

MongoDB

MongoDB is a modern, open-source database handling data in flexible JSON like documents. Its NOSQL approach allows for easy scalability and adaptation to changing data Structures.

MongoDB Compass

MongoDB compass is a GUI based tools to interact with MongoDB servers.

It allows users to visually explore data and performance CRUD operations.

Supports installation on Linux, Mac and Windows.

Query performance optimization is one of its features.

MongoDB compass connection guide

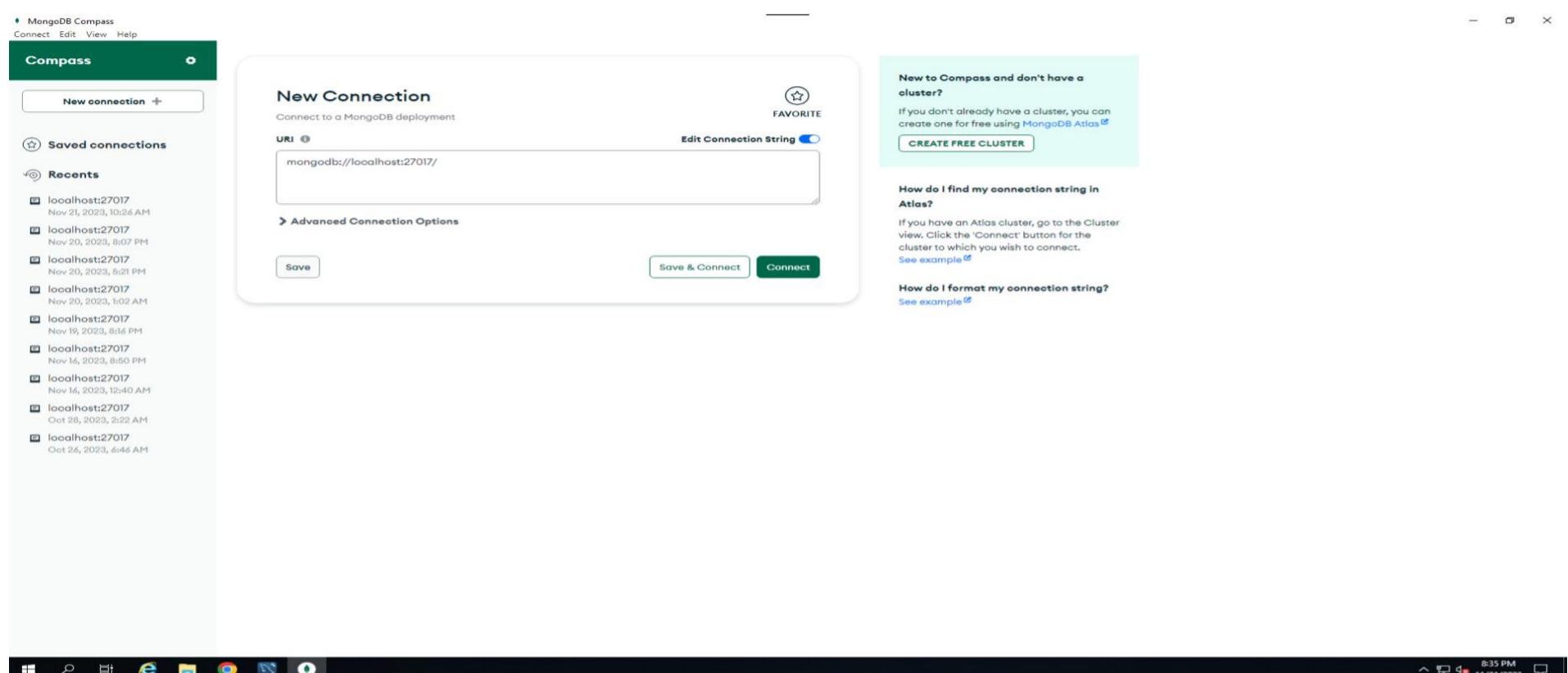


Fig:1

Open MongoDB compass:

Launch MongoDB compass on your computer.

Connect to server:

Click on “connect” to open the connection dialog as shown in **fig1**.

MongoDB compass connection success:

