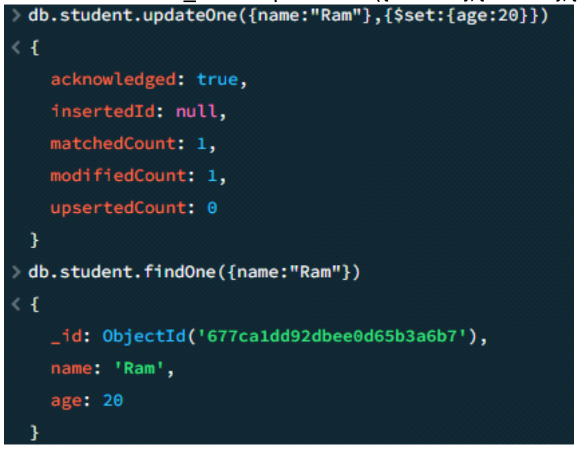
2. db.collection_name.insertMany({"field1":"value"},{"field1":"value"},{"field1":"value"})



```
>_MONGOSH
> db.student.insertMany([{name:"Sam",age:22},{name:"Krish",age:18}])
< {
    acknowledged: true,
    insertedIds: {
      '0': ObjectId('677ca27792dbee0d65b3a6b8'),
      '1': ObjectId('677ca27792dbee0d65b3a6b9')
    }
  }
```

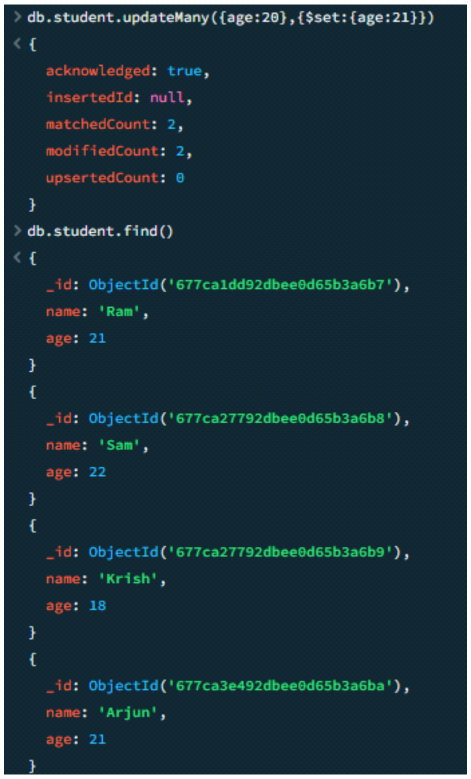
- Update:

1. db.collection_name.updateOne({<filter>},{<data>},{<option>})



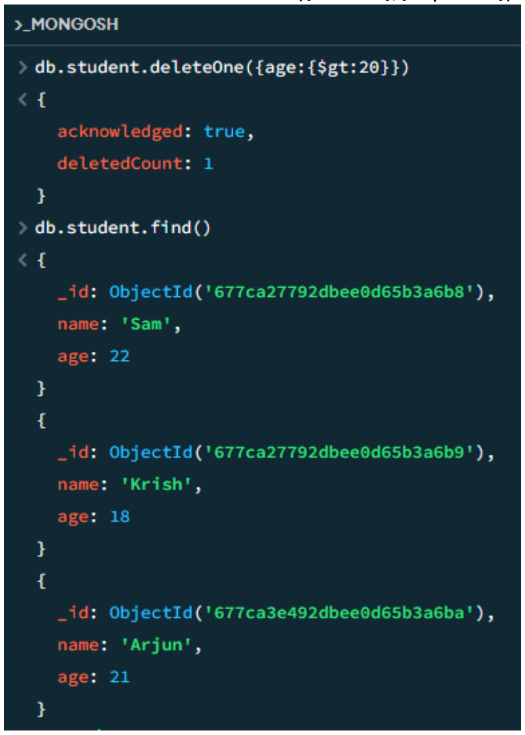
```
> db.student.updateOne({name:"Ram"},{$set:{age:20}})
< {
    acknowledged: true,
    insertedId: null,
    matchedCount: 1,
    modifiedCount: 1,
    upsertedCount: 0
  }
> db.student.findOne({name:"Ram"})
< {
    _id: ObjectId('677ca1dd92dbee0d65b3a6b7'),
    name: 'Ram',
    age: 20
  }
```

1. db.collection_name.updateMany({<filter>},{<data>},{<option>})



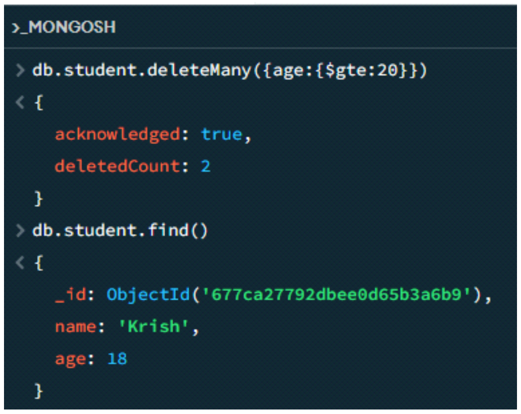
```
> db.student.updateMany({age:20},{$set:{age:21}})
< {
    acknowledged: true,
    insertedId: null,
    matchedCount: 2,
    modifiedCount: 2,
    upsertedCount: 0
  }
> db.student.find()
< {
    _id: ObjectId('677ca1dd92dbee0d65b3a6b7'),
    name: 'Ram',
    age: 21
  }
  {
    _id: ObjectId('677ca27792dbee0d65b3a6b8'),
    name: 'Sam',
    age: 22
  }
  {
    _id: ObjectId('677ca27792dbee0d65b3a6b9'),
    name: 'Krish',
    age: 18
  }
  {
    _id: ObjectId('677ca3e492dbee0d65b3a6ba'),
    name: 'Arjun',
    age: 21
  }
```

4. Delete :
   - db.collection.deleteOne({<filter>},{<option>})



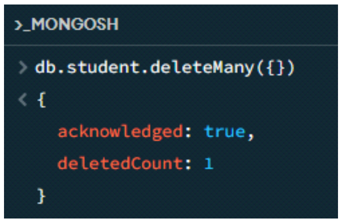
```
>_MONGOSH
> db.student.deleteOne({age:{$gt:20}})
< {
    acknowledged: true,
    deletedCount: 1
  }
> db.student.find()
< {
    _id: ObjectId('677ca27792dbee0d65b3a6b8'),
    name: 'Sam',
    age: 22
  }
  {
    _id: ObjectId('677ca27792dbee0d65b3a6b9'),
    name: 'Krish',
    age: 18
  }
  {
    _id: ObjectId('677ca3e492dbee0d65b3a6ba'),
    name: 'Arjun',
    age: 21
  }
```

5. db.collection_name.deleteMany({<filter>},{<option>})



```
>_MONGOSH
> db.student.deleteMany({age:{$gte:20}})
< {
    acknowledged: true,
    deletedCount: 2
  }
> db.student.find()
< {
    _id: ObjectId('677ca27792dbee0d65b3a6b9'),
    name: 'Krish',
    age: 18
  }
```

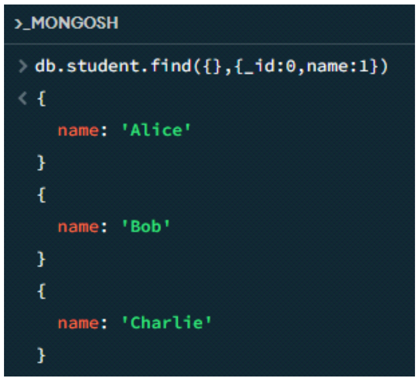
Delete all document :



```
>_MONGOSH
> db.student.deleteMany({})
< {
    acknowledged: true,
    deletedCount: 1
  }
```

What is projection ?

Projection nothing but selection of field.



```
>_MONGOSH
> db.student.find({},{_id:0,name:1})
< {
    name: 'Alice'
  }
  {
    name: 'Bob'
  }
  {
    name: 'Charlie'
  }
```

MongoDb is schemaless?

Yes

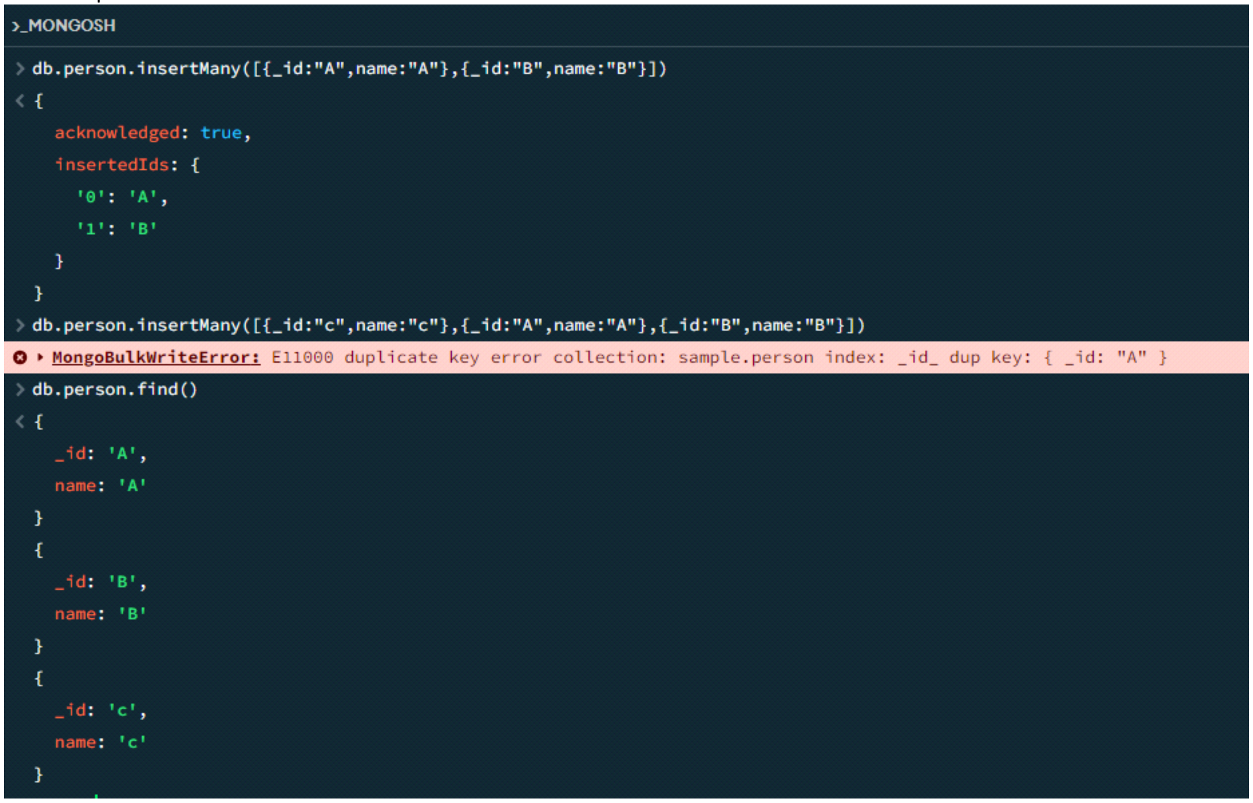
DataTypes In MongoDB:

- Text
- Boolean
- Number:
  1. Integer
  2. NumberLong
  3. Floating number
- ObjectID
- ISODate
- Timestamp
- Array
- Embeeded document

Drop database and collection :

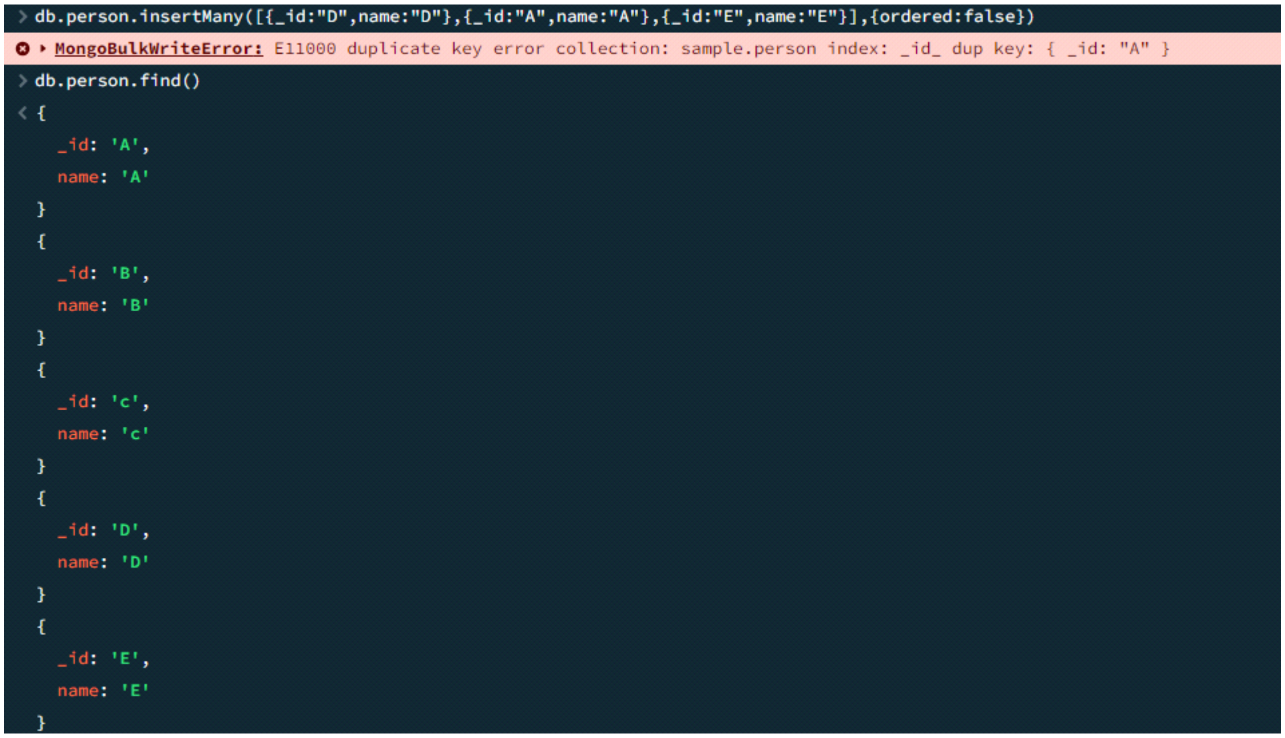
1. db.dropDatabase()
2. db.collection_name.drop()

- Orderd option in insert



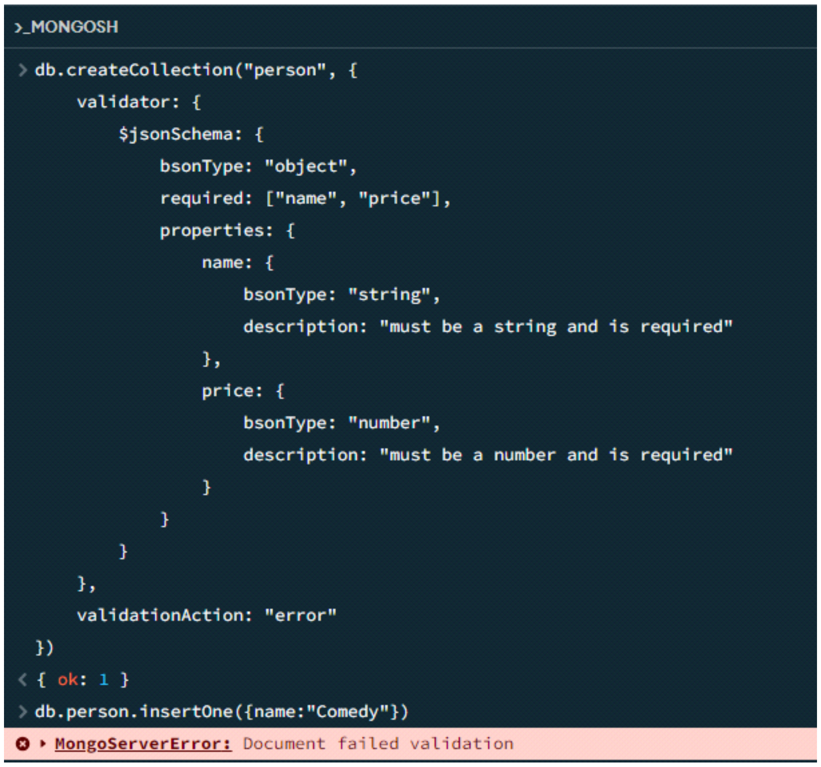
```
>_MONGOSH
> db.person.insertMany([{_id:"A",name:"A"},{_id:"B",name:"B"}])
< {
    acknowledged: true,
    insertedIds: {
      '0': 'A',
      '1': 'B'
    }
  }
> db.person.insertMany([{_id:"c",name:"c"},{_id:"A",name:"A"},{_id:"B",name:"B"}])
✖ ▸ MongoBulkWriteError: E11000 duplicate key error collection: sample.person index: _id_ dup key: { _id: "A" }
> db.person.find()
< {
    _id: 'A',
    name: 'A'
  }
  {
    _id: 'B',
    name: 'B'
  }
  {
    _id: 'c',
    name: 'c'
  }
```

-



```
> db.person.insertMany([{_id:"D",name:"D"},{_id:"A",name:"A"},{_id:"E",name:"E"}],{ordered:false})
✖ ▸ MongoBulkWriteError: E11000 duplicate key error collection: sample.person index: _id_ dup key: { _id: "A" }
> db.person.find()
< {
    _id: 'A',
    name: 'A'
  }
  {
    _id: 'B',
    name: 'B'
  }
  {
    _id: 'c',
    name: 'c'
  }
  {
    _id: 'D',
    name: 'D'
  }
  {
    _id: 'E',
    name: 'E'
  }
```

Validation :



```
>_MONGOSH
> db.createCollection("person", {
      validator: {
          $jsonSchema: {
              bsonType: "object",
              required: ["name", "price"],
              properties: {
                  name: {
                      bsonType: "string",
                      description: "must be a string and is required"
                  },
                  price: {
                      bsonType: "number",
                      description: "must be a number and is required"
                  }
              }
          }
      },
      validationAction: "error"
  })
< { ok: 1 }
> db.person.insertOne({name:"Comedy"})
⊗ ▸ MongoServerError: Document failed validation
```

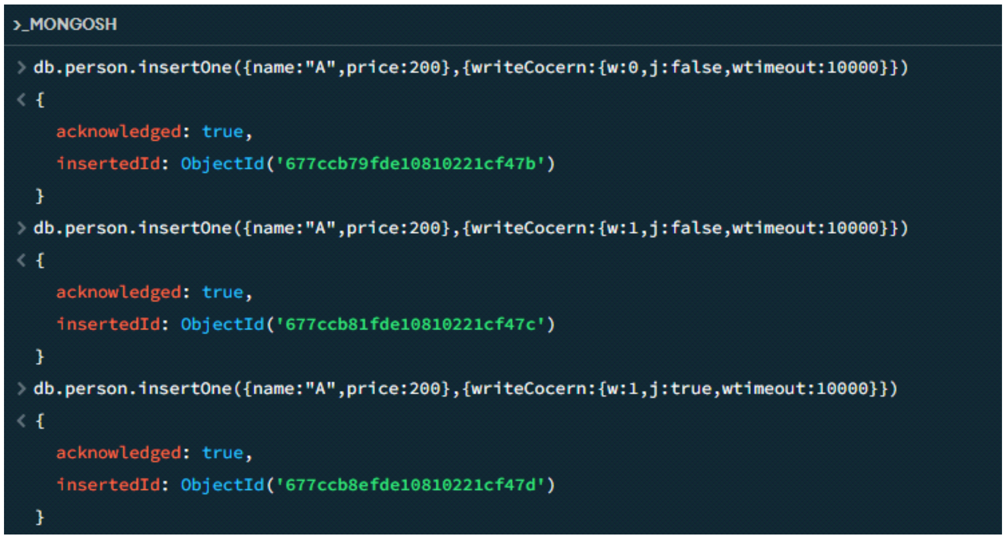
WriteCoern in mongodb for insert query:

Write concern describes the level of acknowledgment requested from MongoDB for write operations to a standalone mongod, replica sets, or sharded clusters.

