Creating a Database Using MongoDB and Mongoosh

Creating and Inserting Data in the Database in the VS Code Terminal

- **db.dbName.insertOne({})** – can only add a single data to the database
- **db.dbName.find()** - to get all the data in the particular db

1. school> db.employee.insertOne({name:"Ranjitha", role:"Programmer", salary:38000}) **// it adds a new data with the existing data**

```
{                                                                    of the employee db it creates a
                                                    new db employee.
  acknowledged: true,
  insertedId: ObjectId('65f34cdcde7d5f1cdd018cc3')
}
school> db.employee.find()                                    // gets all the data in the employee db
[
  {
    _id: ObjectId('65f34cdcde7d5f1cdd018cc3'),
    name: 'Ranjitha',
    role: 'Programmer',
    salary: 38000
  }
]
```

2. school> **db.students.insertOne([{name:"Krish", class:7, section:"D"}, {name:"Raghu", class:9, section:"C"}])**

```
{                                                                    // even if you add multiple data it stores it
as a single data
  acknowledged: true,
  insertedId: ObjectId('65f34bafde7d5f1cdd018cc1')
}
school> db.students.find()
[
```

Creating a Database Using MongoDB and Mongoosh

```
  {
    '0': { name: 'Krish', class: 7, section: 'D' },
    '1': { name: 'Raghu', class: 9, section: 'C' },
    _id: ObjectId('65f34bafde7d5f1cdd018cc1')
  }
]
```

3. school> **db.employee.insertMany([{name:"Sandhya", role:"Associate Analyst", salary:18000},{name:"Sri", role:"Telecaller", salary:12000},{name:"Jyoshna", role:"Sr.executive", salary:35000}])**

```
{                                                    // Multiple data
  acknowledged: true,
  insertedIds: {
    '0': ObjectId('65f350e3de7d5f1cdd018cc4'),
    '1': ObjectId('65f350e3de7d5f1cdd018cc5'),
    '2': ObjectId('65f350e3de7d5f1cdd018cc6')
  }
}
school> db.employee.find()
[
  {
    _id: ObjectId('65f34cdcde7d5f1cdd018cc3'),                    // previous data
    name: 'Ranjitha',
    role: 'Programmer',
    salary: 38000
  },
  {
    _id: ObjectId('65f350e3de7d5f1cdd018cc4'),
    name: 'Sandhya',
    role: 'Associate Analyst',
    salary: 18000
```

Creating a Database Using MongoDB and Mongoosh

```
  },
  {
    _id: ObjectId('65f350e3de7d5f1cdd018cc5'),
    name: 'Sri',
    role: 'Telecaller',
    salary: 12000
  },
  {
    _id: ObjectId('65f350e3de7d5f1cdd018cc6'),
    name: 'Jyoshna',
    role: 'Sr.executive',
    salary: 35000
  }
]
```

Sorting

1. A-Z order (alphabetical order)

```
school> db.students.find().sort({name:1})
[
  {
    _id: ObjectId('65f345e5b13bff0899ed3d79'),
    name: 'Govind',
    class: 10,
    section: 'A'
  },
  {
    _id: ObjectId('65f35505b13bff0899ed3d89'),
    name: 'Joy',
    class: 11,
    section: 'B'
```

```
  },
  {
    _id: ObjectId('65f34667b13bff0899ed3d7a'),
    name: 'Sai',
    class: 7,
    section: 'B'
  }
]
```

2. Z-A order (reverse-alphabetical order)

```
school> db.students.find().sort({name:-1})
[
  {
    _id: ObjectId('65f34667b13bff0899ed3d7a'),
    name: 'Sai',
    class: 7,
    section: 'B'
  },
  {
    _id: ObjectId('65f35505b13bff0899ed3d89'),
    name: 'Joy',
    class: 11,
    section: 'B'
  },
  {
    _id: ObjectId('65f345e5b13bff0899ed3d79'),
    name: 'Govind',
    class: 10,
    section: 'A'
```

```
  }
]
```