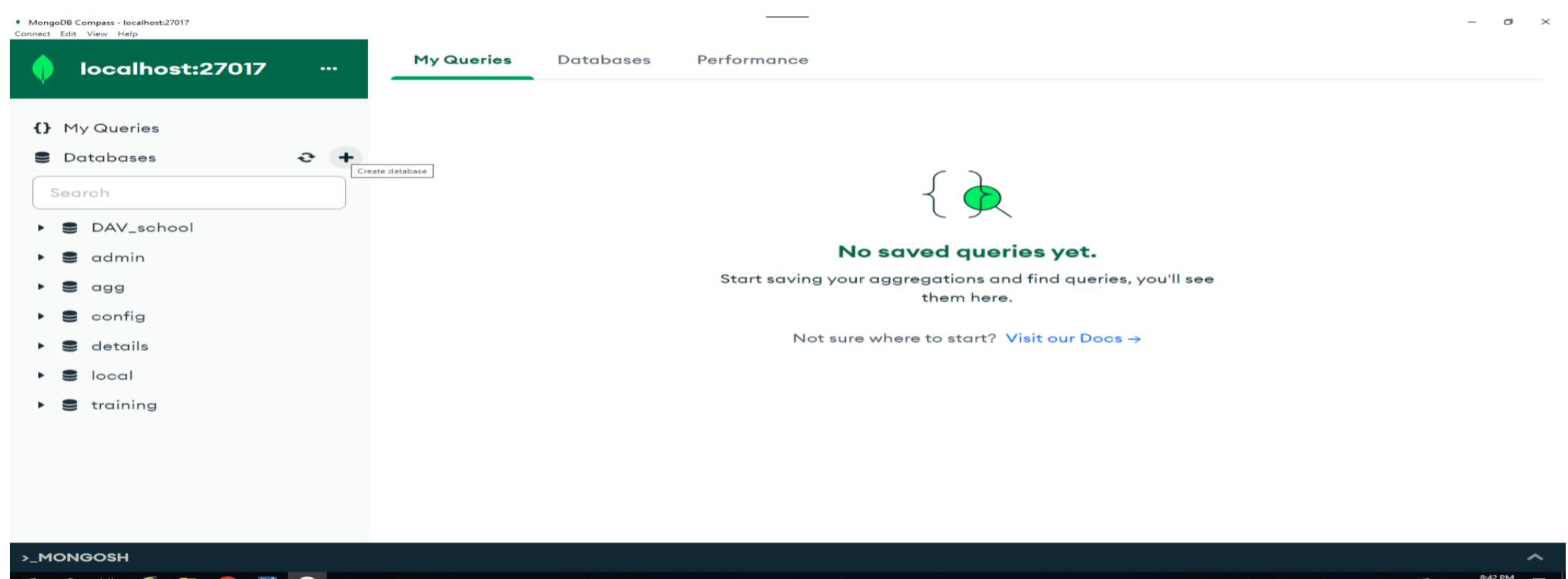


Fig:2

After successful connection to MongoDB compass directed to the welcoming page as shown in **fig2**.

Navigate effortlessly through databases, collections and indexes using menu on the left. Enhance overall visualization of your MongoDB environment.

Database creation in MongoDB Compass:

Click on “+” symbol present on left menu to create Database. After clicking a new page (fig3) opens, guiding you through database creation process

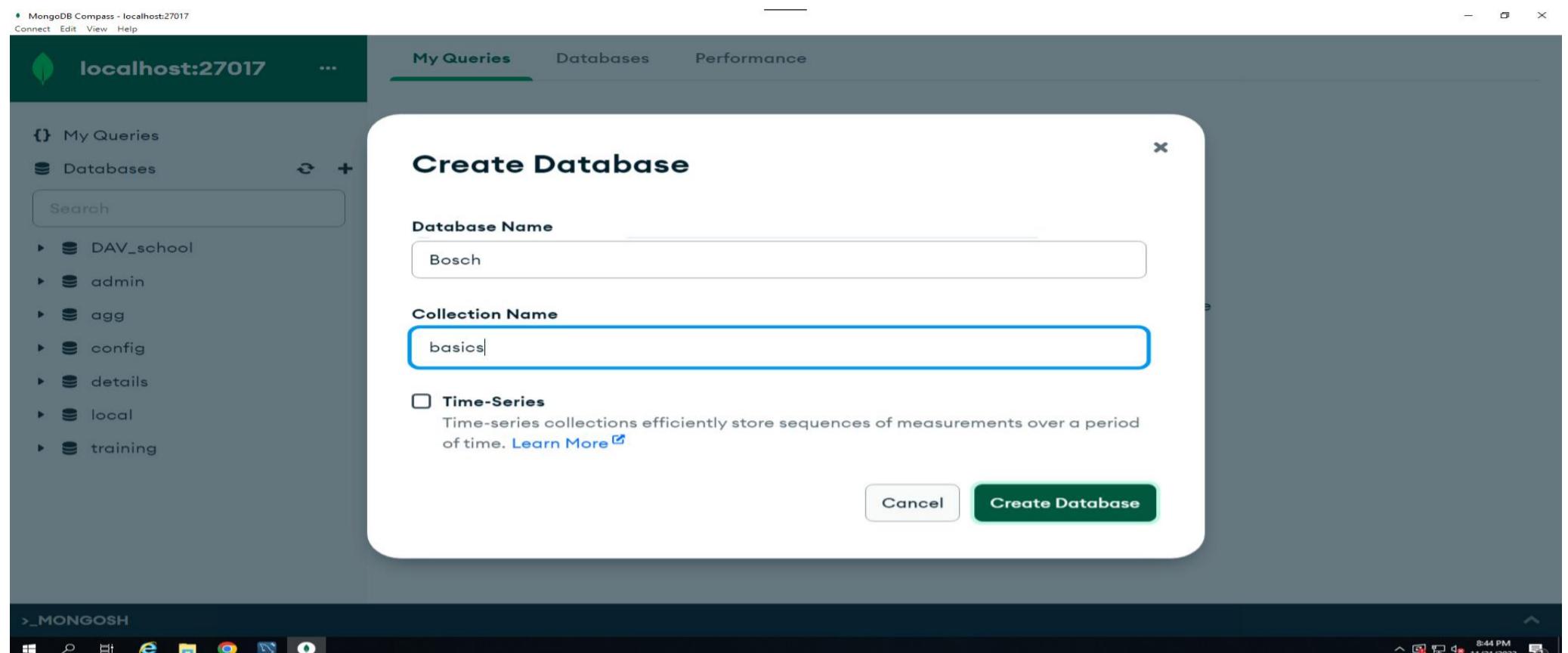


Fig:3

Enter Database name “Bosch” and Collection name “basics”.