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

- the w option to request acknowledgment that the write operation has propagated to a specified number of mongod instances or to mongod instances with specified tags.
- the j option to request acknowledgment that the write operation has been written to the on-disk journal, and
- the wtimeout option to specify a time limit to prevent write operations from blocking indefinitely.

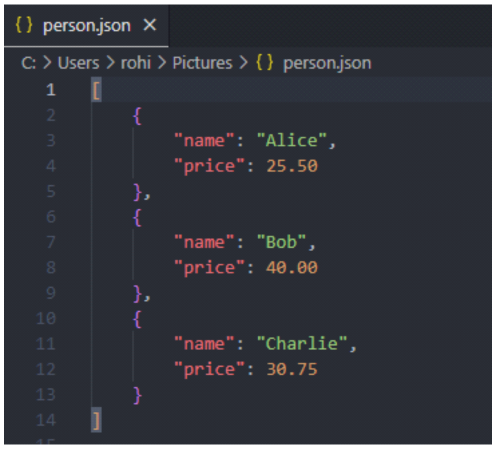
From <https://www.mongodb.com/docs/manual/reference/write-concern/>



```
>_MONGOSH
> db.person.insertOne({name:"A",price:200},{writeCocern:{w:0,j:false,wtimeout:10000}})
< {
    acknowledged: true,
    insertedId: ObjectId('677ccb79fde10810221cf47b')
  }
> db.person.insertOne({name:"A",price:200},{writeCocern:{w:1,j:false,wtimeout:10000}})
< {
    acknowledged: true,
    insertedId: ObjectId('677ccb81fde10810221cf47c')
  }
> db.person.insertOne({name:"A",price:200},{writeCocern:{w:1,j:true,wtimeout:10000}})
< {
    acknowledged: true,
    insertedId: ObjectId('677ccb8efde10810221cf47d')
  }
```

Import json in mongodb

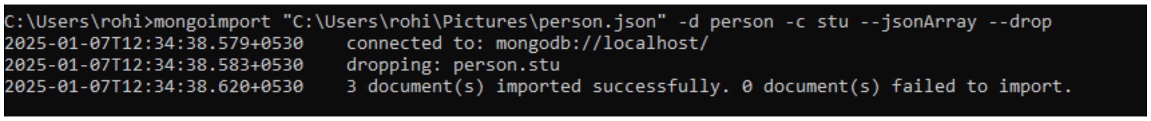
Mongoimport "path of json file" -d db_name -c collection_name --jsonArray --drop



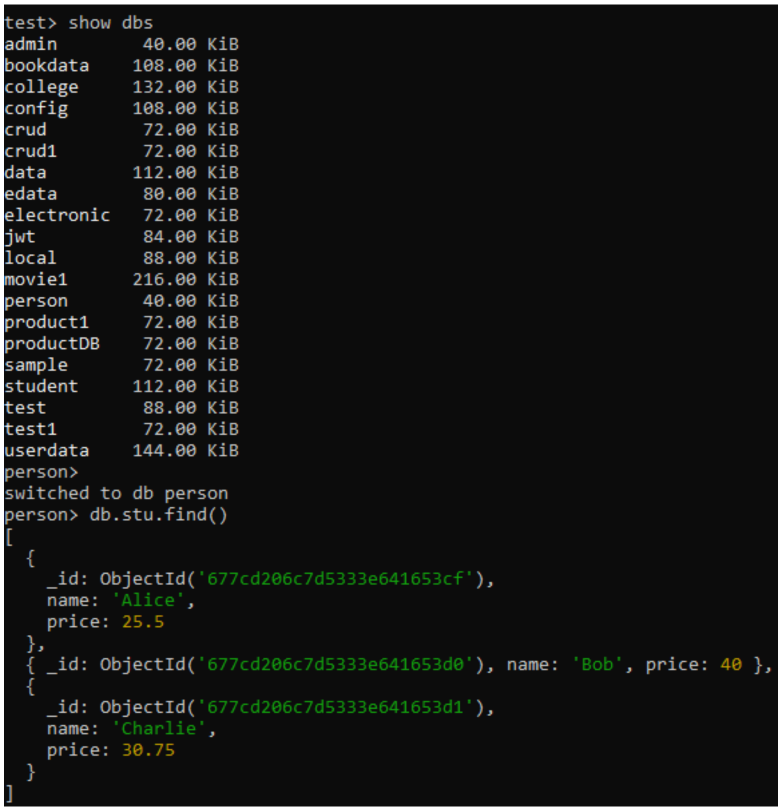
person.json

C: > Users > rohi > Pictures > person.json

```
[
    {
        "name": "Alice",
        "price": 25.50
    },
    {
        "name": "Bob",
        "price": 40.00
    },
    {
        "name": "Charlie",
        "price": 30.75
    }
]
```



```
C:\Users\rohi>mongoimport "C:\Users\rohi\Pictures\person.json" -d person -c stu --jsonArray --drop
2025-01-07T12:34:38.579+0530    connected to: mongodb://localhost/
2025-01-07T12:34:38.583+0530    dropping: person.stu
2025-01-07T12:34:38.620+0530    3 document(s) imported successfully. 0 document(s) failed to import.
```



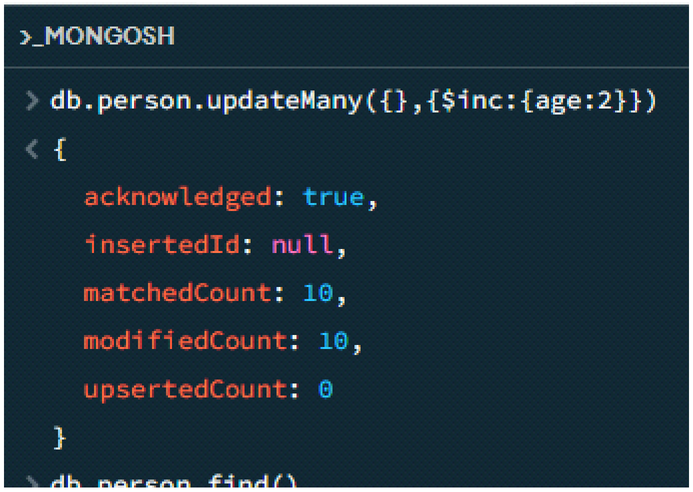
```
test> show dbs
admin        40.00 KiB
bookdata    108.00 KiB
college     132.00 KiB
config      108.00 KiB
crud         72.00 KiB
crud1        72.00 KiB
data        112.00 KiB
edata        80.00 KiB
electronic   72.00 KiB
jwt          84.00 KiB
local        88.00 KiB
movie1      216.00 KiB
person       40.00 KiB
product1     72.00 KiB
productDB    72.00 KiB
sample       72.00 KiB
student     112.00 KiB
test         88.00 KiB
test1        72.00 KiB
userdata    144.00 KiB
person>
switched to db person
person> db.stu.find()
[
  {
    _id: ObjectId('677cd206c7d5333e641653cf'),
    name: 'Alice',
    price: 25.5
  },
  { _id: ObjectId('677cd206c7d5333e641653d0'), name: 'Bob', price: 40 },
  {
    _id: ObjectId('677cd206c7d5333e641653d1'),
    name: 'Charlie',
    price: 30.75
  }
]
```

# Operator

Wednesday, January 8, 2025 8:06 AM

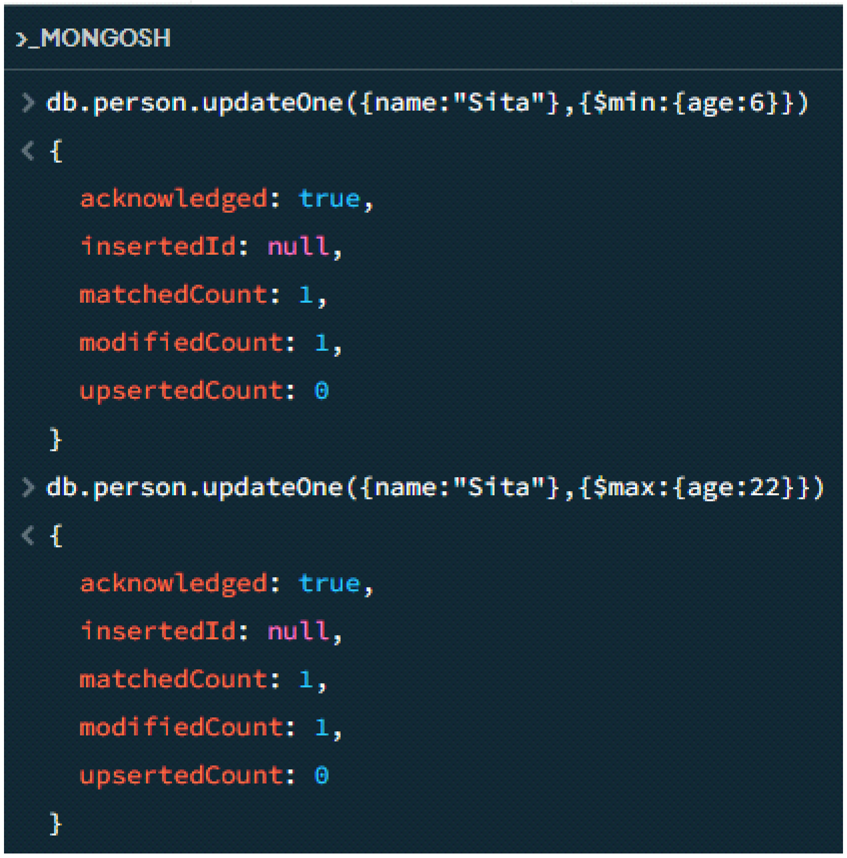
## $inc, $min, $max, $mul, $unset, $rename & Upsert

### $inc :



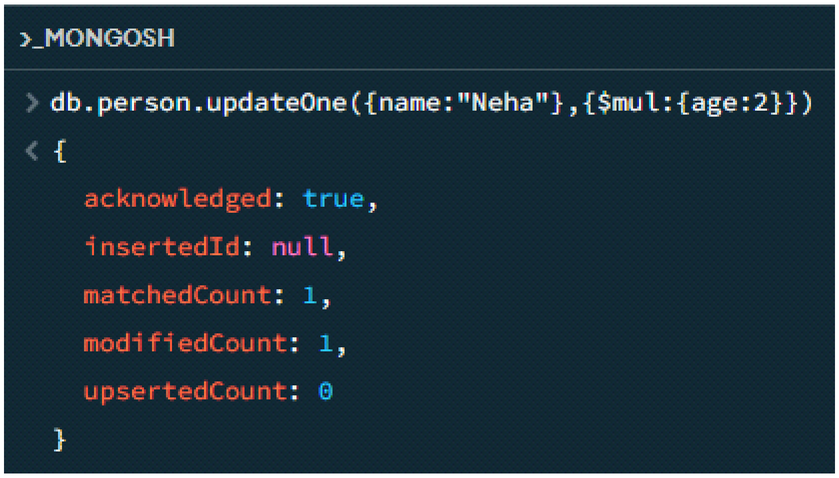
```
>_MONGOSH
> db.person.updateMany({},{$inc:{age:2}})
< {
    acknowledged: true,
    insertedId: null,
    matchedCount: 10,
    modifiedCount: 10,
    upsertedCount: 0
  }
> db.person.find()
```

### $min,$max :



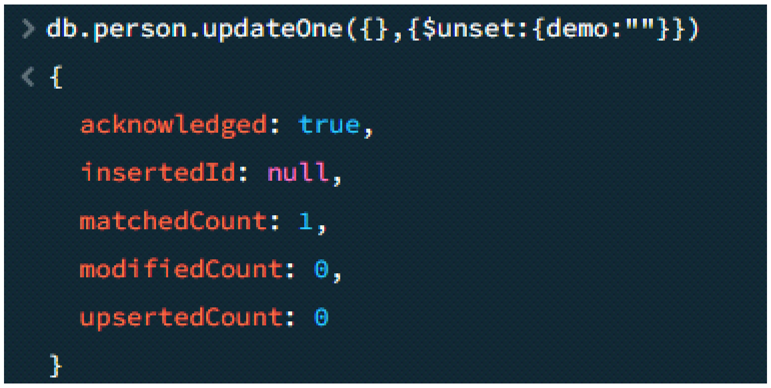
```
>_MONGOSH
> db.person.updateOne({name:"Sita"},{$min:{age:6}})
< {
    acknowledged: true,
    insertedId: null,
    matchedCount: 1,
    modifiedCount: 1,
    upsertedCount: 0
  }
> db.person.updateOne({name:"Sita"},{$max:{age:22}})
< {
    acknowledged: true,
    insertedId: null,
    matchedCount: 1,
    modifiedCount: 1,
    upsertedCount: 0
  }
```

### $mul:



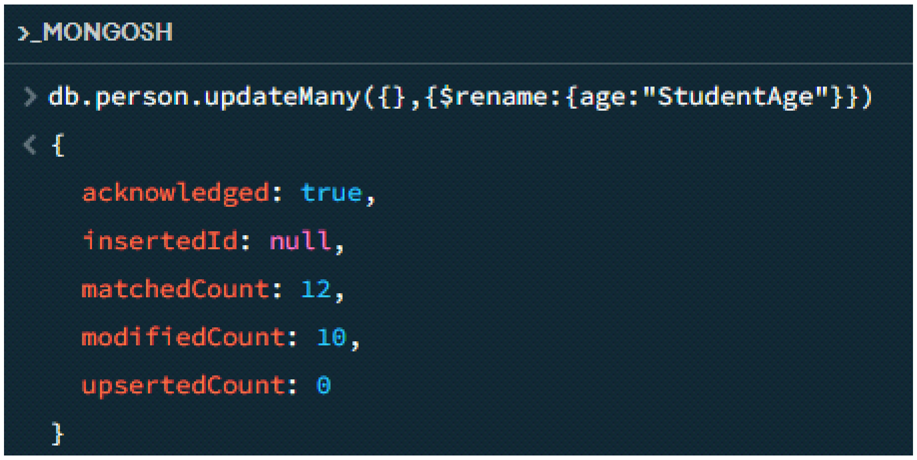
```
>_MONGOSH
> db.person.updateOne({name:"Neha"},{$mul:{age:2}})
< {
    acknowledged: true,
    insertedId: null,
    matchedCount: 1,
    modifiedCount: 1,
    upsertedCount: 0
  }
```

**$unset: used to remove field**



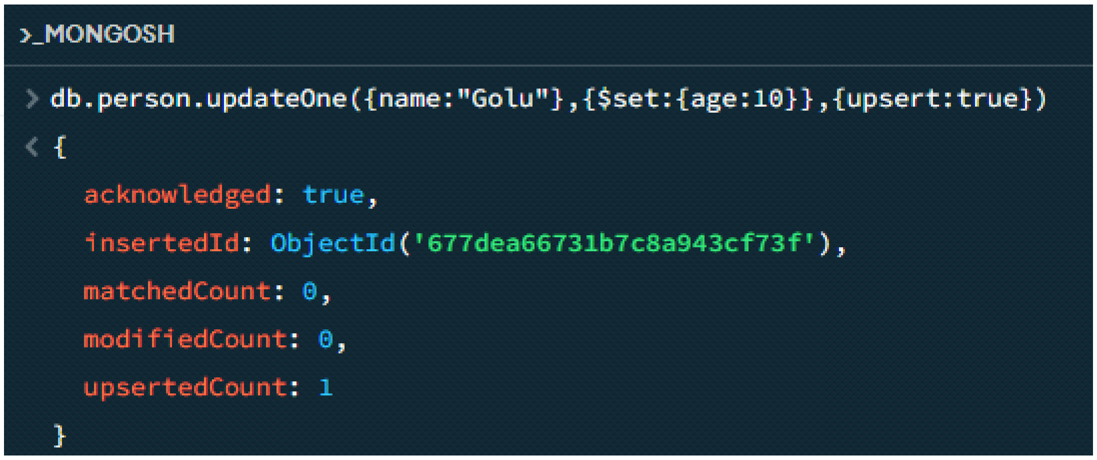
```
> db.person.updateOne({},{$unset:{demo:""}})
< {
    acknowledged: true,
    insertedId: null,
    matchedCount: 1,
    modifiedCount: 0,
    upsertedCount: 0
  }
```

**$rename**



```
>_MONGOSH
> db.person.updateMany({},{$rename:{age:"StudentAge"}})
< {
    acknowledged: true,
    insertedId: null,
    matchedCount: 12,
    modifiedCount: 10,
    upsertedCount: 0
  }
```

**upsert: used for the search and update fields. If not available thar record then insert**



```
>_MONGOSH
> db.person.updateOne({name:"Golu"},{$set:{age:10}},{upsert:true})
< {
    acknowledged: true,
    insertedId: ObjectId('677dea66731b7c8a943cf73f'),
    matchedCount: 0,
    modifiedCount: 0,
    upsertedCount: 1
  }
```

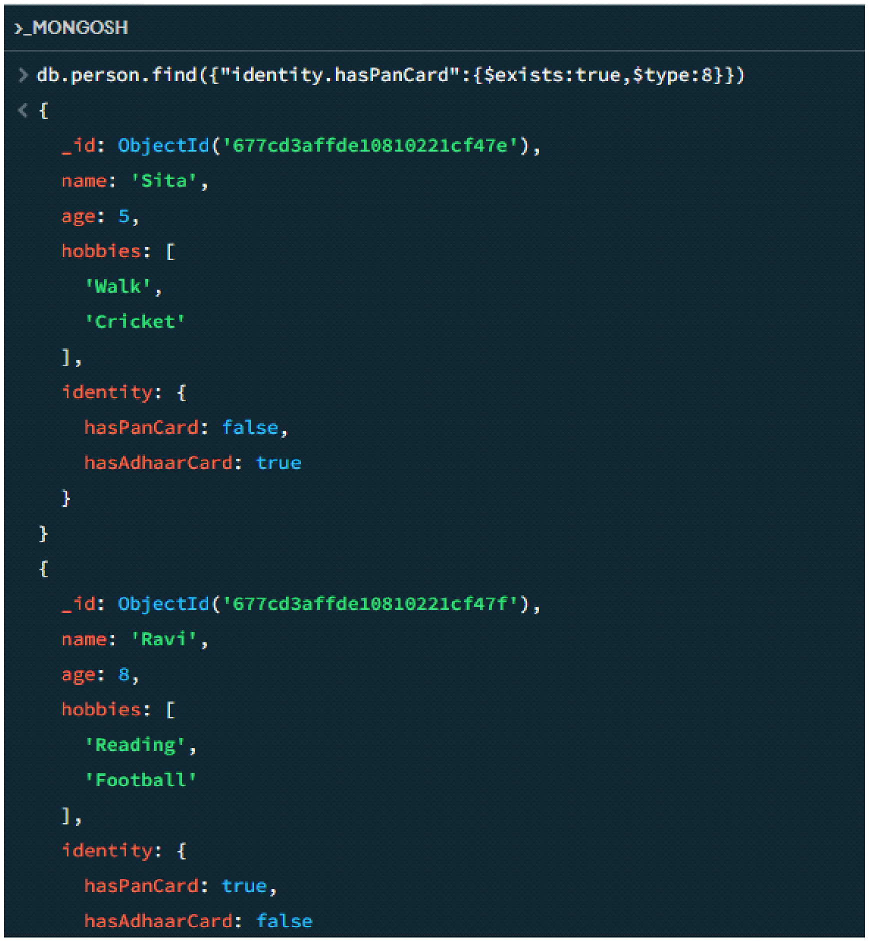
# MongoDB

Tuesday, January 7, 2025 9:20 PM

$exists :
The $exists operator matches documents that contain or do not contain a specified field, including documents where the field value is null.