3. 1,2,3….. (numerical order)

```
school> db.students.find().sort({class:1})
[
  {
    _id: ObjectId('65f34667b13bff0899ed3d7a'),
    name: 'Sai',
    class: 7,
    section: 'B'
  },
  {
    _id: ObjectId('65f345e5b13bff0899ed3d79'),
    name: 'Govind',
    class: 10,
    section: 'A'
  },
  {
    _id: ObjectId('65f35505b13bff0899ed3d89'),
    name: 'Joy',
    class: 11,
    section: 'B'
  }
]
```

4. 100,99,98…. (reverse- numerical order)

```
school> db.students.find().sort({class:-1})
[
  {
```

```
  _id: ObjectId('65f35505b13bff0899ed3d89'),
  name: 'Joy',
  class: 11,
  section: 'B'
 },
 {
  _id: ObjectId('65f345e5b13bff0899ed3d79'),
  name: 'Govind',
  class: 10,
  section: 'A'
 },
 {
  _id: ObjectId('65f34667b13bff0899ed3d7a'),
  name: 'Sai',
  class: 7,
  section: 'B'
 }
]
```

Limiting

- Returns the limited number of data only according to the default id's.

```
school> db.students.find().limit(2)                    // returns only the limited number of data
[
 {
  _id: ObjectId('65f345e5b13bff0899ed3d79'),
  name: 'Govind',
  class: 10,
  section: 'A'
 },
 {
```

Creating a Database Using MongoDB and Mongoosh

```
  _id: ObjectId('65f34667b13bff0899ed3d7a'),
  name: 'Sai',
  class: 7,
  section: 'B'
 }
]
```

Using both sorting and limiting

- Returns the data by both sorting and also a limited data as per the limit given.

```
school> db.students.find().sort({name:1}).limit(2)
[
 {
  _id: ObjectId('65f345e5b13bff0899ed3d79'),
  name: 'Govind',
  class: 10,
  section: 'A'
 },
 {
  _id: ObjectId('65f35505b13bff0899ed3d89'),
  name: 'Joy',
  class: 11,
  section: 'B'
 }
]
```

How to use sort( ) in MongoDB Compass

- Under Documents, go to filter search bar
- Now enter the query/queries(filter)
- If you want to sort the data by applying some sorta filters, then go to options in the corner select and enter your filters to sort the data

Creating a Database Using MongoDB and Mongoosh

find( ) – this find( ) method consists of two parameters in it which is not mandatory in all the cases.

- .find( ) – this method is used to get all the updated data
- .find({query}) – this is used when you want a particular person's details.

1. **Single query**-

```
school> db.students.find({name:"Sai"})
[
  {
    _id: ObjectId('65f34667b13bff0899ed3d7a'),
    name: 'Sai',
    class: 7,
    section: 'B'
  }
]
```

2. **Multiple queries-**

```
school> db.students.find({section:"B",class:11})
[
  {
    _id: ObjectId('65f35505b13bff0899ed3d89'),
    name: 'Joy',
    class: 11,
    section: 'B'
  }
]
```

- .find({query}, {projection}) – this is used when I need a list of data with particular details to satisfy.

In this if you don't have a query you can simply { } leave the query space blank with enclosed braces.

1. **Single filter Projection-**

```
school> db.students.find({}, {name:true})
[
  { _id: ObjectId('65f345e5b13bff0899ed3d79'), name: 'Govind' },
  { _id: ObjectId('65f34667b13bff0899ed3d7a'), name: 'Sai' },
  { _id: ObjectId('65f35505b13bff0899ed3d89'), name: 'Joy' }
]
```

1. **Multiple filter projection-**

```
school> db.students.find({}, {name:true, _id:false})
[ { name: 'Govind' }, { name: 'Sai' }, { name: 'Joy' } ]
school> db.employee.find({}, {name:true, _id:false, salary:true})
[
```

Creating a Database Using MongoDB and Mongoosh

```
  { name: 'Ranjitha', salary: 38000 },
  { name: 'Sandhya', salary: 18000 },
  { name: 'Sri', salary: 12000 },
  { name: 'Jyoshna', salary: 35000 }
]
```

How to use find( ) in MongoDB Compass

- Under Documents, go to filter search bar
- Now enter the query/queries(filter)
- If you want to project the data by applying some sorta filters, then go to options in the corner select and enter your filters to project the data

Update( ) - it is used to modify or update a particular data.

- We modify the data using **$set** function. It takes 2 parameters **updateOne/Many({filter}, {$set:{update}})**.
- **$set:** sets the existing field , or if there's no field it creates a new one.