After entering required information enter “Create Database”.

Database is created in MongoDB compass with database “Bosch” and collection “basics”.

Database creation in MongoDB SHELL :

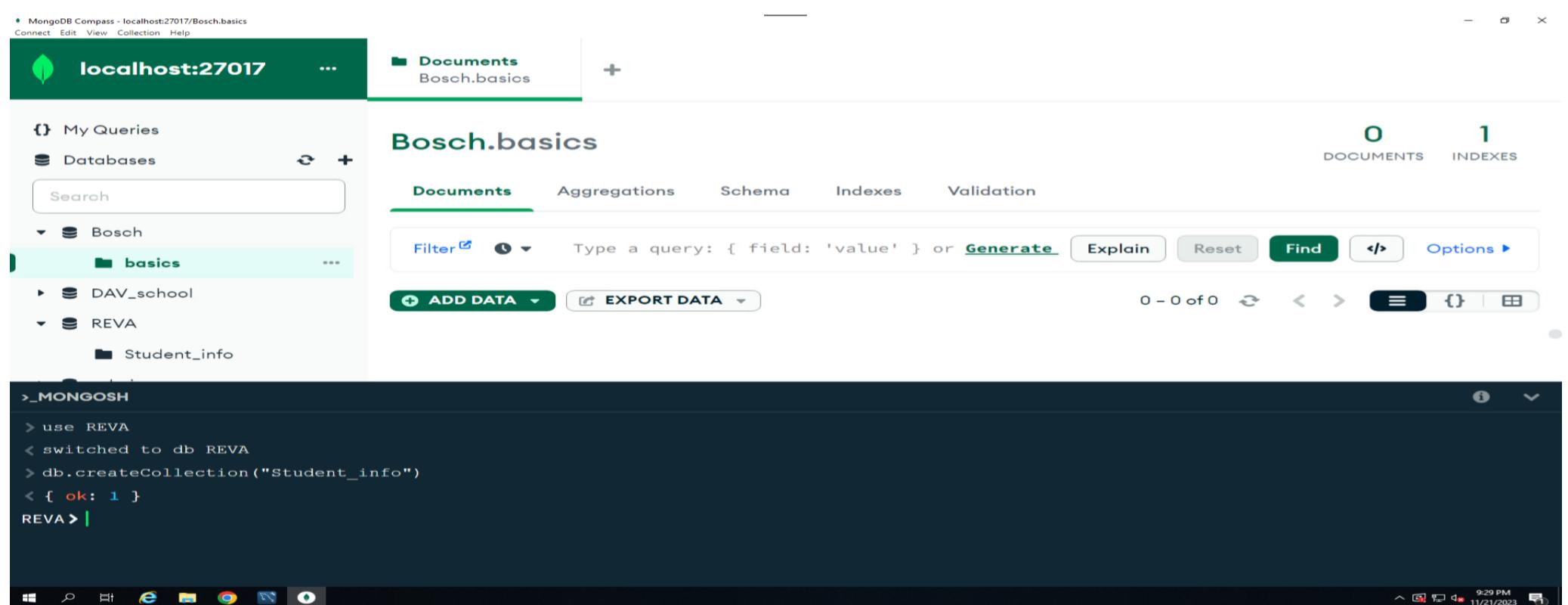


Fig:4

Syntax:

Use <database_name>: create database or open created database.

db.createcollection("collection_name"). : create collection

ex:

use REVA //database REVA created.

db.createcollection("Student_info") //collection Student_info is created.

Add collection in MongoDB compass:

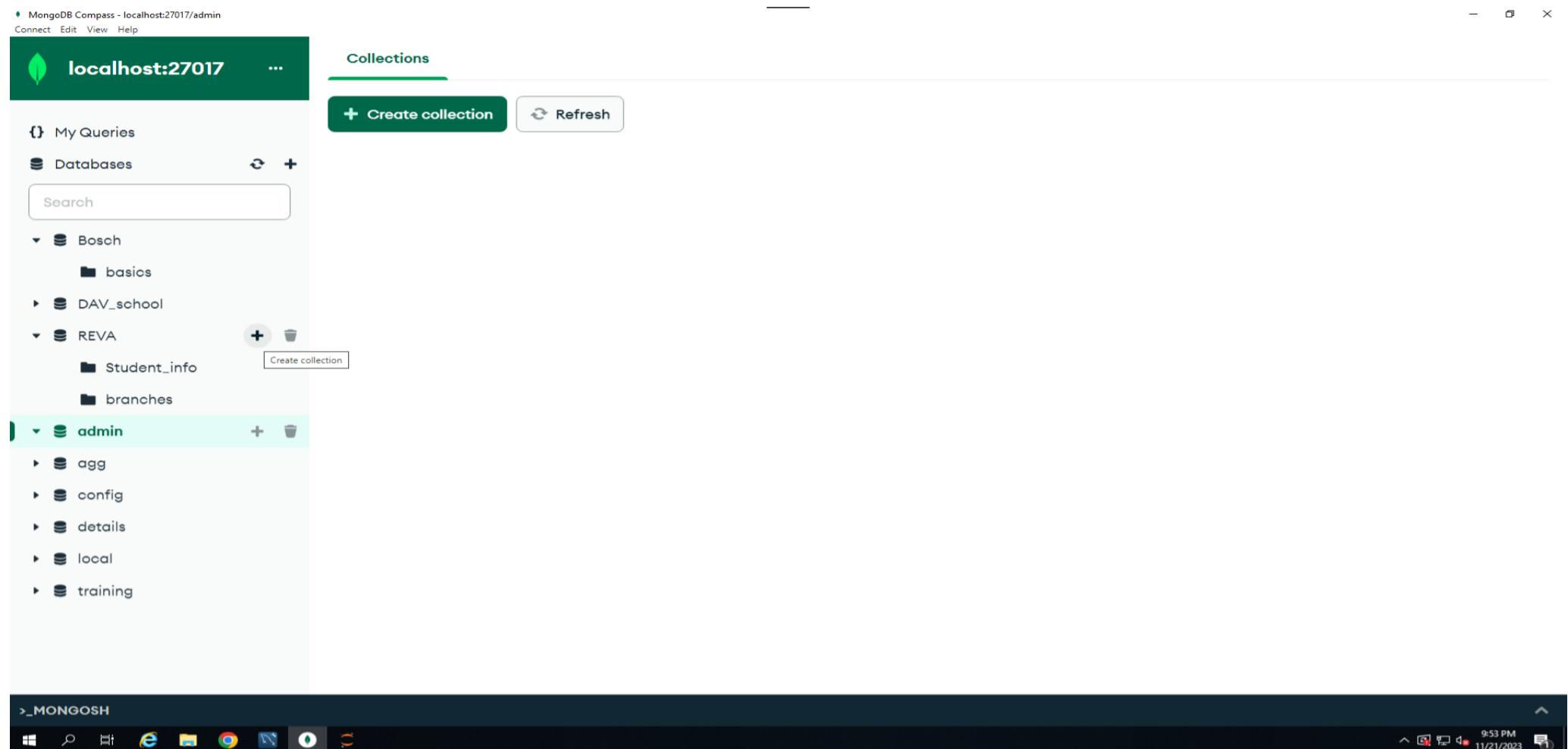


Fig:5

Click on “+” symbol present on left menu to create new collection as shown in fig 5.

Note: Database consists of N-number of collections. Collections consists of multiple documents.