{ field: { $exists: <boolean> } }

$type :
$type selects documents where the *value* of the field is an instance of the specified BSON type(s).

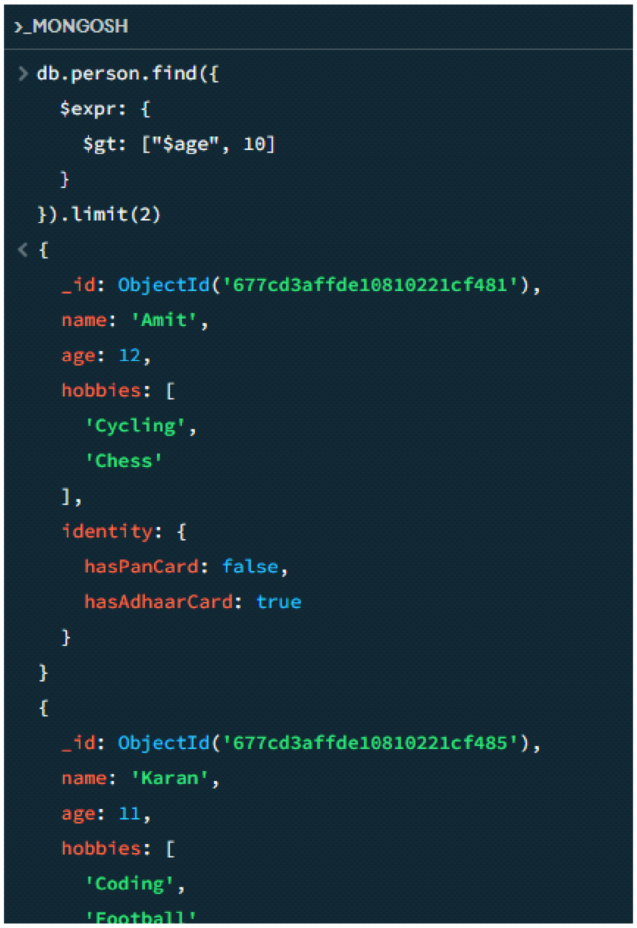


```
>_MONGOSH
> db.person.find({"identity.hasPanCard":{$exists:true,$type:8}})
< {
    _id: ObjectId('677cd3affde10810221cf47e'),
    name: 'Sita',
    age: 5,
    hobbies: [
      'Walk',
      'Cricket'
    ],
    identity: {
      hasPanCard: false,
      hasAdhaarCard: true
    }
  }
  {
    _id: ObjectId('677cd3affde10810221cf47f'),
    name: 'Ravi',
    age: 8,
    hobbies: [
      'Reading',
      'Football'
    ],
    identity: {
      hasPanCard: true,
      hasAdhaarCard: false
```

## Evaluation Query Operators

Evaluation operators return data based on evaluations of either individual fields or the entire collection's documents.

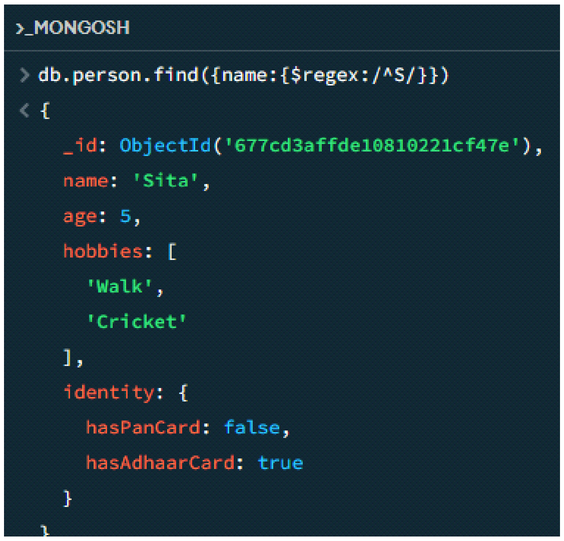
| Name | Description |
|---|---|
| $expr | Allows use of aggregation expressions within the query language. |
| $jsonSchema | Validate documents against the given JSON Schema. |
| $mod | Performs a modulo operation on the value of a field and selects documents with a specified result. |
| $regex | Selects documents where values match a specified regular expression. |
| $where | Matches documents that satisfy a JavaScript expression. |



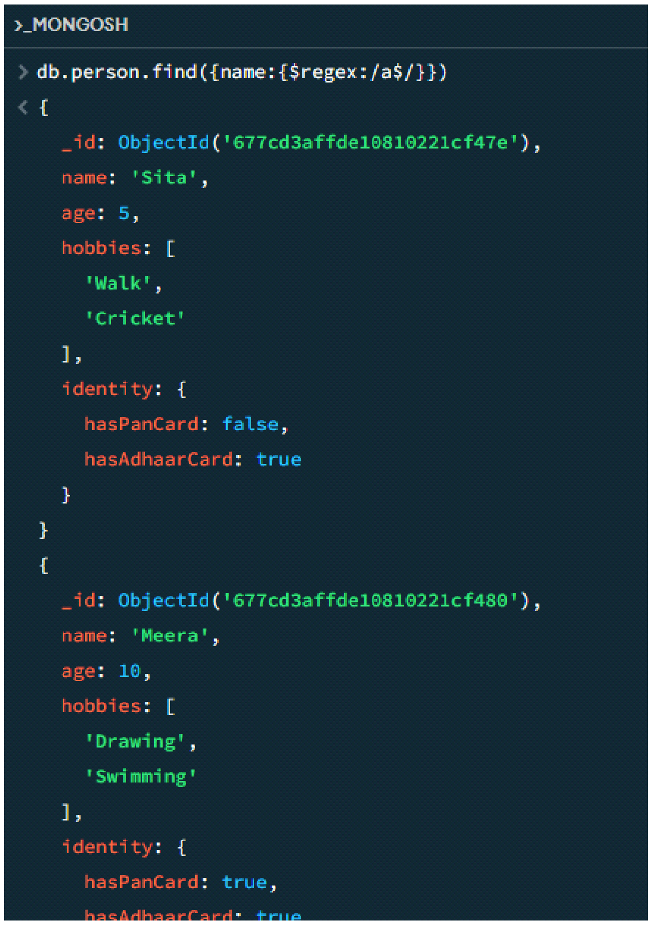
```
>_MONGOSH
> db.person.find({
    $expr: {
      $gt: ["$age", 10]
    }
  }).limit(2)
< {
    _id: ObjectId('677cd3affde10810221cf481'),
    name: 'Amit',
    age: 12,
    hobbies: [
      'Cycling',
      'Chess'
    ],
    identity: {
      hasPanCard: false,
      hasAdhaarCard: true
    }
  }
  {
    _id: ObjectId('677cd3affde10810221cf485'),
    name: 'Karan',
    age: 11,
    hobbies: [
      'Coding',
      'Football'
```

## $regex

Provides regular expression capabilities for pattern matching *strings* in queries.



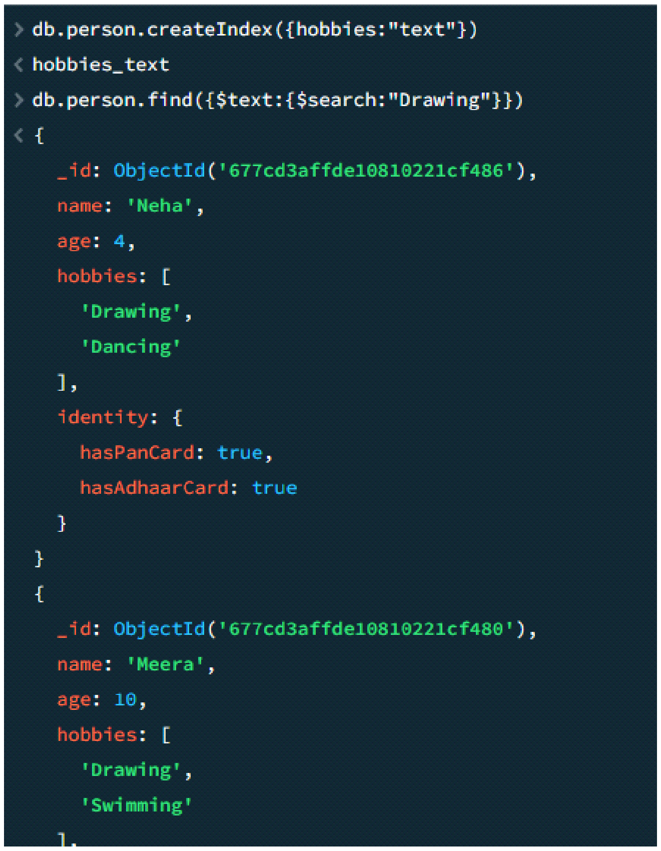
```
>_MONGOSH
> db.person.find({name:{$regex:/^S/}})
< {
    _id: ObjectId('677cd3affde10810221cf47e'),
    name: 'Sita',
    age: 5,
    hobbies: [
      'Walk',
      'Cricket'
    ],
    identity: {
      hasPanCard: false,
      hasAdhaarCard: true
    }
  }
```



```
>_MONGOSH
> db.person.find({name:{$regex:/a$/}})
< {
    _id: ObjectId('677cd3affde10810221cf47e'),
    name: 'Sita',
    age: 5,
    hobbies: [
      'Walk',
      'Cricket'
    ],
    identity: {
      hasPanCard: false,
      hasAdhaarCard: true
    }
  }
  {
    _id: ObjectId('677cd3affde10810221cf480'),
    name: 'Meera',
    age: 10,
    hobbies: [
      'Drawing',
      'Swimming'
    ],
    identity: {
      hasPanCard: true,
      hasAdhaarCard: true
```

**$text**

$text performs a text query on the content of the fields indexed with a text index.



```
> db.person.createIndex({hobbies:"text"})
< hobbies_text
> db.person.find({$text:{$search:"Drawing"}})
< {
    _id: ObjectId('677cd3affde10810221cf486'),
    name: 'Neha',
    age: 4,
    hobbies: [
      'Drawing',
      'Dancing'
    ],
    identity: {
      hasPanCard: true,
      hasAdhaarCard: true
    }
  }
  {
    _id: ObjectId('677cd3affde10810221cf480'),
    name: 'Meera',
    age: 10,
    hobbies: [
      'Drawing',
      'Swimming'
    ],
```

Sorting:

```
db.coolection_name.find().sort({field:-1/1})
```

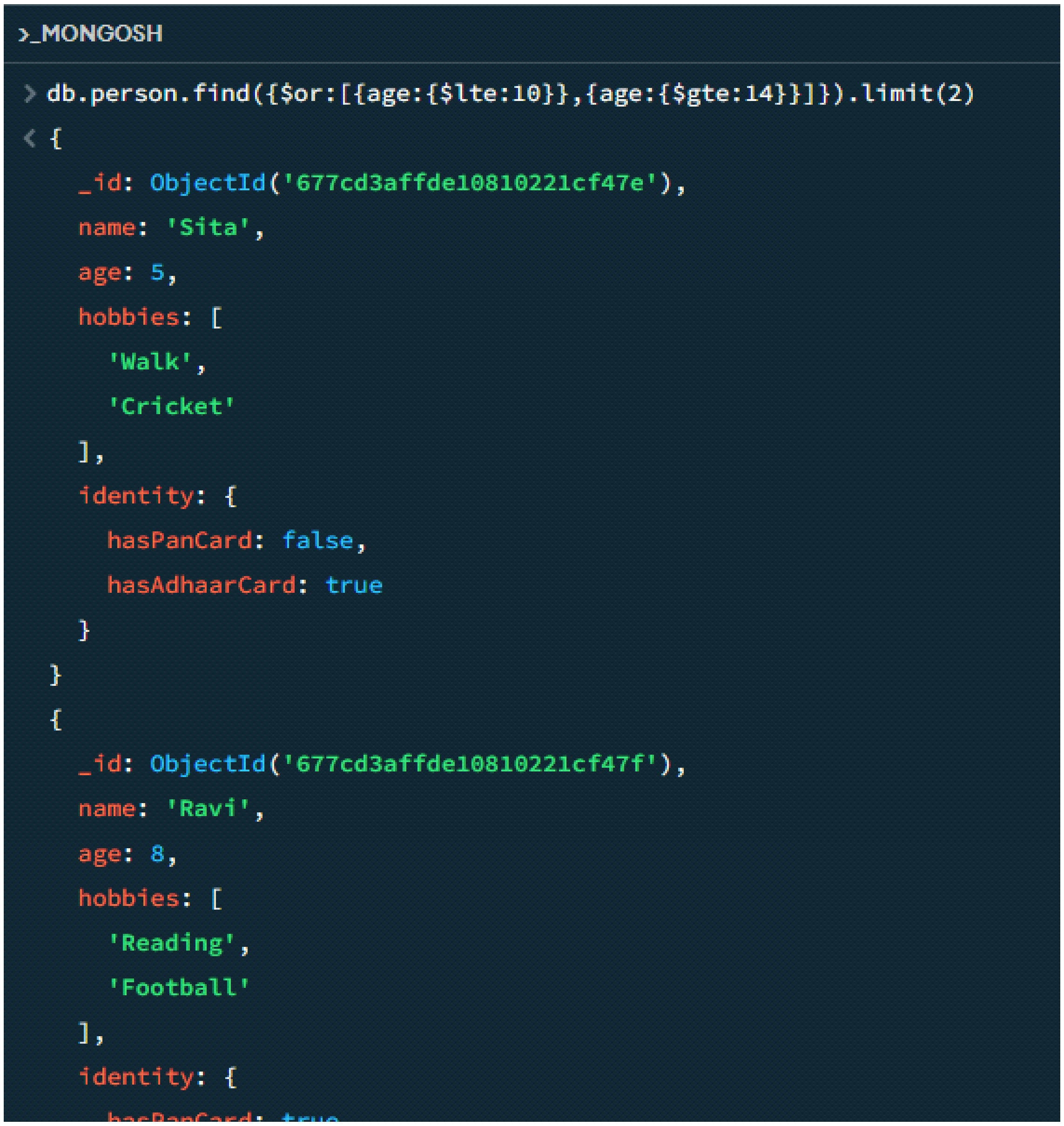
# MongoDB

Tuesday, January 7, 2025 9:00 PM

Logical Operator :

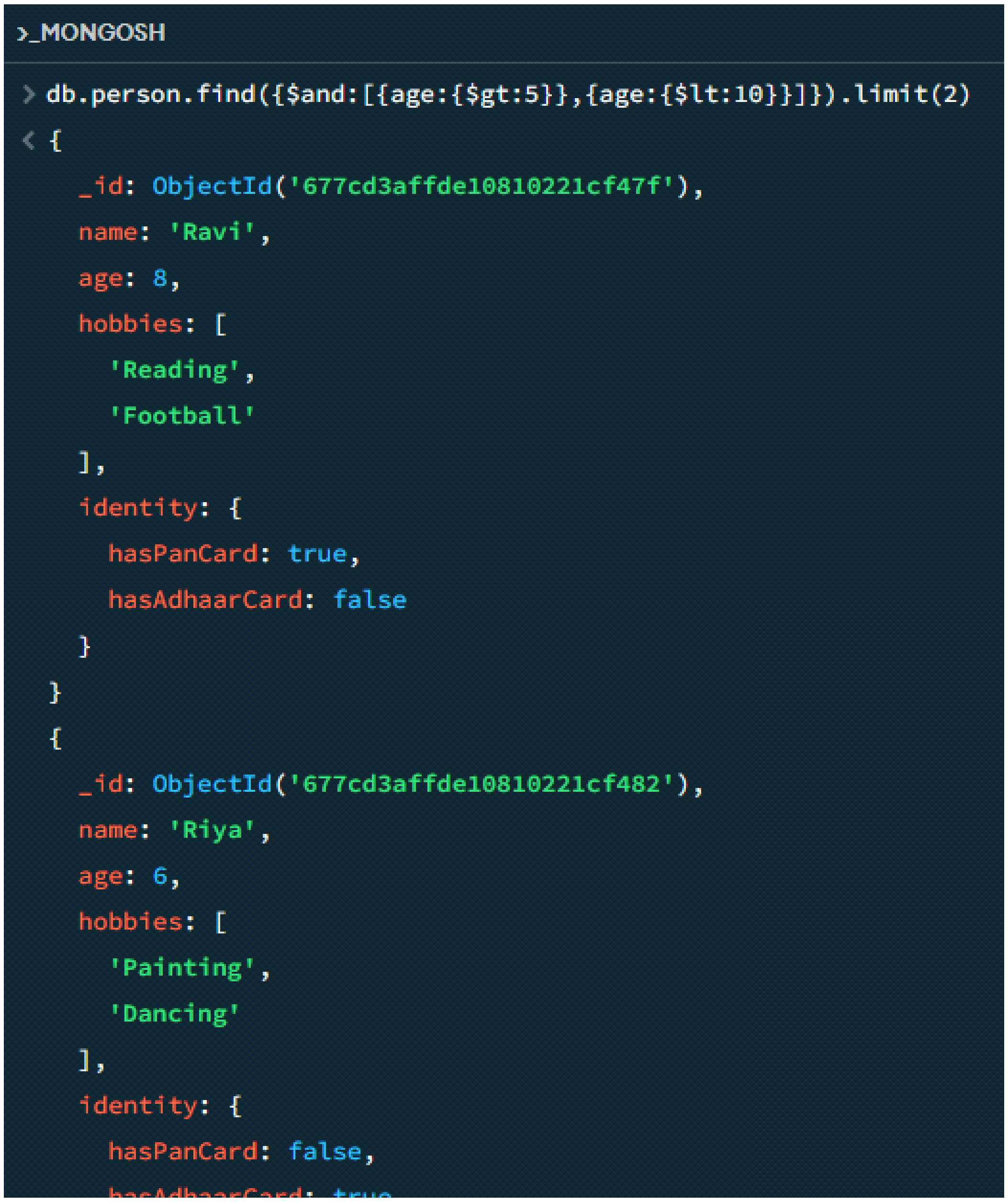
$and,$or,$not,$nor

$or :