1. updateOne – to update a single person data.
2. updateMany – to update multiple person data.

- Multiple people can have the same name, so it is better to update using **_id** as it is unique.

1. **updateOne( )-**

```
school> db.employee.updateOne({name:"Jyoshna"}, {$set:{salary:40000}})
{
  acknowledged: true,
  insertedId: null,
  matchedCount: 1,
  modifiedCount: 1,
  upsertedCount: 0
}
school> db.employee.find({name:"Jyoshna"})
[
  {
    _id: ObjectId('65f350e3de7d5f1cdd018cc6'),
    name: 'Jyoshna',
    role: 'Sr.executive',
    salary: 40000
  }
]
```

2. **updateMany( )-**

Creating a Database Using MongoDB and Mongoosh

```
school> db.employee.updateMany({}, {$set:{FullTime:true}})
{
  acknowledged: true,
  insertedId: null,
  matchedCount: 3,
  modifiedCount: 3,
  upsertedCount: 0
}
school> db.employee.find()
[
  {
    _id: ObjectId('65f350e3de7d5f1cdd018cc4'),
    name: 'Sandhya',
    role: 'Associate Analyst',
    salary: 18000,
    FullTime: true
  },
  {
    _id: ObjectId('65f350e3de7d5f1cdd018cc5'),
    name: 'Sri',
    role: 'Telecaller',
    salary: 12000,
    FullTime: true
  },
  {
    _id: ObjectId('65f350e3de7d5f1cdd018cc6'),
    name: 'Jyoshna',
    role: 'Sr.executive',
    salary: 40000,
    FullTime: true
  }
]
```

To remove a particular field

- **$unset-** it is used to remove a field

```
school> db.employee.updateOne({name:"Ranjitha"}, {$unset:{salary:""}})
{
  acknowledged: true,
  insertedId: null,
  matchedCount: 1,
```

Creating a Database Using MongoDB and Mongoosh

```
  modifiedCount: 1,
  upsertedCount: 0
}
school> db.employee.find({name:"Ranjitha"})                    // removed salary field for ranjitha
[
  {
    _id: ObjectId('65f34cdcde7d5f1cdd018cc3'),
    name: 'Ranjitha',
    role: 'Programmer'
  }
]
```

exists( )- checks if it exists or not and then apply the filter (used mostly to update or delete)

```
school> db.students.updateMany({fee:{$exists:false}}, {$set:{fee:"Paid"}})                    // checks if fee: field exists or not
{
  acknowledged: true,
  insertedId: null,
  matchedCount: 3,
  modifiedCount: 3,
  upsertedCount: 0
}
school> db.students.find()
[
  {
    _id: ObjectId('65f345e5b13bff0899ed3d79'),
    name: 'Govind',
    class: 10,
    section: 'A',
    fee: 'Paid'
  },
  {
    _id: ObjectId('65f34667b13bff0899ed3d7a'),
    name: 'Sai',
    class: 7,
    section: 'B',
    fee: 'Paid'
  },
  {
```

```
  _id: ObjectId('65f35505b13bff0899ed3d89'),
  name: 'Joy',
  class: 11,
  section: 'B',
  fee: 'Paid'
 }
]
```

How to update( ) in MongoDB Compass

- Select the person's data, then various icons appear on the right top
- Pencil icon – to edit the filed
- Delete icon – to delete the field
- + icon - to add the field

delete( ) -

1. deleteOne( )-

```
school> db.employee.deleteOne({name: "Ranjitha"})
{ acknowledged: true, deletedCount: 1 }
school> db.employee.find()
[
 {
  _id: ObjectId('65f350e3de7d5f1cdd018cc4'),
  name: 'Sandhya',
  role: 'Associate Analyst',
  salary: 18000,
  FullTime: true
 },
 {
  _id: ObjectId('65f350e3de7d5f1cdd018cc5'),
  name: 'Sri',
  role: 'Telecaller',
  salary: 12000,
  FullTime: true
 },
 {
  _id: ObjectId('65f350e3de7d5f1cdd018cc6'),
  name: 'Jyoshna',
  role: 'Sr.executive',
  salary: 40000,
  FullTime: true
```