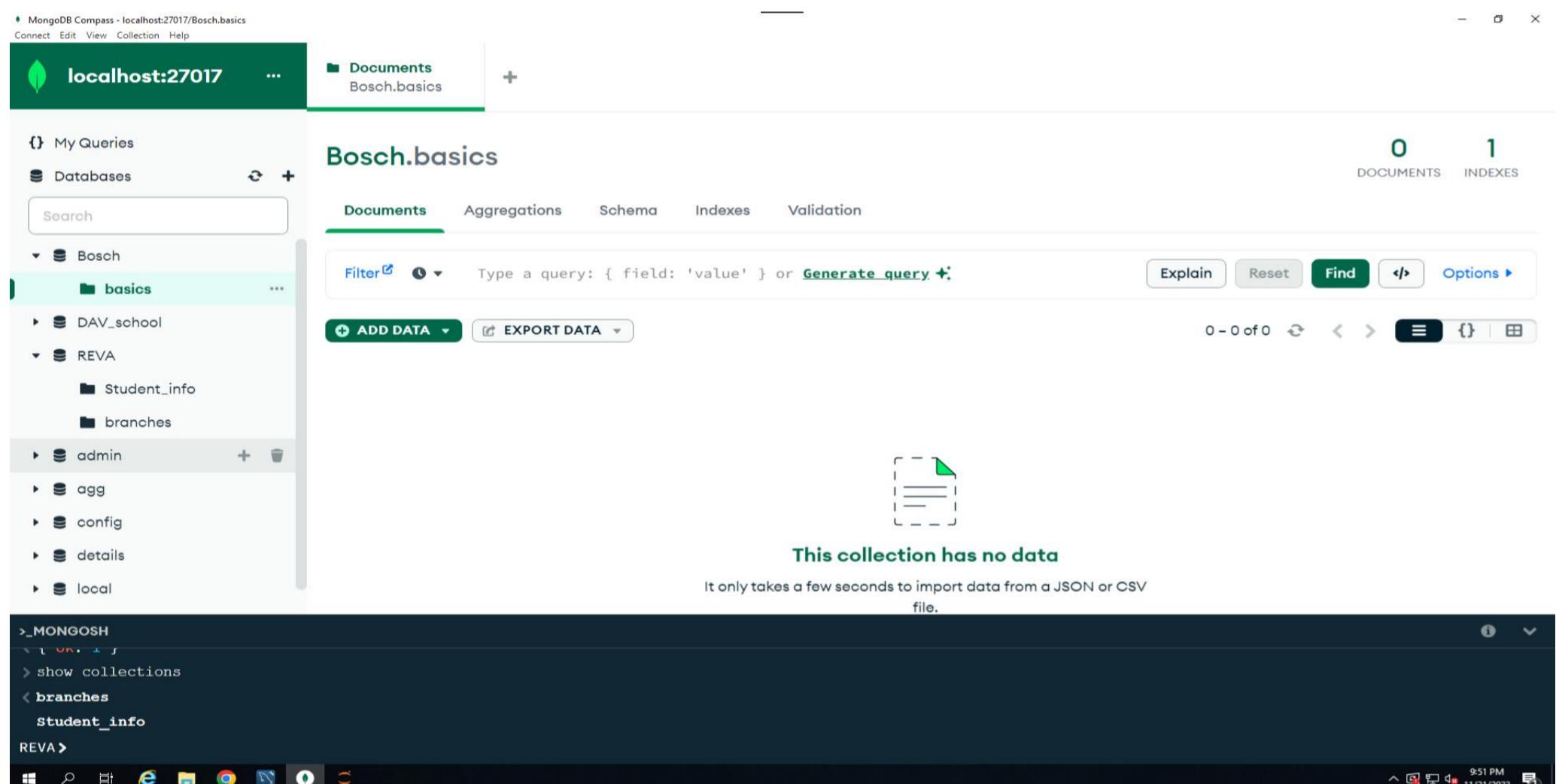


Fig:6

“show collections” used to know collections present in Database.

Step1 :use REVA

Step2: show collections

o/p :

branches

Student_info

Insert Document in collection MongoDB Compass :

The screenshot shows the MongoDB Compass interface. The left sidebar lists databases and collections, including 'REVA' and 'Student_info'. The main area displays the 'REVA.Student_info' collection with a message stating 'This collection has no data'. It includes buttons for 'Import JSON or CSV file' and 'Insert document'. A small icon of a document with a green checkmark is visible.

Fig:7

Follow steps :

Goto REVA <<Student_info<<ADD DATA<<insert document.

After clicking “insert document” a new page Insert Document page is open as shown in fig8.

The screenshot shows the MongoDB Compass interface with the 'Insert Document' dialog box open. The dialog box is titled 'Insert Document' and says 'To collection REVA.Student_info'. It contains a code editor with the following JSON document:

```
1 {  
2   "name": "nikki",  
3   "age": 21,  
4   "branch": "electronics and communication"  
5 }
```

Below the code editor are 'Cancel' and 'Insert' buttons. The background shows the REVA collection with 0 documents and 1 index.

Fig:8

Syntax :

```
{ "field_name1 ":"value_1", "field_name2 ":"value_2", ..... "fieldname_n ":"value_n"}  
{ "name":"nikki", "age":21,"branch":electronics and communication"}
```

After entering the document in collection (Student_info) click insert.

Insert multiple documents:

The screenshot shows the MongoDB Compass interface. On the left, the sidebar lists databases and collections, with 'REVA' and 'Student_info' selected. In the center, the 'Documents' tab is active, showing the 'REVA.Student_info' collection. A modal window titled 'Insert Document' is open, prompting the user to insert data into the 'REVA.Student_info' collection. The document being inserted is a JSON array containing two student records:

```
[{"name": "nikki", "age": 21, "branch": "electronics and communication"}, {"name": "raj", "age": 22, "branch": "electronics and communication"}]
```

At the bottom of the modal are 'Cancel' and 'Insert' buttons. The main interface shows '0 DOCUMENTS' and '1 INDEXES'.

Fig:9

Syntax :

```
[{"field_name1 ":"value_1"}, {"field_name1 ":"value_1"},..... "field_nameN ":"value_N"]
```

Ex :

```
[{"name":"nikki","age":21,"branch":"electronics and communication"}, {"name":"raj","age":22,"branch":"electronics and communication"}]
```

After entering the document in collection (Student_info) click insert.

Insert Document in ARRAY:

The screenshot shows the MongoDB Compass interface. The left sidebar shows 'REVA' and 'Student_info' selected. The 'Documents' tab is active, showing the 'REVA.Student_info' collection. A modal window titled 'Insert Document' is open, prompting the user to insert data into the 'REVA.Student_info' collection. The document being inserted is a JSON array containing two student records, each with a 'subjects' array field:

```
[{"name": "nikki", "age": 21, "branch": "electronics and communication", "subjects": ["embeddedC", "digital_electronics", "linux"]}, {"name": "raj", "age": 22, "branch": "electronics and communication", "subjects": ["embeddedC", "digital_electronics", "linux"]}]
```

At the bottom of the modal are 'Cancel' and 'Insert' buttons. The main interface shows '2 DOCUMENTS' and '1 INDEXES'. To the right, a table shows the inserted documents with their '_id' and 'subjects' arrays.

Fig:10

Example :

```
[{"name":"nikki","age":21,"branch":"electronics and communication","subjects":["embeddedC","digital_electronics","linux"]}, {"name":"raj","age":22,"branch":"electronics and communication","subjects":["embeddedC","digital_electronics","linux"]}]
```

O/P :

Viewing Document 3 types :

List view (Fig :13)

JSON view(Fig : 12)

Table view(Fig : 11)

Table view :

The screenshot shows the MongoDB Compass interface with the 'REVA.Student_info' collection selected. The 'Documents' tab is active. At the top, there are buttons for 'ADD DATA' and 'EXPORT DATA'. Below is a table with two rows of data:

	_id	name	age	branch	subjects
1	ObjectId('655da179a88b2e1c348...')	"nikki"	21	"electronics and communication"	[] 3 elements
2	ObjectId('655da179a88b2e1c348...')	"raj"	22	"electronics and communication"	[] 3 elements

At the bottom of the interface, there is a terminal window titled '>_MONGOSH' showing the command prompt.