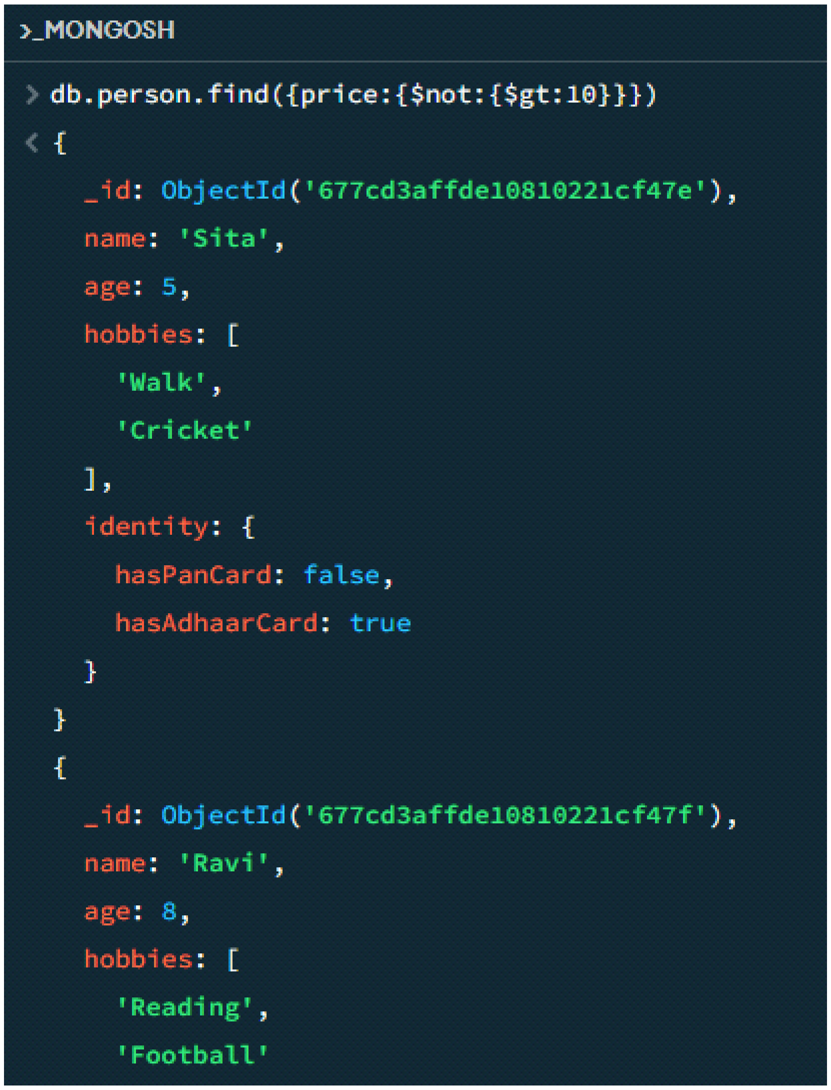
```
>_MONGOSH
> db.person.find({$or:[{age:{$lte:10}},{age:{$gte:14}}]}).limit(2)
< {
    _id: ObjectId('677cd3affde10810221cf47e'),
    name: 'Sita',
    age: 5,
    hobbies: [
      'Walk',
      'Cricket'
    ],
    identity: {
      hasPanCard: false,
      hasAdhaarCard: true
    }
  }
  {
    _id: ObjectId('677cd3affde10810221cf47f'),
    name: 'Ravi',
    age: 8,
    hobbies: [
      'Reading',
      'Football'
    ],
    identity: {
      hasPanCard: true
```

$and:



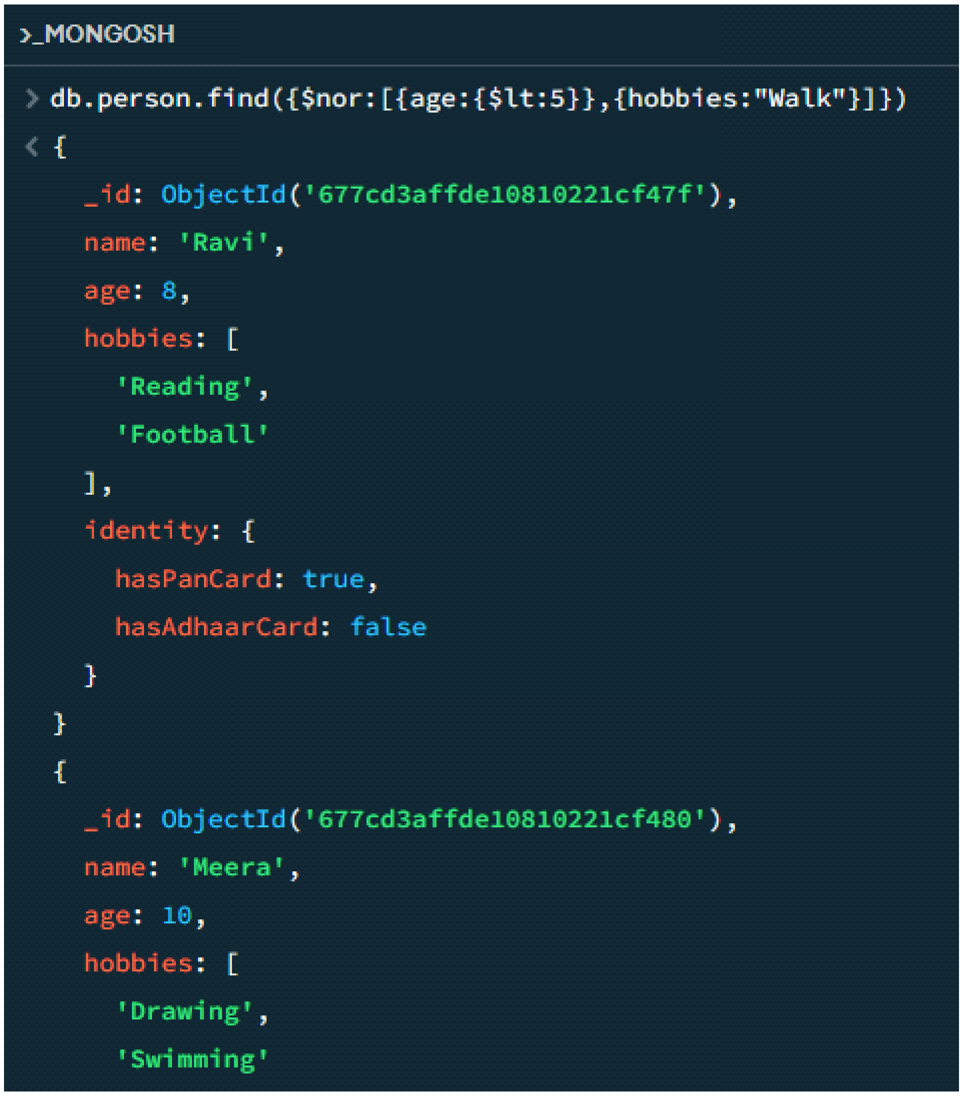
```
>_MONGOSH
> db.person.find({$and:[{age:{$gt:5}},{age:{$lt:10}}]}).limit(2)
< {
    _id: ObjectId('677cd3affde10810221cf47f'),
    name: 'Ravi',
    age: 8,
    hobbies: [
      'Reading',
      'Football'
    ],
    identity: {
      hasPanCard: true,
      hasAdhaarCard: false
    }
  }
  {
    _id: ObjectId('677cd3affde10810221cf482'),
    name: 'Riya',
    age: 6,
    hobbies: [
      'Painting',
      'Dancing'
    ],
    identity: {
      hasPanCard: false,
```

$not:



```
>_MONGOSH
> db.person.find({price:{$not:{$gt:10}}})
< {
    _id: ObjectId('677cd3affde10810221cf47e'),
    name: 'Sita',
    age: 5,
    hobbies: [
      'Walk',
      'Cricket'
    ],
    identity: {
      hasPanCard: false,
      hasAdhaarCard: true
    }
  }
  {
    _id: ObjectId('677cd3affde10810221cf47f'),
    name: 'Ravi',
    age: 8,
    hobbies: [
      'Reading',
      'Football'
```

$nor:


```
>_MONGOSH
> db.person.find({$nor:[{age:{$lt:5}},{hobbies:"Walk"}]})
< {
    _id: ObjectId('677cd3affde10810221cf47f'),
    name: 'Ravi',
    age: 8,
    hobbies: [
      'Reading',
      'Football'
    ],
    identity: {
      hasPanCard: true,
      hasAdhaarCard: false
    }
  }
  {
    _id: ObjectId('677cd3affde10810221cf480'),
    name: 'Meera',
    age: 10,
    hobbies: [
      'Drawing',
      'Swimming'
```

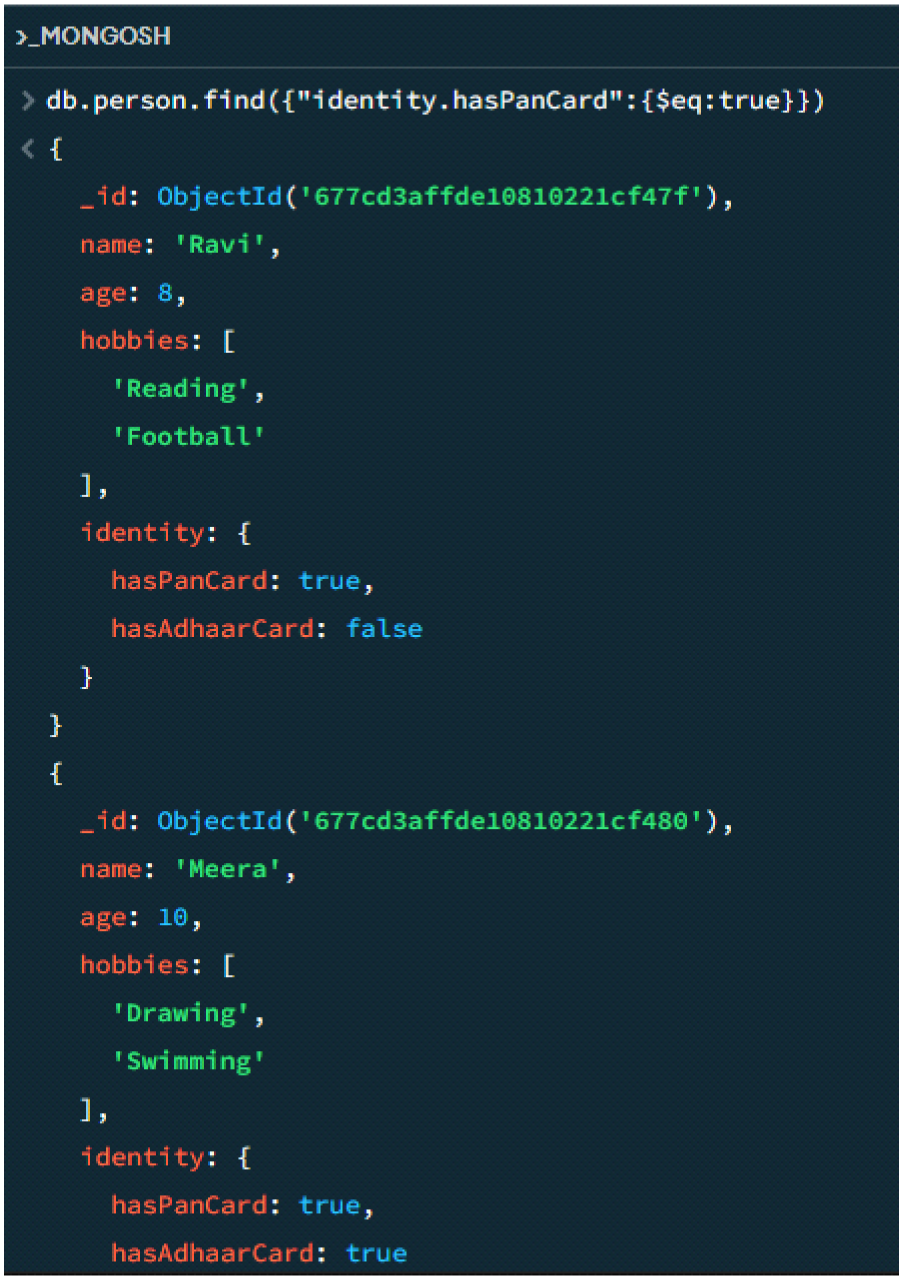
# MongoDB

Tuesday, January 7, 2025 12:30 PM

Comparison Operator :

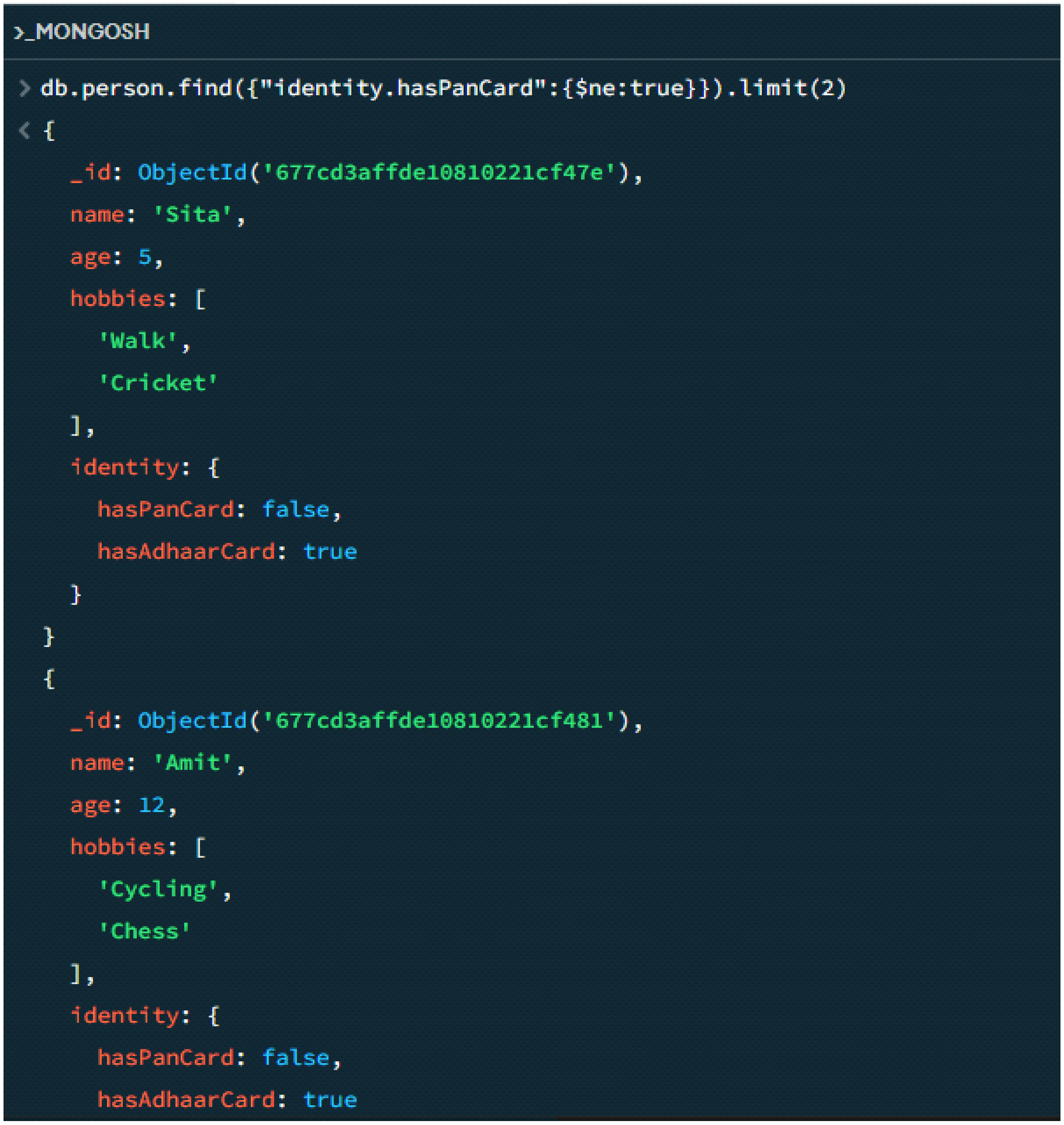
| | |
|---|---|
| $eq | equal to |
| $gt | greater than |
| $gte | greater than or equal to |
| $lt | less than |
| $lte | less than or equal to |
| $ne | not equal to |
| $nin | not in |
| $in | in |

$eq :



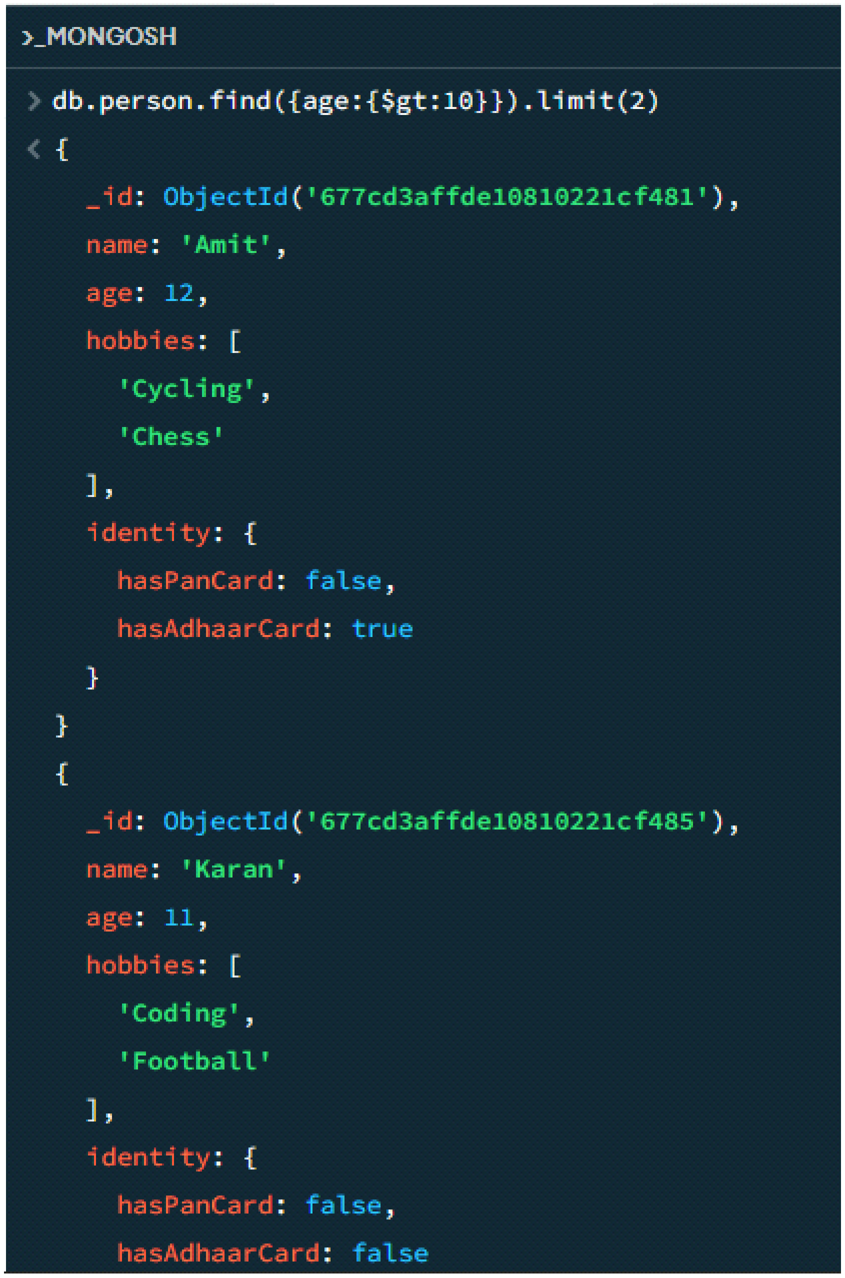
```
>_MONGOSH
> db.person.find({"identity.hasPanCard":{$eq:true}})
< {
    _id: ObjectId('677cd3affde10810221cf47f'),
    name: 'Ravi',
    age: 8,
    hobbies: [
      'Reading',
      'Football'
    ],
    identity: {
      hasPanCard: true,
      hasAdhaarCard: false
    }
  }
  {
    _id: ObjectId('677cd3affde10810221cf480'),
    name: 'Meera',
    age: 10,
    hobbies: [
      'Drawing',
      'Swimming'
    ],
    identity: {
      hasPanCard: true,
      hasAdhaarCard: true
```