Creating a Database Using MongoDB and Mongoosh

```
 }
]
```

2. **deleteMany( )-**

school> **db.employee.deleteMany({FullTime:true})**

{ acknowledged: true, deletedCount: 3 }

school> **db.employee.find()** // deleted every data because everyone is a fulltime employee

Comparision Operators - used to compare

1. **$ne:** - not equals
2. **$lt:** - less than
3. **$lte** - less than equals
4. **$gt:** - greater than
5. **$gte** - greater than equals
6. **$in:** - in an array (list) - checks the whole list items and returns the result which is in the list
7. **$nin:** - not in an array (list) - checks the whole list items and returns the result which is not in the list

i. not equals

```
school> db.students.find({name:{$ne:"Joy"}})
```
// compares the name with joy and returns the remaining as we used **$ne:(**not equals**)**

```
[
  {
    _id: ObjectId('65f345e5b13bff0899ed3d79'),
    name: 'Govind',
    class: 10,
    section: 'A',
    fee: 'Paid'
  },
  {
    _id: ObjectId('65f34667b13bff0899ed3d7a'),
    name: 'Sai',
    class: 7,
    section: 'B',
    fee: 'Paid'
  }
]
```

ii. less than

```
school> db.students.find({age:{$lt:15}})
```

Creating a Database Using MongoDB and Mongoosh

```
[
  {
    _id: ObjectId('65f34667b13bff0899ed3d7a'),
    name: 'Sai',
    class: 7,
    section: 'B',
    fee: 'Paid',
    age: 12
  }
]
```

a) less than equals

```
school> db.students.find({age:{$lte:15}})
[
  {
    _id: ObjectId('65f345e5b13bff0899ed3d79'),
    name: 'Govind',
    class: 10,
    section: 'A',
    fee: 'Paid',
    age: 15
  },
  {
    _id: ObjectId('65f34667b13bff0899ed3d7a'),
    name: 'Sai',
    class: 7,
    section: 'B',
    fee: 'Paid',
    age: 12
  }
]
```

iii. greater than

```
school> db.students.find({age:{$gt:15}})
[
  {
    _id: ObjectId('65f35505b13bff0899ed3d89'),
    name: 'Joy',
    class: 11,
```

Creating a Database Using MongoDB and Mongoosh

```
  section: 'B',
  fee: 'Paid',
  age: 16
 }
]
```

a) greater than equals

```
school> db.students.find({age:{$gte:15}})
[
 {
  _id: ObjectId('65f345e5b13bff0899ed3d79'),
  name: 'Govind',
  class: 10,
  section: 'A',
  fee: 'Paid',
  age: 15
 },
 {
  _id: ObjectId('65f35505b13bff0899ed3d89'),
  name: 'Joy',
  class: 11,
  section: 'B',
  fee: 'Paid',
  age: 16
 }
]
```

iv. $in: - in an array - checks the whole list items and returns the result

```
school> db.students.find({name:{$in:["Govind", "Jaya", "Sai"]}})
[
 {
  _id: ObjectId('65f345e5b13bff0899ed3d79'),
  name: 'Govind',
  class: 10,
  section: 'A',
  fee: 'Paid',
  age: 15
 },
 {
```

Creating a Database Using MongoDB and Mongoosh

```
    _id: ObjectId('65f34667b13bff0899ed3d7a'),
    name: 'Sai',
    class: 7,
    section: 'B',
    fee: 'Paid',
    age: 12
  }
]
```

v. $nin

```
school> db.students.find({name:{$nin:["Govind", "Jaya", "Sai"]}})
[
  {
    _id: ObjectId('65f35505b13bff0899ed3d89'),
    name: 'Joy',
    class: 11,
    section: 'B',
    fee: 'Paid',
    age: 16
  }
]
```

Using two comparisons

```
school> db.students.find({class:{$gte:10, $lt:11}})
[
  {
    _id: ObjectId('65f345e5b13bff0899ed3d79'),
    name: 'Govind',
    class: 10,
    section: 'A',
    fee: 'Paid',
    age: 15
  }
]
```