Fig:11

JSON view :

The screenshot shows the MongoDB Compass interface with the 'REVA.Student_info' collection selected. The 'Documents' tab is active. The data is displayed as two JSON objects:

```
{ "_id": { }, "name": "nikki", "age": 21, "branch": "electronics and communication", "subjects": [ ] } { "_id": { }, "name": "raj", "age": 22, "branch": "electronics and communication", "subjects": [ ] }
```

At the bottom of the interface, there is a terminal window titled '>_MONGOSH' showing the command prompt.

Fig:12

List view

The screenshot shows the MongoDB Compass interface. On the left, the sidebar lists databases and collections, with 'REVA' and 'Student_info' selected. The main area displays the 'REVA.Student_info' collection. It shows two documents:

```
_id: ObjectId('655da179a88b2e1c34835615')
name: "nikki"
age: 21
branch: "electronics and communication"
subjects: Array (3)

_id: ObjectId('655da179a88b2e1c34835616')
name: "raj"
age: 22
branch: "electronics and communication"
subjects: Array (3)
```

At the bottom, a terminal window titled '_MONGOSH' shows the command: `> db.Student_info.insertOne({ "name": "nikki", "age": 21, "branch": "electronics and communication", "subjects": ["Math", "Physics", "Chemistry"] })`.

Fig:13

Insert document in mongosh shell :

Methods to insert documents into a collection :

`Insert()` :insert one or more documents into collection.

`InsertOne()` :insert single document into collection.

`InsertMany()`: insert multiple documents into collection.

`InsertOne()`:

The screenshot shows the MongoDB Compass interface. On the left, the sidebar lists databases and collections, with 'REVA' and 'branches' selected. The main area displays the 'REVA.branches' collection. It shows one document:

```
_id: ObjectId('655da6dd92c2b2e30b12a5f7')
branch_name: "Electronics"
director: "srinivas"
block_name: "CVR"
no_sec: "4"
```

At the bottom, a terminal window titled '_MONGOSH' shows the command: `> db.branches.insertOne({ "branch_name": "Electronics", "director": "srinivas", "block_name": "CVR", "no_sec": "4" })`.

Fig:14

Syntax:

`use<database_name>`

`db.collection_name.insertOne({“field_name”:”value”})`

ex:

`db.branches.insertOne({“branch_name”:”electronics”, “director”:”srinivas”, “block_name”:”CVR”, “no_sec”:4})`

O/P :

The screenshot shows the MongoDB Compass interface. On the left, the database structure is visible with a tree view. Under the 'REVA' database, there is a 'branches' collection highlighted. The main area displays the 'Documents' tab with a table showing one document. The document has the following fields:

_id	branch_name	director	block_name	no_sec
ObjectId('655da6dd92c2b2e30b1...')	"Electronics"	"srinivas"	"CVR"	"4"

Below the table, the MONGOSH shell shows the command used to insert the document:

```
> db.branches.insertOne({"branch_name": "Electronics", "director": "srinivas", "block_name": "CVR", "no_sec": "4"})
```

The status of the insertion is shown as:

```
< { acknowledged: true, insertedId: ObjectId("655da6dd92c2b2e30b12a5f7") }
```

Fig:15

insertMany():

The screenshot shows the MongoDB Compass interface with the same database structure as Fig:15. The 'branches' collection is still highlighted. The main area now shows two documents listed under the 'Documents' tab. The first document is identical to the one in Fig:15. The second document is:

_id	branch_name	director	block_name	no_sec
ObjectId('655dc4ce92c2b2e30b12a5f8')	"MBA"	"pranai"	"RV"	"4"

The MONGOSH shell shows the command used to insert both documents:

```
> db.branches.insertMany([{"branch_name": "MBA", "director": "pranai", "block_name": "RV", "no_sec": 4}, {"branch_name": "Computers", "director": "dinesh", "block_name": "SSV", "no_sec": 5}])
```

The status of the insertion is shown as:

```
< { acknowledged: true, insertedIds: { '0': ObjectId("655dc4ce92c2b2e30b12a5f8"), '1': ObjectId("655dc4ce92c2b2e30b12a5f9") } }
```

Fig:16

Syntax:

use<database_name>

db.collection_name.insertMany({“field_name”:”value”}, {“field_name”:”value”}, ...{“field_name”:”value”})

O/P:

The screenshot shows the MongoDB Compass interface. On the left, the sidebar lists databases: Bosch, DAV_school, REVA, and admin. Under REVA, there are collections: Student_info and branches (which is selected). The main area displays the 'REVA.branches' collection with 1 document and 1 index. The table has columns: _id, branch_name, director, block_name, and no_sec. The data shows three documents:

	_id	branch_name	director	block_name	no_sec
1	ObjectId('655da6dd92c2b2e30b1...')	"Electronics"	"srinivas"	"CVR"	"4"
2	ObjectId('655dc4ce92c2b2e30b1...')	"MBA"	"pranai"	"RV"	4
3	ObjectId('655dc4ce92c2b2e30b1...')	"Computers"	"dinesh"	"SSV"	5

Below the table are 'ADD DATA' and 'EXPORT DATA' buttons. The bottom of the screen shows the Windows taskbar with icons for File Explorer, Internet Explorer, Google Chrome, and others.