$ne :



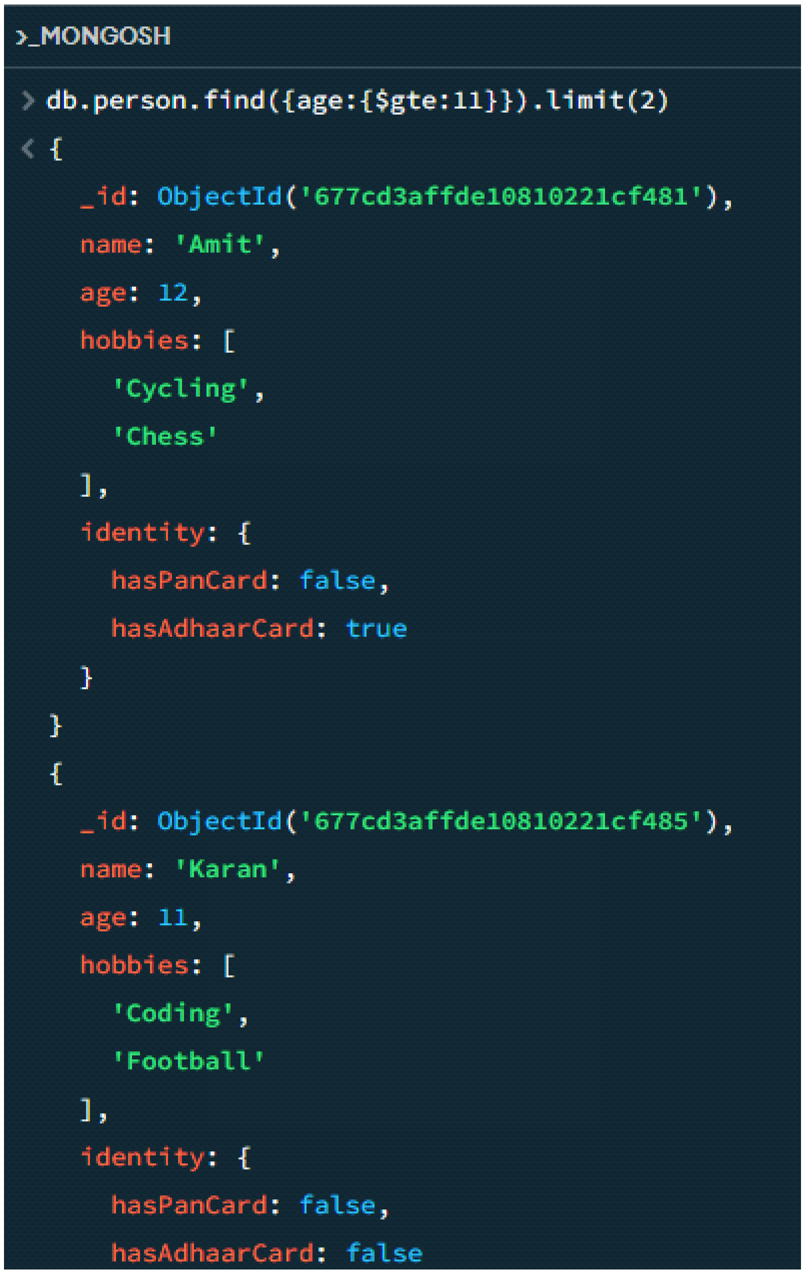
```
>_MONGOSH
> db.person.find({"identity.hasPanCard":{$ne:true}}).limit(2)
< {
    _id: ObjectId('677cd3affde10810221cf47e'),
    name: 'Sita',
    age: 5,
    hobbies: [
      'Walk',
      'Cricket'
    ],
    identity: {
      hasPanCard: false,
      hasAdhaarCard: true
    }
  }
  {
    _id: ObjectId('677cd3affde10810221cf481'),
    name: 'Amit',
    age: 12,
    hobbies: [
      'Cycling',
      'Chess'
    ],
    identity: {
      hasPanCard: false,
      hasAdhaarCard: true
```

$gt :



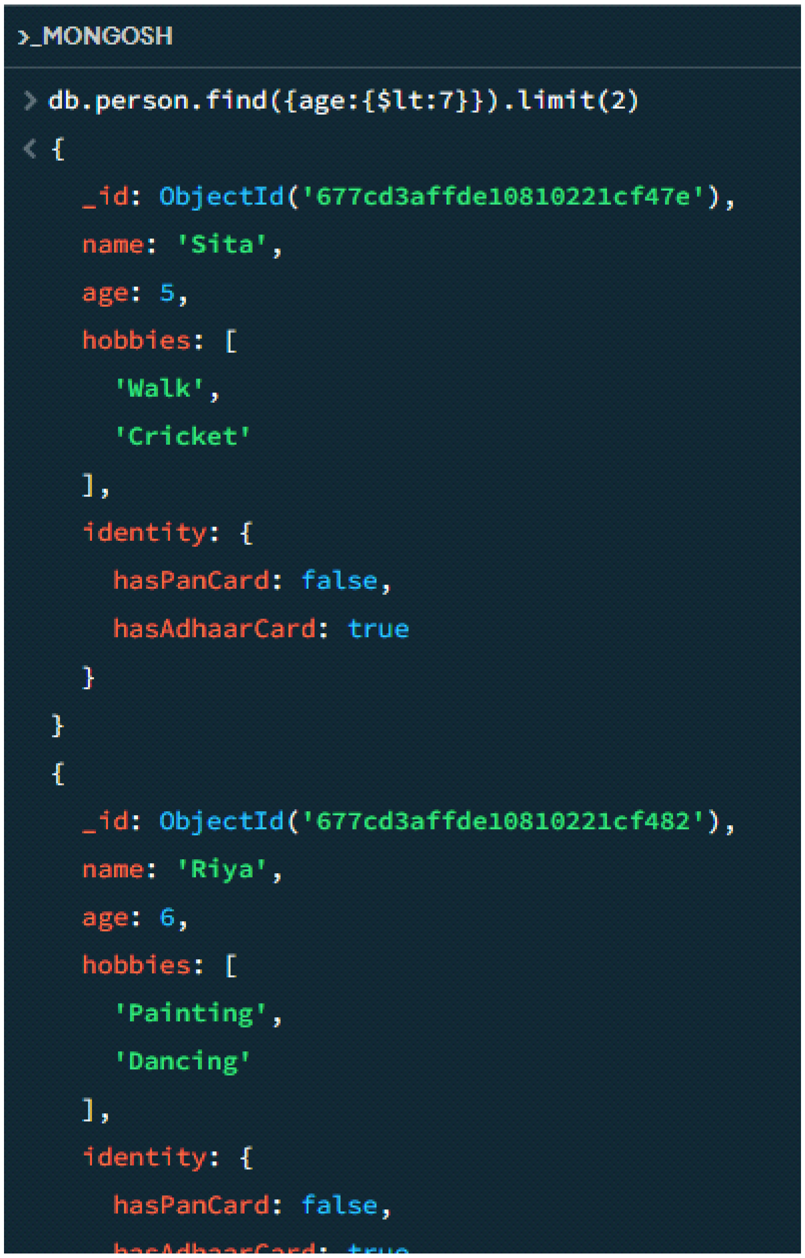
```
>_MONGOSH
> db.person.find({age:{$gt:10}}).limit(2)
< {
    _id: ObjectId('677cd3affde10810221cf481'),
    name: 'Amit',
    age: 12,
    hobbies: [
      'Cycling',
      'Chess'
    ],
    identity: {
      hasPanCard: false,
      hasAdhaarCard: true
    }
  }
  {
    _id: ObjectId('677cd3affde10810221cf485'),
    name: 'Karan',
    age: 11,
    hobbies: [
      'Coding',
      'Football'
    ],
    identity: {
      hasPanCard: false,
      hasAdhaarCard: false
```

$gte :



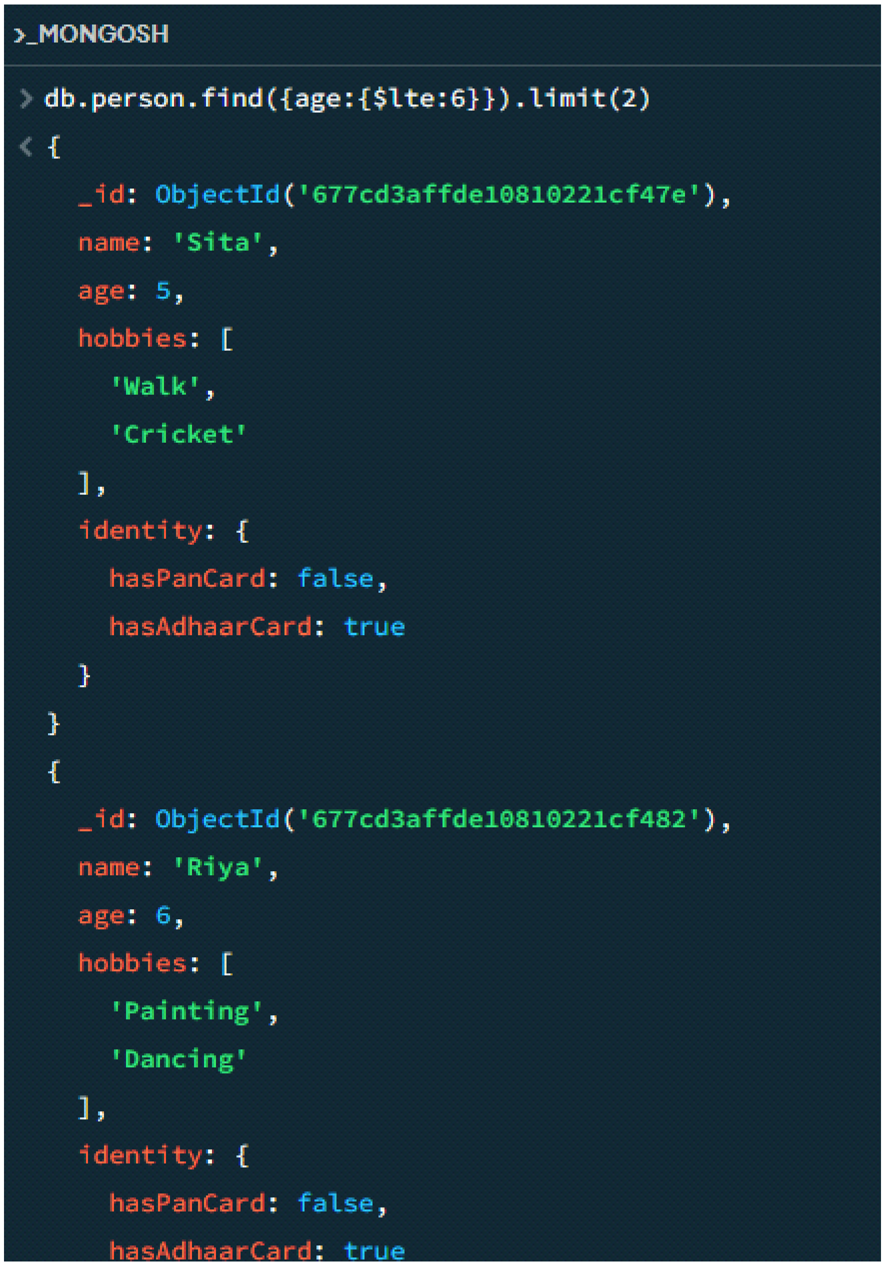
```
>_MONGOSH
> db.person.find({age:{$gte:11}}).limit(2)
< {
    _id: ObjectId('677cd3affde10810221cf481'),
    name: 'Amit',
    age: 12,
    hobbies: [
      'Cycling',
      'Chess'
    ],
    identity: {
      hasPanCard: false,
      hasAdhaarCard: true
    }
  }
  {
    _id: ObjectId('677cd3affde10810221cf485'),
    name: 'Karan',
    age: 11,
    hobbies: [
      'Coding',
      'Football'
    ],
    identity: {
      hasPanCard: false,
      hasAdhaarCard: false
```

$lt:



```
> db.person.find({age:{$lt:7}}).limit(2)
< {
    _id: ObjectId('677cd3affde10810221cf47e'),
    name: 'Sita',
    age: 5,
    hobbies: [
      'Walk',
      'Cricket'
    ],
    identity: {
      hasPanCard: false,
      hasAdhaarCard: true
    }
  }
  {
    _id: ObjectId('677cd3affde10810221cf482'),
    name: 'Riya',
    age: 6,
    hobbies: [
      'Painting',
      'Dancing'
    ],
    identity: {
      hasPanCard: false,
```

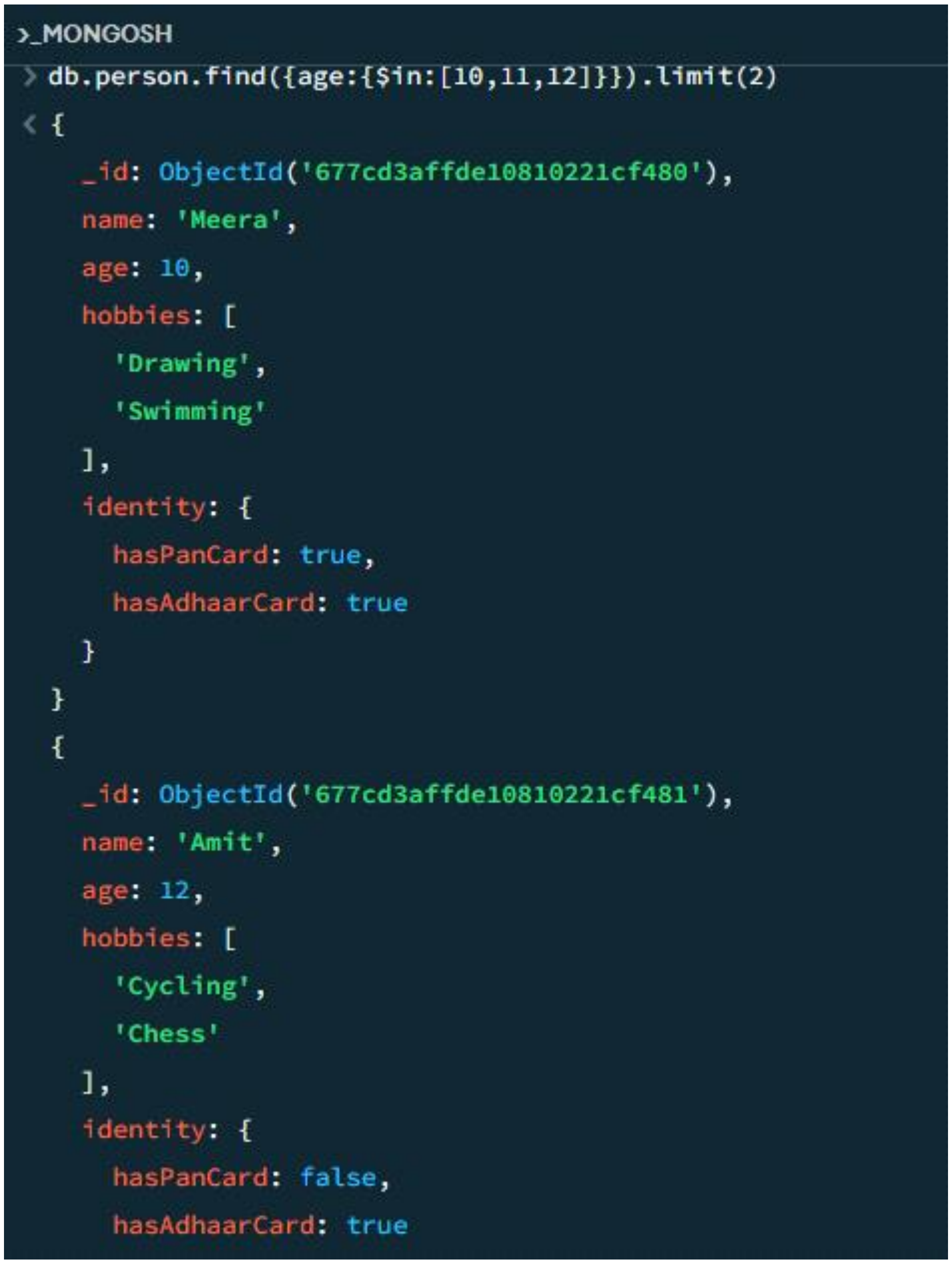
$lte :