Creating a Database Using MongoDB and Mongoosh

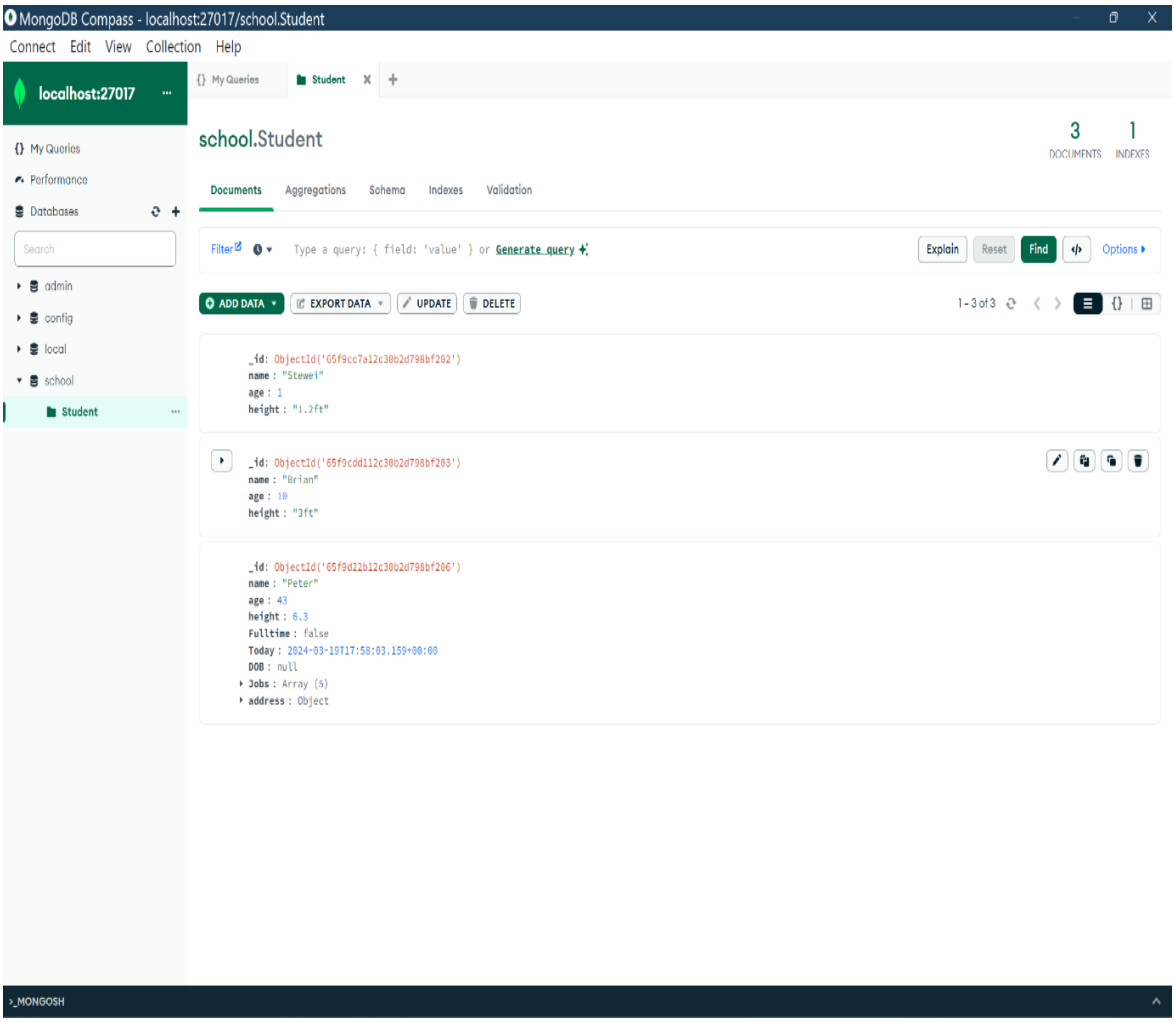
☐ Deletion Operation:

The MongoDB shell provides the following methods to delete documents from a collection:

☐ To delete multiple documents, use `db.collection.deleteMany()`.

☐ To delete a single document, use `db.collection.deleteOne()`.

☐ Delete a user document based on a specific name.



Creating a Database Using MongoDB and Mongoosh

Index Creation:

Indexes support efficient execution of queries in MongoDB. Without indexes, MongoDB must scan every document in a collection to return query results. If an appropriate index exists for a query, MongoDB