Fig:17

Find document from collection:

Methods to find document from collection.

findOne(): returns first document that matches with specified criteria.

find(): returns a cursor to selected document that matches specified criteria.

find ():

The screenshot shows the MongoDB Compass interface with a mongo shell session in the bottom panel. The session starts with 'db.branches.find()' and then lists three documents from the REVA.branches collection. The documents have the same structure as in Fig:17. The bottom of the screen shows the Windows taskbar with icons for File Explorer, Internet Explorer, Google Chrome, and others.

```
> db.branches.find()
< [
  {
    "_id": ObjectId("655da6dd92c2b2e30b12a5f7"),
    "branch_name": "Electronics",
    "director": "srinivas",
    "block_name": "CVR",
    "no_sec": "4"
  },
  {
    "_id": ObjectId("655dc4ce92c2b2e30b12a5f8"),
    "branch_name": "MBA",
    "director": "pranai",
    "block_name": "RV",
    "no_sec": 4
  },
  {
    "_id": ObjectId("655dc4ce92c2b2e30b12a5f9"),
    "branch_name": "Computers",
    "director": "dinesh",
    "block_name": "SSV",
    "no_sec": 5
  }
]
```

Fig:18

Syntax:

db.collection_name.find()

ex:db.branches.find()

findOne():

REVA.branches

Documents Aggregations Schema Indexes Validation

Type a query: { field: 'value' } or [Generate query](#)

[ADD DATA](#) [EXPORT DATA](#)

branches	_id ObjectId	branch_name String	director String	block_name String	no_sec Mixed
1	ObjectId("655da6dd92c2b2e30b12a5f1...")	"Electronics"	"srinivas"	"CVR"	"4"
2	ObjectId("655dc4ce92c2b2e30b12a5f2...")	"MBA"	"pranai"	"RV"	4
3	ObjectId("655dc4ce92c2b2e30b12a5f3...")	"Computers"	"dinesh"	"SSV"	5

```
>_MONGOSH
}
> db.branches.findOne({"branch_name": "MBA"})
< {
  _id: ObjectId("655dc4ce92c2b2e30b12a5f8"),
  branch_name: 'MBA',
  director: 'pranai',
  block_name: 'RV',
  no_sec: 4
}
REVA >
```

Fig:19

Syntax :

db.collection_name.findOne({“field_name”, “value”})

ex : db.branches.findOne({“branches”：“MBA”})

find().sort() :used to sort the resulted documents in a specified order (ascending order or descending order)

REVA.branches

Documents Aggregations Schema Indexes Validation

Type a query: { field: 'value' } or [Generate query](#)

[ADD DATA](#) [EXPORT DATA](#)

branches	_id ObjectId	branch_name String	director String	block_name String	no_sec Mixed
1	ObjectId("655da6dd92c2b2e30b12a5f7")	"Electronics"	"srinivas"	"CVR"	4
2	ObjectId("655dc4ce92c2b2e30b12a5f8")	"MBA"	"pranai"	"RV"	4
3	ObjectId("655dc4ce92c2b2e30b12a5f9")	"Computers"	"dinesh"	"SSV"	5

```
>_MONGOSH
}
> db.branches.find().sort({"no_sec":1})
< [
  {
    _id: ObjectId("655da6dd92c2b2e30b12a5f7"),
    branch_name: 'Electronics',
    director: 'srinivas',
    block_name: 'CVR',
    no_sec: 4
  },
  {
    _id: ObjectId("655dc4ce92c2b2e30b12a5f8"),
    branch_name: 'MBA',
    director: 'pranai',
    block_name: 'RV',
    no_sec: 4
  },
  {
    _id: ObjectId("655dc4ce92c2b2e30b12a5f9"),
    branch_name: 'Computers',
    director: 'dinesh',
    block_name: 'SSV',
    no_sec: 5
  }
]
REVA > db.branches.find().sort({"no_sec":-1})
```

Fig:20

Syntax: db.collection_name.find().sort({“field_name”:1}) arrange the document in ascending order.

Ex: db.branches.find().sort({“no_sec”:1})

```

>_MONGOSH
}
> db.branches.find().sort({"no_sec": -1})
< [
  {
    _id: ObjectId("655dc4ce92c2b2e30b12a5f9"),
    branch_name: 'Computers',
    director: 'dinesh',
    block_name: 'SSV',
    no_sec: 5
  },
  {
    _id: ObjectId("655da6dd92c2b2e30b12a5f7"),
    branch_name: 'Electronics',
    director: 'srinivas',
    block_name: 'CVR',
    no_sec: 4
  },
  {
    _id: ObjectId("655dc4ce92c2b2e30b12a5f8"),
    branch_name: 'MBA',
    director: 'pranai',
    block_name: 'RV',
    no_sec: 4
  }
]
REVA>

```

Activate Windows
Go to Settings to activate Windows.

Fig:21

Syntax: db.collection_name.find().sort({“field_name”:-1}) arrange the document in descending order.