>_MONGOSH

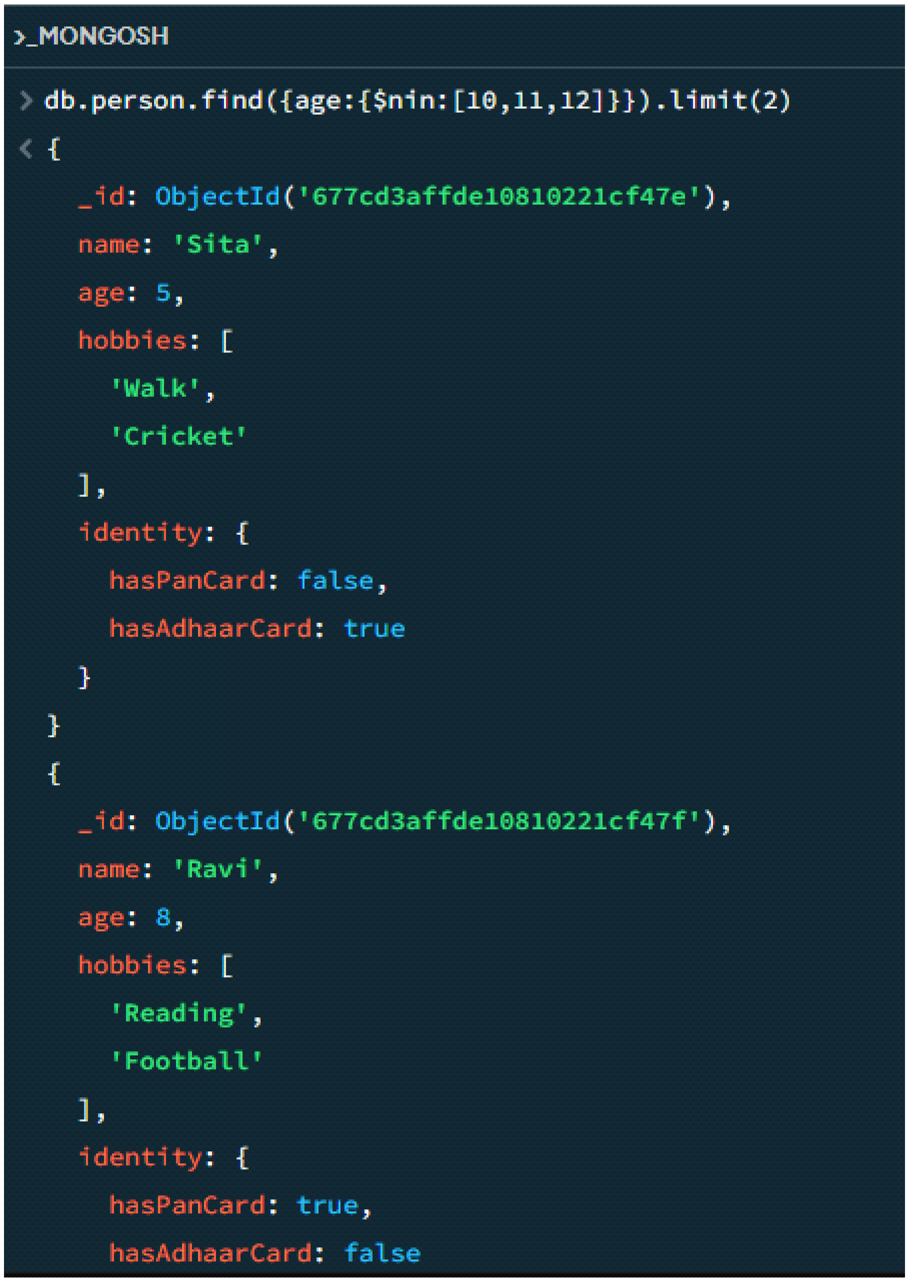
```
> db.person.find({age:{$lte:6}}).limit(2)
< {
    _id: ObjectId('677cd3affde10810221cf47e'),
    name: 'Sita',
    age: 5,
    hobbies: [
      'Walk',
      'Cricket'
    ],
    identity: {
      hasPanCard: false,
      hasAdhaarCard: true
    }
  }
  {
    _id: ObjectId('677cd3affde10810221cf482'),
    name: 'Riya',
    age: 6,
    hobbies: [
      'Painting',
      'Dancing'
    ],
    identity: {
      hasPanCard: false,
      hasAdhaarCard: true
```

$in :



```
>_MONGOSH
> db.person.find({age:{$in:[10,11,12]}}).limit(2)
< {
    _id: ObjectId('677cd3affde10810221cf480'),
    name: 'Meera',
    age: 10,
    hobbies: [
      'Drawing',
      'Swimming'
    ],
    identity: {
      hasPanCard: true,
      hasAdhaarCard: true
    }
  }
  {
    _id: ObjectId('677cd3affde10810221cf481'),
    name: 'Amit',
    age: 12,
    hobbies: [
      'Cycling',
      'Chess'
    ],
    identity: {
      hasPanCard: false,
      hasAdhaarCard: true
```

$nin :

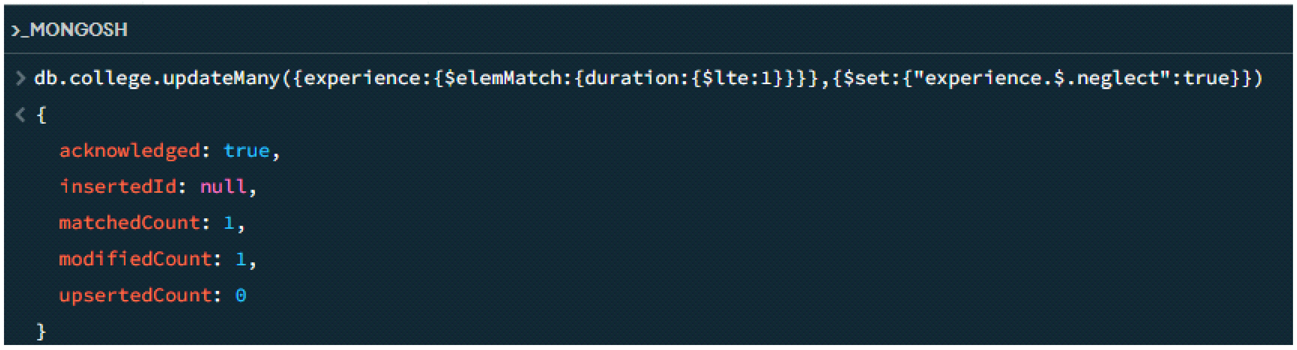


```
>_MONGOSH
> db.person.find({age:{$nin:[10,11,12]}}).limit(2)
< {
    _id: ObjectId('677cd3affde10810221cf47e'),
    name: 'Sita',
    age: 5,
    hobbies: [
      'Walk',
      'Cricket'
    ],
    identity: {
      hasPanCard: false,
      hasAdhaarCard: true
    }
  }
  {
    _id: ObjectId('677cd3affde10810221cf47f'),
    name: 'Ravi',
    age: 8,
    hobbies: [
      'Reading',
      'Football'
    ],
    identity: {
      hasPanCard: true,
      hasAdhaarCard: false
```

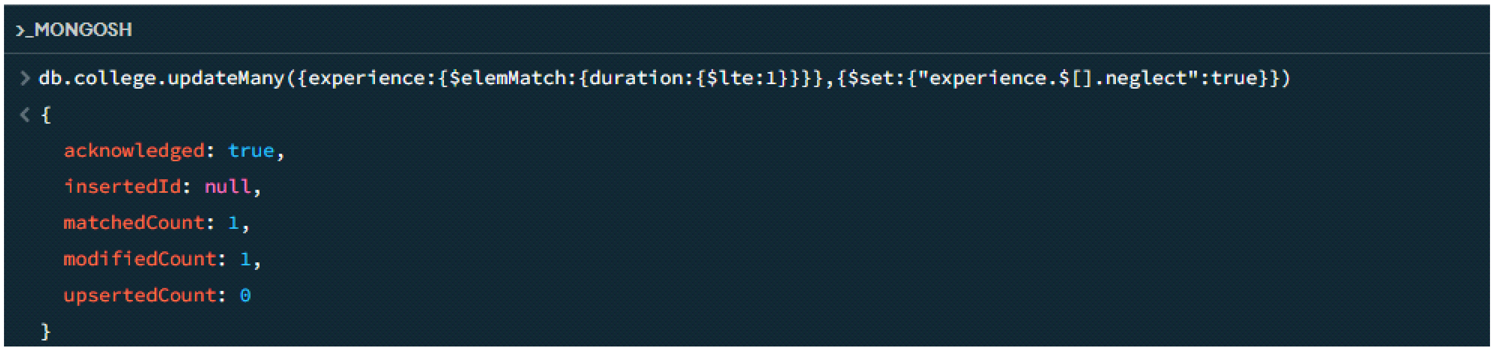
# Array Update

Wednesday, January 8, 2025 8:34 AM

Add new field in first matching collection :



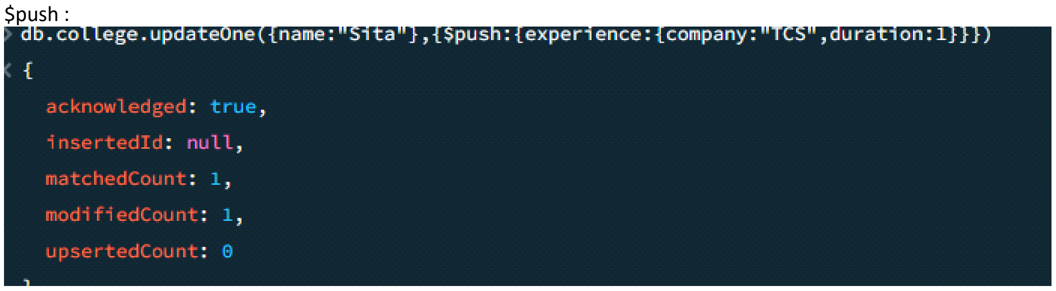
```
>_MONGOSH
> db.college.updateMany({experience:{$elemMatch:{duration:{$lte:1}}}},{$set:{"experience.$.neglect":true}})
< {
    acknowledged: true,
    insertedId: null,
    matchedCount: 1,
    modifiedCount: 1,
    upsertedCount: 0
  }
```



```
>_MONGOSH
> db.college.updateMany({experience:{$elemMatch:{duration:{$lte:1}}}},{$set:{"experience.$[].neglect":true}})
< {
    acknowledged: true,
    insertedId: null,
    matchedCount: 1,
    modifiedCount: 1,
    upsertedCount: 0
  }
```

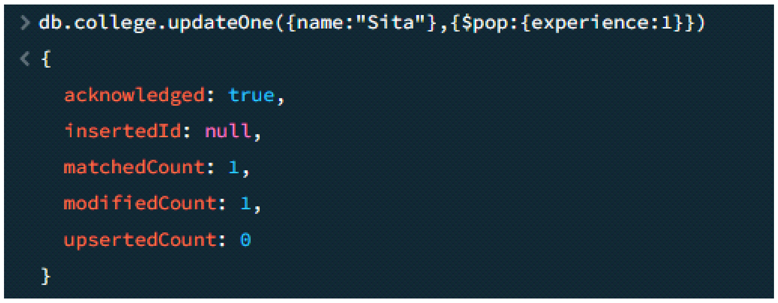
For all

$push :



```
> db.college.updateOne({name:"Sita"},{$push:{experience:{company:"TCS",duration:1}}})
< {
    acknowledged: true,
    insertedId: null,
    matchedCount: 1,
    modifiedCount: 1,
    upsertedCount: 0
```

$pop :



```
> db.college.updateOne({name:"Sita"},{$pop:{experience:1}})
< {
    acknowledged: true,
    insertedId: null,
    matchedCount: 1,
    modifiedCount: 1,
    upsertedCount: 0
  }
```

## $pull

The $pull operator removes from an existing array all instances of a value or values that match a specified condition.

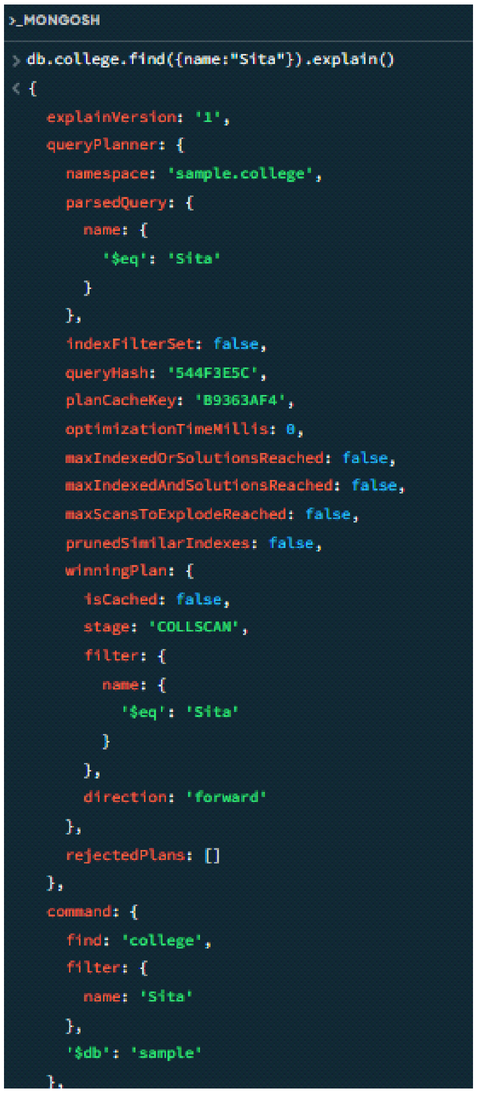
```
db.stores.updateMany(
   { },
   { $pull: { fruits: { $in: [ "apples", "oranges" ] }, vegetables: "carrots" } }
)
```

# Indexing

Wednesday, January 8, 2025 9:14 AM

1. Index is a data structure that store index in sorted manner and point corresponding document.
2. Indexes are stored in B-Trees data structure
3. When a query is executed, mongoDB can use the index to quickely locate the document that match the query by searching through the B-Tree
   - The trade-off
     - Storage space
     - Write performance
4. Types of indexes:

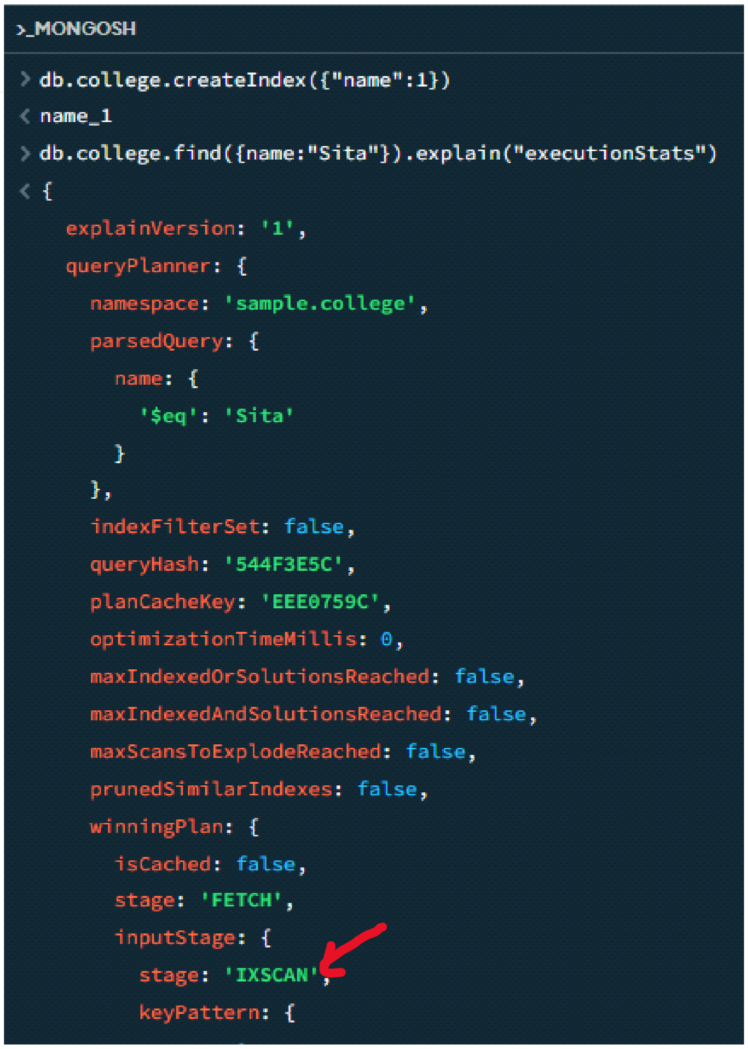
- Single field indexes
- Compound indexes
- Text indexes

5. 

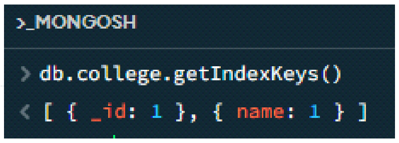
```
>_MONGOSH
> db.college.find({name:"Sita"}).explain()
< {
    explainVersion: '1',
    queryPlanner: {
      namespace: 'sample.college',
      parsedQuery: {
        name: {
          '$eq': 'Sita'
        }
      },
      indexFilterSet: false,
      queryHash: '544F3E5C',
      planCacheKey: 'B9363AF4',
      optimizationTimeMillis: 0,
      maxIndexedOrSolutionsReached: false,
      maxIndexedAndSolutionsReached: false,
      maxScansToExplodeReached: false,
      prunedSimilarIndexes: false,
      winningPlan: {
        isCached: false,
        stage: 'COLLSCAN',
        filter: {
          name: {
            '$eq': 'Sita'
          }
        },
        direction: 'forward'
      },
      rejectedPlans: []
    },
    command: {
      find: 'college',
      filter: {
        name: 'Sita'
      },
      '$db': 'sample'
    },
```

```
db.college.find({name:"Sita"}).explain("executionStats")
```

How to create index



```
>_MONGOSH
> db.college.createIndex({"name":1})
< name_1
> db.college.find({name:"Sita"}).explain("executionStats")
< {
    explainVersion: '1',
    queryPlanner: {
      namespace: 'sample.college',
      parsedQuery: {
        name: {
          '$eq': 'Sita'
        }
      },
      indexFilterSet: false,
      queryHash: '544F3E5C',
      planCacheKey: 'EEE0759C',
      optimizationTimeMillis: 0,
      maxIndexedOrSolutionsReached: false,
      maxIndexedAndSolutionsReached: false,
      maxScansToExplodeReached: false,
      prunedSimilarIndexes: false,
      winningPlan: {
        isCached: false,
        stage: 'FETCH',
        inputStage: {
          stage: 'IXSCAN',
          keyPattern: {
```

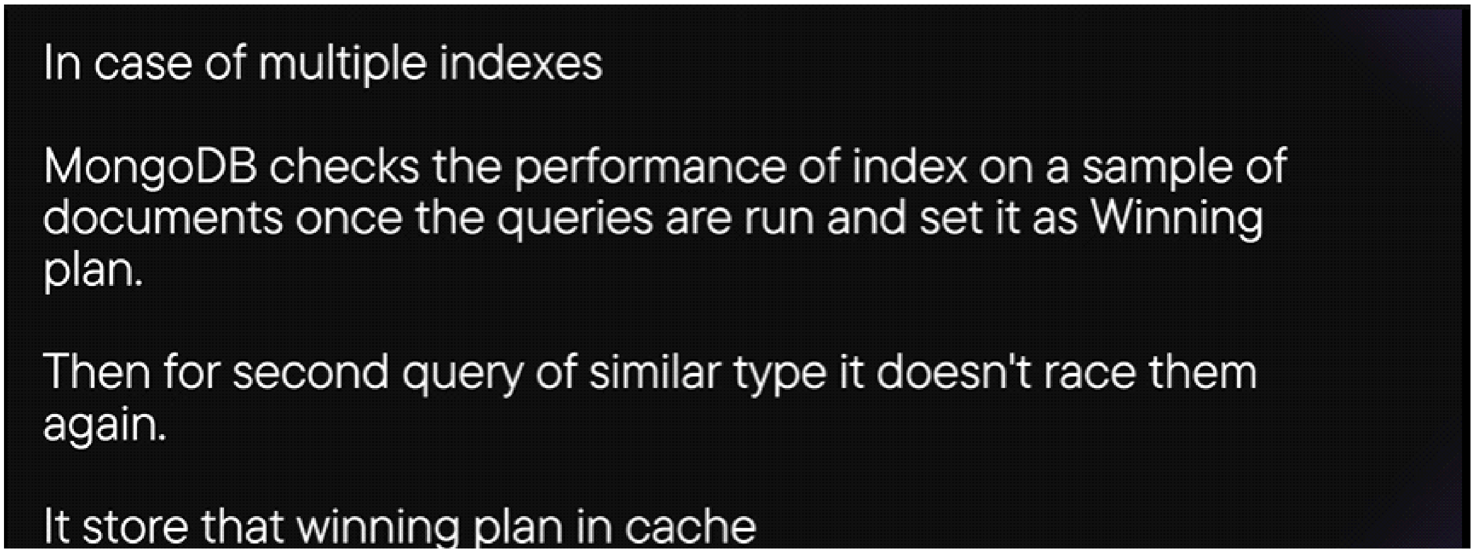


```
>_MONGOSH
> db.college.getIndexKeys()
< [ { _id: 1 }, { name: 1 } ]
```

How to delete indexing on given field
db.collectionname.dropIndex("fieldname")

When not to use indexing in mongodb?

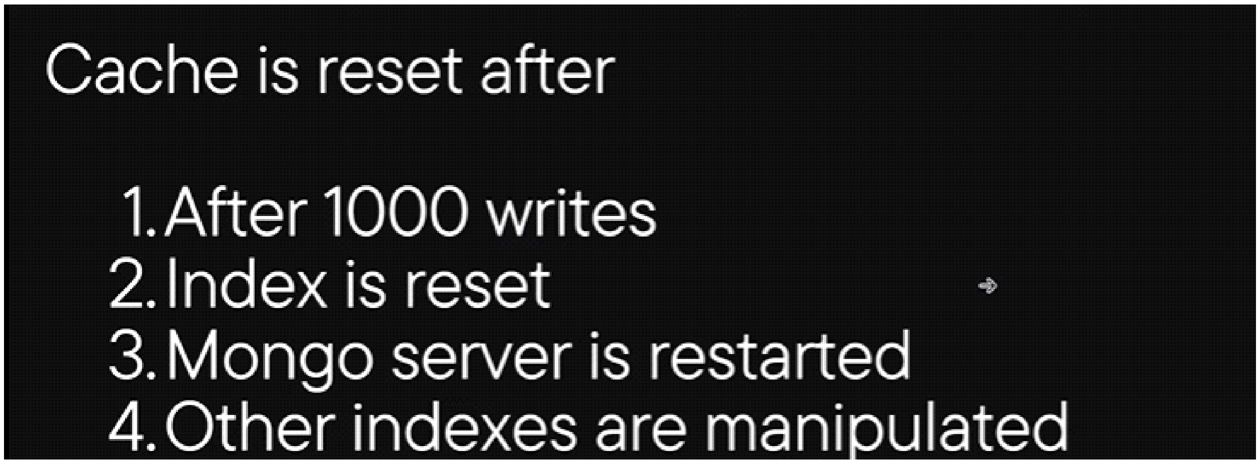
- Collection is small
- When collection is frequently updated
- When the queries are complex
- When collection is large.


In case of multiple indexes

MongoDB checks the performance of index on a sample of documents once the queries are run and set it as Winning plan.

Then for second query of similar type it doesn't race them again.

It store that winning plan in cache


Cache is reset after

1. After 1000 writes
2. Index is reset
3. Mongo server is restarted
4. Other indexes are manipulated

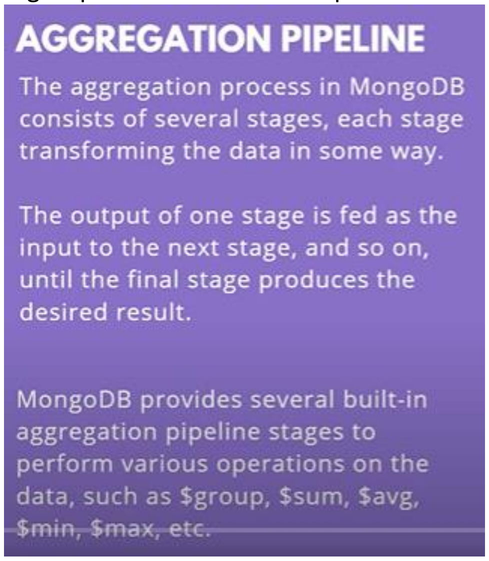
# Aggregation

Wednesday, January 8, 2025 12:36 PM

What is Aggregation?

To write Aggregate query

A pipeline Operation

- It groups the data from multiple documents into a single document based on the specified expression.
- 

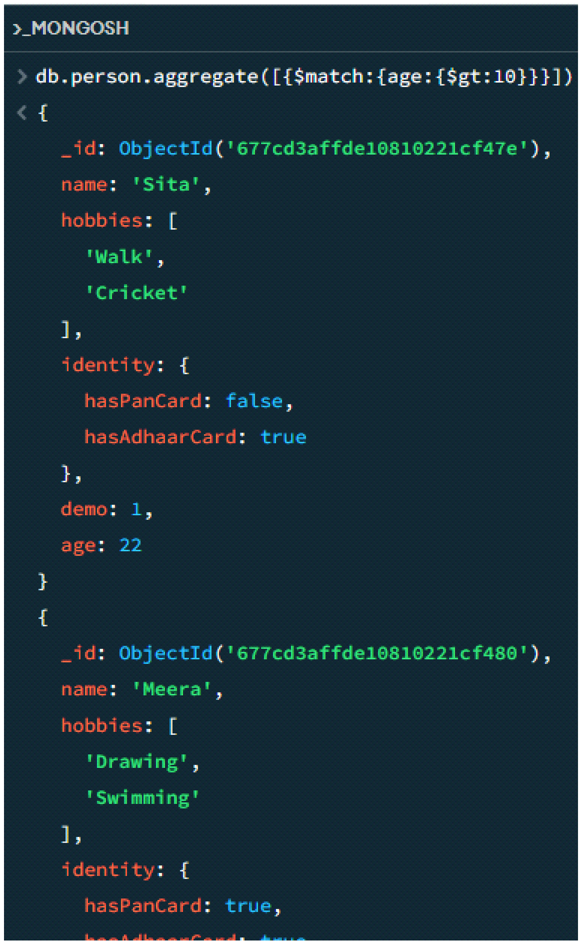
  **AGGREGATION PIPELINE**

  The aggregation process in MongoDB consists of several stages, each stage transforming the data in some way.

  The output of one stage is fed as the input to the next stage, and so on, until the final stage produces the desired result.

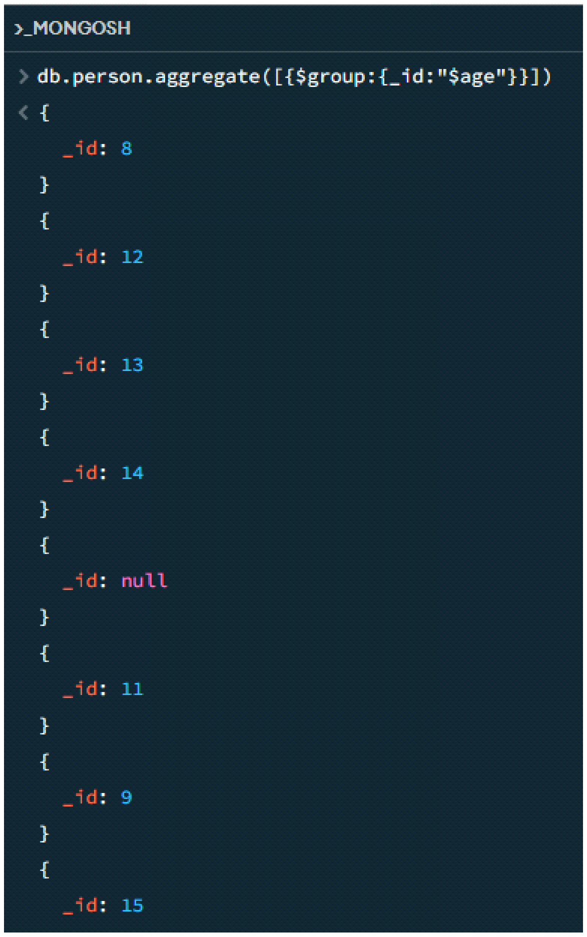
  MongoDB provides several built-in aggregation pipeline stages to perform various operations on the data, such as $group, $sum, $avg, $min, $max, etc.
- db.collection.aggregate(pipeline,option)

$match :



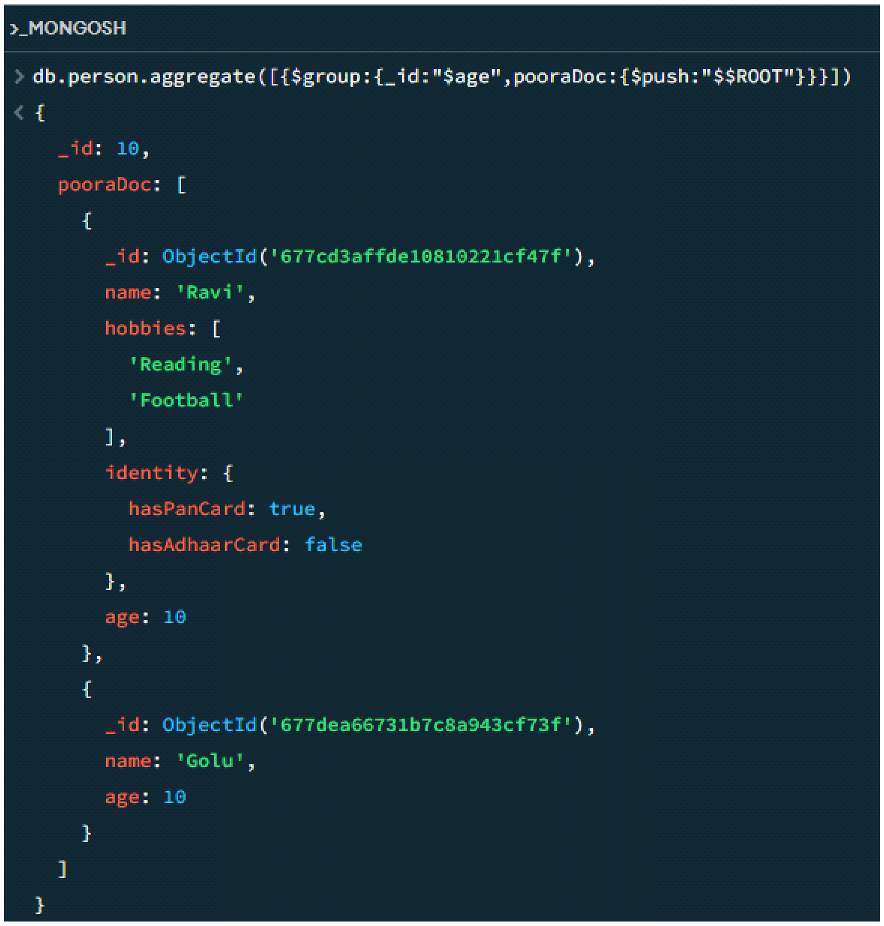
```
>_MONGOSH
> db.person.aggregate([{$match:{age:{$gt:10}}}])
< {
    _id: ObjectId('677cd3affde10810221cf47e'),
    name: 'Sita',
    hobbies: [
      'Walk',
      'Cricket'
    ],
    identity: {
      hasPanCard: false,
      hasAdhaarCard: true
    },
    demo: 1,
    age: 22
  }
  {
    _id: ObjectId('677cd3affde10810221cf480'),
    name: 'Meera',
    hobbies: [
      'Drawing',
      'Swimming'
    ],
    identity: {
      hasPanCard: true,
```

$group :

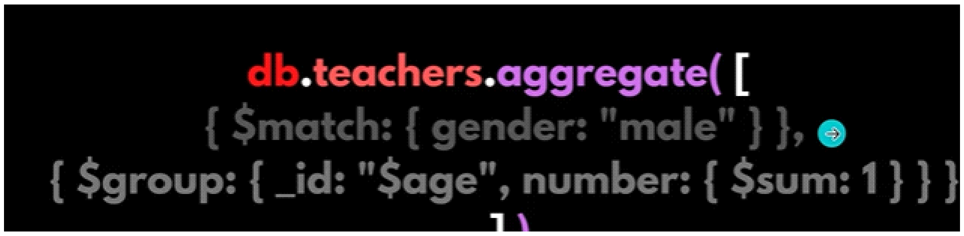


```
>_MONGOSH
> db.person.aggregate([{$group:{_id:"$age"}}])
< {
    _id: 8
  }
  {
    _id: 12
  }
  {
    _id: 13
  }
  {
    _id: 14
  }
  {
    _id: null
  }
  {
    _id: 11
  }
  {
    _id: 9
  }
  {
    _id: 15
```

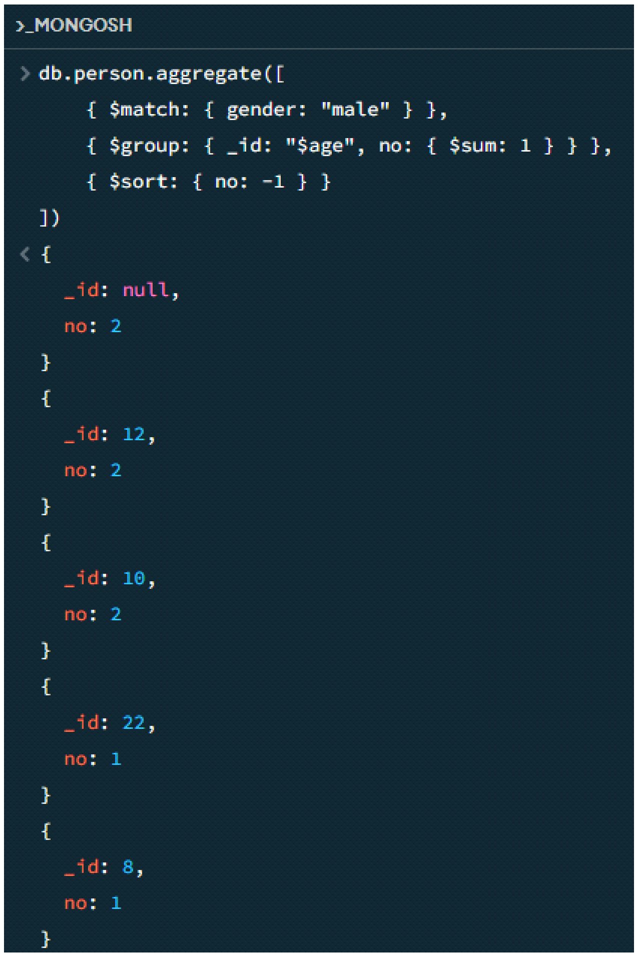
Get all data :



```
>_MONGOSH
> db.person.aggregate([{$group:{_id:"$age",pooraDoc:{$push:"$$ROOT"}}}])
< {
    _id: 10,
    pooraDoc: [
      {
        _id: ObjectId('677cd3affde10810221cf47f'),
        name: 'Ravi',
        hobbies: [
          'Reading',
          'Football'
        ],
        identity: {
          hasPanCard: true,
          hasAdhaarCard: false
        },
        age: 10
      },
      {
        _id: ObjectId('677dea66731b7c8a943cf73f'),
        name: 'Golu',
        age: 10
      }
    ]
  }
```



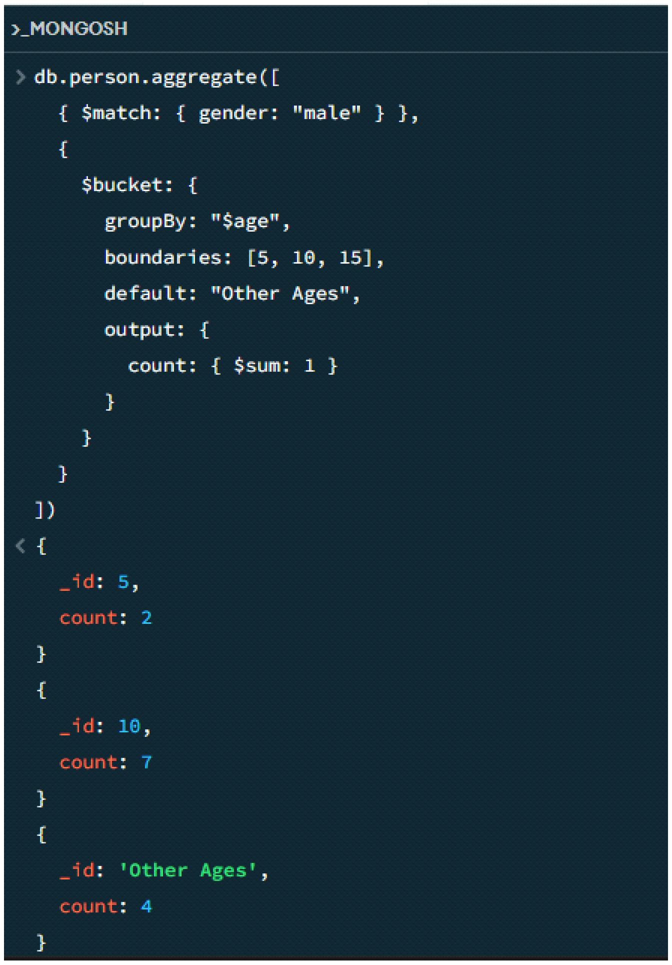
```
db.teachers.aggregate( [
      { $match: { gender: "male" } },
{ $group: { _id: "$age", number: { $sum: 1 } } }
])
```


```
>_MONGOSH
> db.person.aggregate([
      { $match: { gender: "male" } },
      { $group: { _id: "$age", no: { $sum: 1 } } },
      { $sort: { no: -1 } }
  ])
< {
    _id: null,
    no: 2
  }
  {
    _id: 12,
    no: 2
  }
  {
    _id: 10,
    no: 2
  }
  {
    _id: 22,
    no: 1
  }
  {
    _id: 8,
    no: 1
  }
```

$bucket:

Categorizes incoming documents into groups, called buckets, based on a specified expression and bucket boundaries and outputs a document per each bucket.

When use ?
When you want to categorize into discreate group based on specified boundries.



>_MONGOSH

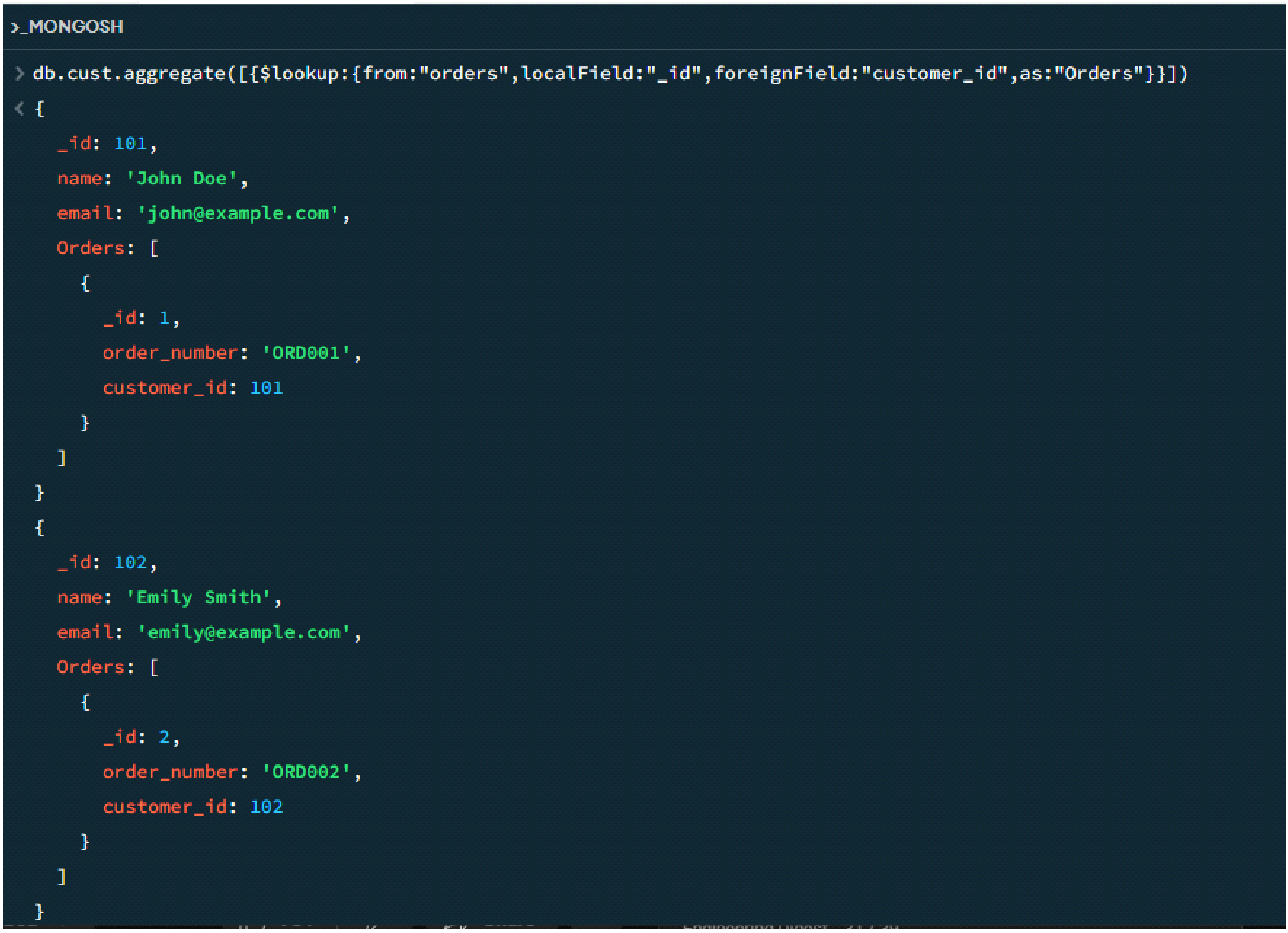
```
> db.person.aggregate([
    { $match: { gender: "male" } },
    {
      $bucket: {
        groupBy: "$age",
        boundaries: [5, 10, 15],
        default: "Other Ages",
        output: {
          count: { $sum: 1 }
        }
      }
    }
  ])
< {
    _id: 5,
    count: 2
  }
  {
    _id: 10,
    count: 7
  }
  {
    _id: 'Other Ages',
    count: 4
  }
```

# Join

Thursday, January 9, 2025 9:08 AM

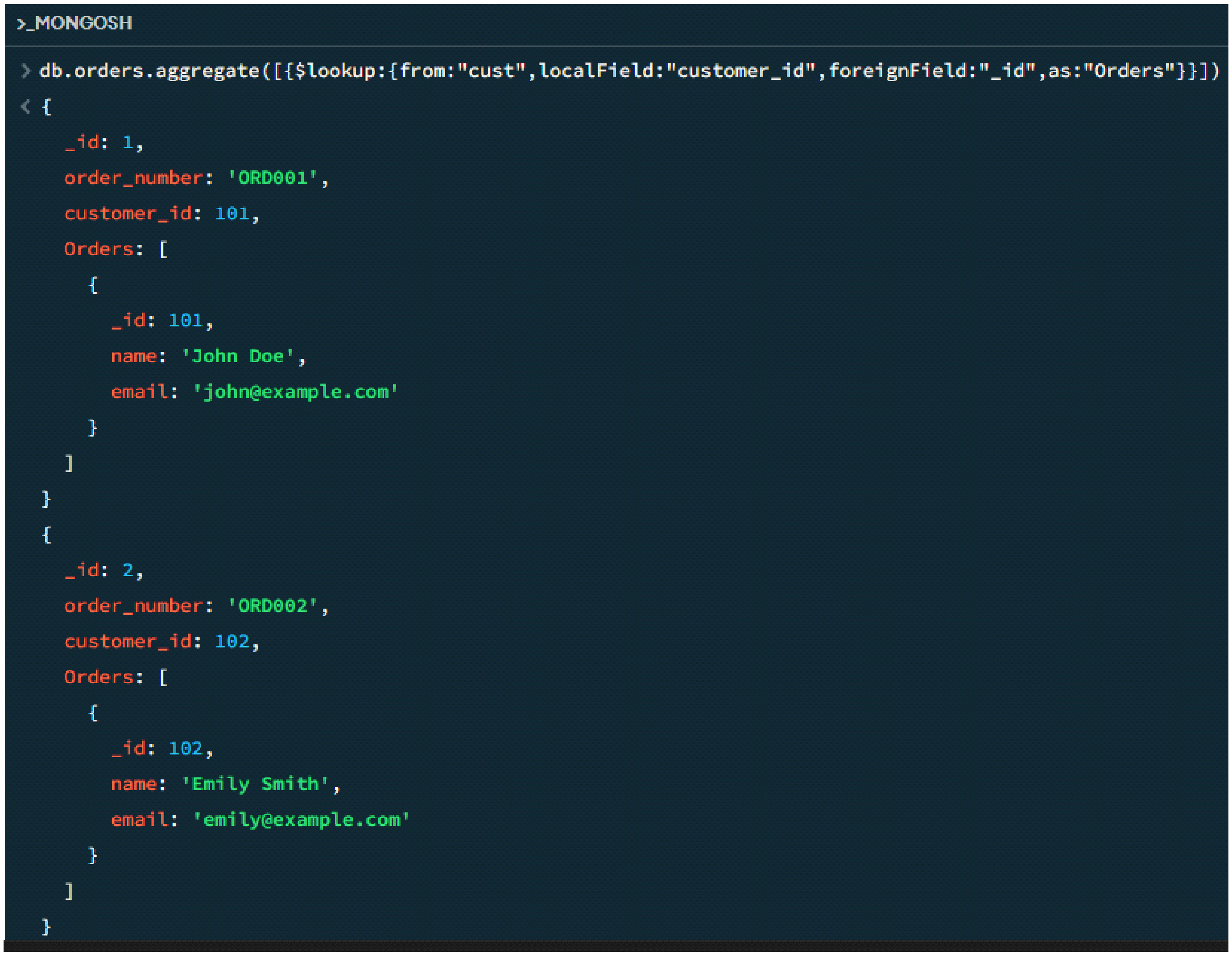
$lookup: the $lookup is an aggregate pipeline stage that allow you to perform a left outer join between two collection.

Left Outer Join :



```
>_MONGOSH
> db.cust.aggregate([{$lookup:{from:"orders",localField:"_id",foreignField:"customer_id",as:"Orders"}}])
< {
    _id: 101,
    name: 'John Doe',
    email: 'john@example.com',
    Orders: [
      {
        _id: 1,
        order_number: 'ORD001',
        customer_id: 101
      }
    ]
  }
  {
    _id: 102,
    name: 'Emily Smith',
    email: 'emily@example.com',
    Orders: [
      {
        _id: 2,
        order_number: 'ORD002',
        customer_id: 102
      }
    ]
  }
```

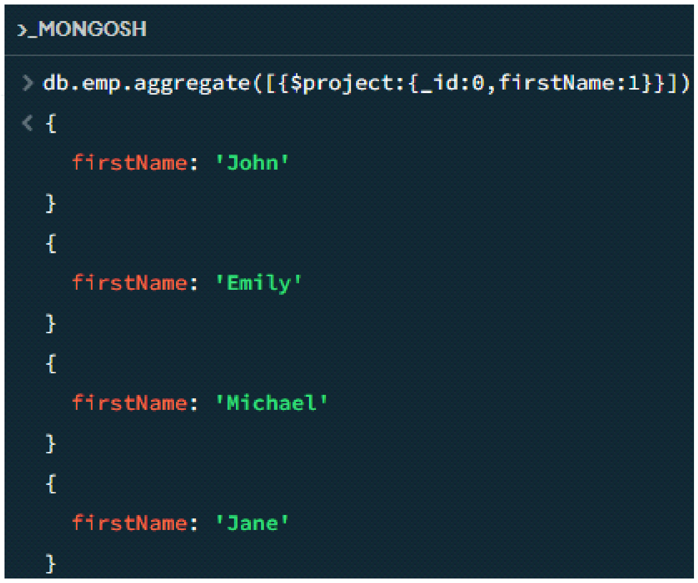
Right Outer Join:



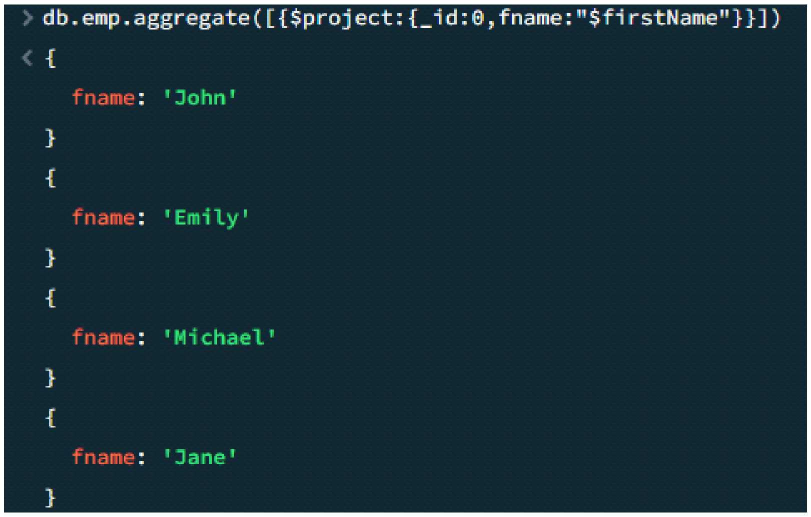
```
>_MONGOSH
> db.orders.aggregate([{$lookup:{from:"cust",localField:"customer_id",foreignField:"_id",as:"Orders"}}])
< {
    _id: 1,
    order_number: 'ORD001',
    customer_id: 101,
    Orders: [
      {
        _id: 101,
        name: 'John Doe',
        email: 'john@example.com'
      }
    ]
  }
  {
    _id: 2,
    order_number: 'ORD002',
    customer_id: 102,
    Orders: [
      {
        _id: 102,
        name: 'Emily Smith',
        email: 'emily@example.com'
      }
    ]
  }
```

$project:

$project stage is used in the aggregation pipeline to reshape documents, include or exclude fields, and create compound fields.



```
>_MONGOSH
> db.emp.aggregate([{$project:{_id:0,firstName:1}}])
< {
    firstName: 'John'
  }
  {
    firstName: 'Emily'
  }
  {
    firstName: 'Michael'
  }
  {
    firstName: 'Jane'
  }
```



```
> db.emp.aggregate([{$project:{_id:0,fname:"$firstName"}}])
< {
    fname: 'John'
  }
  {
    fname: 'Emily'
  }
  {
    fname: 'Michael'
  }
  {
    fname: 'Jane'
  }
```

- Capped collection :

A **capped collection** in MongoDB is a special type of collection that has a fixed size and maintains the order of insertion. It works like a circular queue, meaning that when the specified size limit is reached, the oldest documents are automatically overwritten by the newest ones. This type of collection is useful for scenarios where you need to store logs, cache, or any other use case where only the most recent data is relevant.

```
db.createCollection("myCappedCollection", {
  capped: true,
  size: 5242880,  // Maximum size in bytes (e.g., 5 MB)
  max: 5000       // (Optional) Maximum number of documents
})
```

# Replication & Sharding

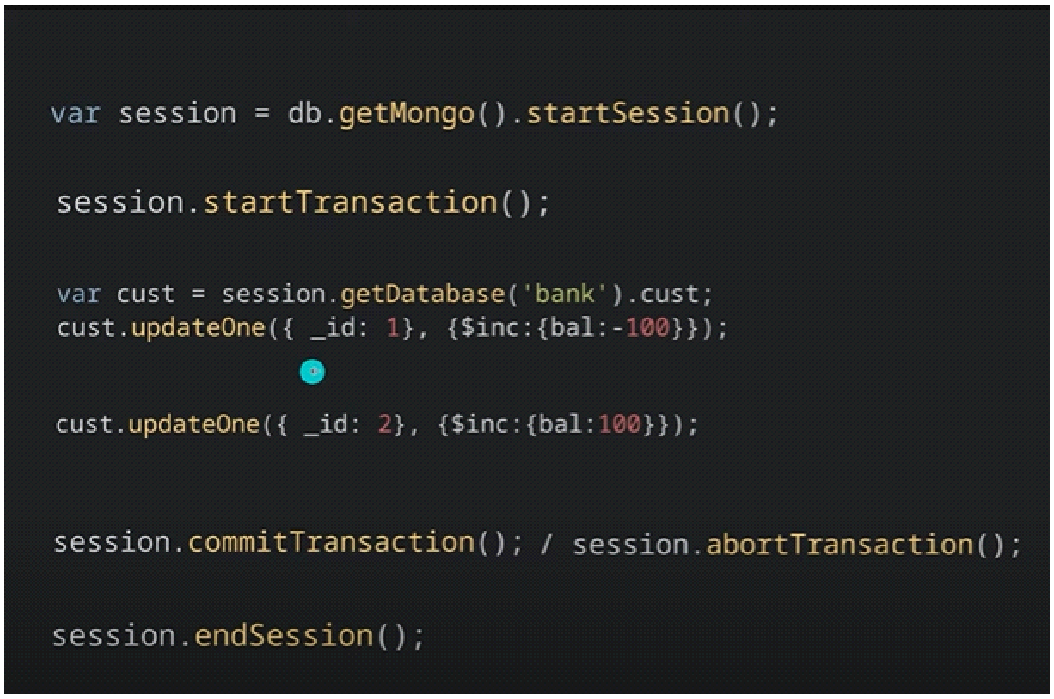
Thursday, January 9, 2025 11:05 AM

Replication :
MongoDB replication is a process that creates multiple copies of data across multiple servers or nodes to improve data availability, fault tolerance, and scalability.

Sharding :
Sharding is a method in MongoDB that distributes data across multiple machines to support large data sets and high-performance operations.

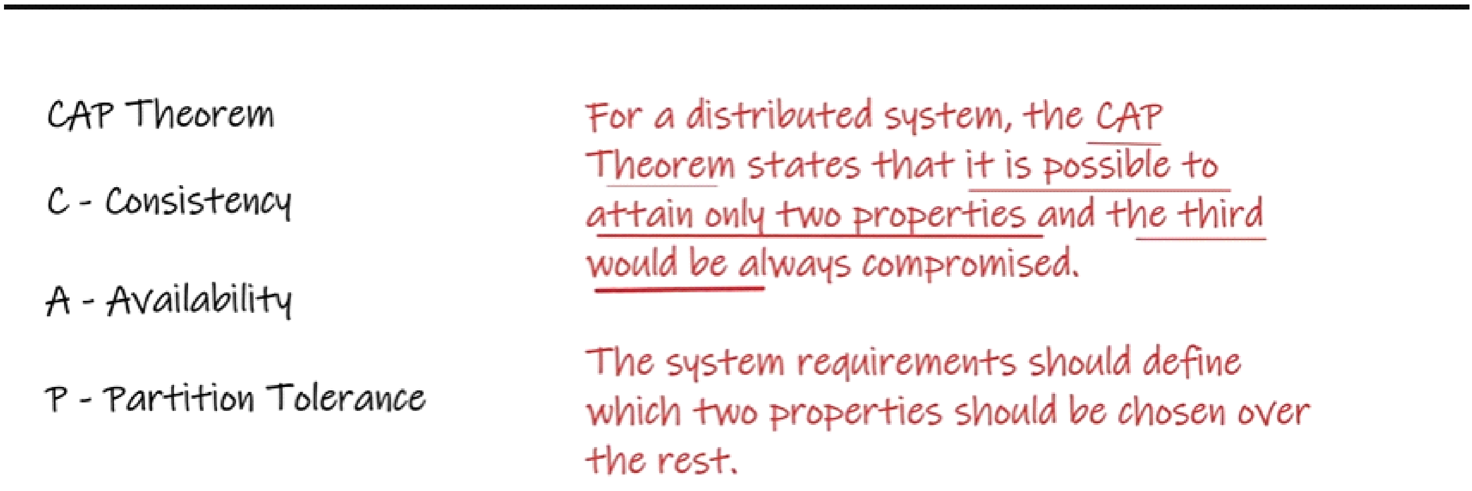
What is transaction?

- A transaction is a set of operations executed as a single, atomic unit
- Transaction provide data consistency by ensuring that either all the operation within the transaction are committed to the database, or none of them are.
- Transaction are designed to provide ACID Property.



```
var session = db.getMongo().startSession();

session.startTransaction();

var cust = session.getDatabase('bank').cust;
cust.updateOne({ _id: 1}, {$inc:{bal:-100}});

cust.updateOne({ _id: 2}, {$inc:{bal:100}});

session.commitTransaction(); / session.abortTransaction();

session.endSession();
```

CAP thoerem :



CAP Theorem

C - Consistency

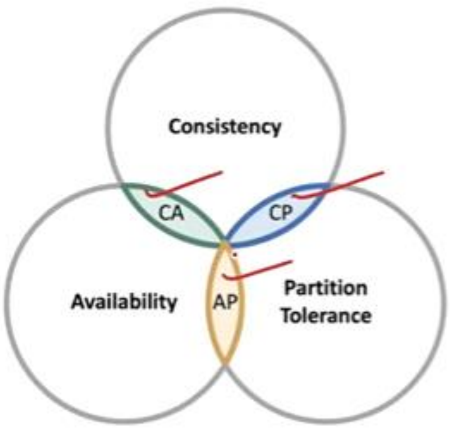
A - Availability

P - Partition Tolerance

For a distributed system, the CAP Theorem states that it is possible to attain only two properties and the third would be always compromised.

The system requirements should define which two properties should be chosen over the rest.

- The system designer can select Consistency and Partition Tolerance but the availability would be compromised then.
- The system designer can select Partition Tolerance and Availability but the consistency would be compromised then.
- The system designer can select Availability and Consistency but the Partition Tolerance would be compromised then.


Consistency
CA
CP
Availability
AP
Partition Tolerance