Ex: db.branches.find().sort({“no_sec”:-1})

Update document from collection in mongoDB compass:

_id	branch_name	director	block_name	no_sec
655da6dd92c2b2e30b12a5f7	Electronics	srinivas	CVR	4
655dc4ce92c2b2e30b12a5f8	MBA	pranai	RV	4
655dc4ce92c2b2e30b12a5f9	Computers	dinesh	SSV	5

```

>_MONGOSH

```

Fig:22

Steps:

Move cursor to edit option as shown in fig22. click the “edit document”.

REVA.branches

Documents Aggregations Schema Indexes Validation

Type a query: { field: 'value' } or [Generate query](#)

[ADD DATA](#) [EXPORT DATA](#)

1 - 3 of 3

[Explain](#) [Reset](#) [Find](#) [Options](#)

[ObjectID](#) [String](#) [String](#) [String](#) [Int32](#)

[Document modified.](#)

[CANCEL](#) [UPDATE](#)

Activate Windows
Go to Settings to activate Windows.

Fig:23

After modifying the required value click on “update” the document is updated in collection

REVA.branches

Documents Aggregations Schema Indexes Validation

Type a query: { field: 'value' } or [Generate query](#)

[ADD DATA](#) [EXPORT DATA](#)

1 - 3 of 3

[Explain](#) [Reset](#) [Find](#) [Options](#)

[ObjectID](#) [String](#) [String](#) [String](#) [Int32](#)

[Document modified.](#)

[Edit](#) [View](#) [Delete](#)

Activate Windows
Go to Settings to activate Windows.

Fig:24

After clicking “update” (fig:23) then the document is updated as shown in figure 24.

Update document in mongosh:

Methods to update documents:

updateOne(): modifies a single document in collection

updateMany(): modifies a multiple document in collection

updateOne

```
_id: ObjectId('655dc4ce92c2b2e30b12a5f8')
branch_name: "MBA"
director: "vinay"
block_name: "RV"
no_sec: 4

>_MONGOSH
{
  block_name: 'SSV',
  no_sec: 5
}
> db.branches.updateOne({director: "vinay"}, {$set: {director: "dev"}})
< {
  acknowledged: true,
  insertedId: null,
  matchedCount: 1,
  modifiedCount: 1,
  upsertedCount: 0
}
REVA >
```

Fig:25

Syntax: db.collection_name.updateOne({_id:"value"},{\$set:{field :"new_value"})

Ex: db.branches.updateOne({director:"vinay"},{\$set:{director :"new_value"})

O/P:

```
_id: ObjectId('655da6dd92c2b2e30b12a5f7')
branch_name: "Electronics"
director: "pavan"
block_name: "CVR"
no_sec: 3

_id: ObjectId('655dc4ce92c2b2e30b12a5f8')
branch_name: "MBA"
director: "dev"
block_name: "RV"
no_sec: 4

_id: ObjectId('655dc4ce92c2b2e30b12a5f9')
branch_name: "Computers"
director: "dinesh"
block_name: "SSV"
no_sec: 5
```

Fig:26

updateMany():

The screenshot shows the MongoDB Compass interface. The top bar indicates the connection is to 'localhost:27017' and the database is 'REVA.branches'. The left sidebar lists databases and collections, with 'REVA' and its 'branches' collection selected. The main area displays the 'Documents' tab for the 'REVA.branches' collection, showing two documents:

```
_id: ObjectId('655dc4ce92c2b2e30b12a5f8')
branch_name: "MBA"
director: "dev"
block_name: "RV"
no_sec: 5
```

```
_id: ObjectId('655dc4ce92c2b2e30b12a5f9')
branch_name: "Computers"
director: "dinesh"
block_name: "SSV"
no_sec: 5
```

The bottom part of the screenshot shows the MONGOSH shell with the command:

```
> db.branches.updateMany({branch_name :{$in :["MBA","Computers"]}},{$set : {no_sec : 5}})
```

The shell output shows the results of the update operation:< {
 acknowledged: true,
 insertedId: null,
 matchedCount: 2,
 modifiedCount: 1,
 upsertedCount: 0
}>

Fig:27

Syntax:

```
db.collection_name.updateMany({_id:{$in:["field1","field2"]}},{$set:{field_name:value}})
```

ex:

```
db.branches.updateMany({branch_name:{$in:["MBA","Computers"]}},{$set:{no_sec:5}})
```

Delete Documents from collection:

Methods to delete documents from collection.

`deleteOne()`: deletes first matching document in a collection.

`deleteMany()`: delete all matching documents in a collection.

`deleteOne()`:

The screenshot shows the MongoDB Compass interface. The left sidebar lists databases and collections, with 'training' selected and 'employee' highlighted. The main area displays the 'training.employee' collection with 5 documents. Each document has fields: name (String), age (Int32), address (String), emp_no (Int32), and email (String). The documents are numbered 1 to 5. The MONGOSH terminal at the bottom shows the command `db.employee.deleteOne({name: "nithin"})` was run, resulting in 1 deleted document.

	name	age	address	emp_no	email
1	"nithin"	22	"kadapa"	1	No field
2	"sona"	22	"pdtr"	2	"mial@gmail.com"
3	"balu"	22	"kadapa"	3	No field
4	"suma"	22	"kadapa"	4	No field
5	"vasu"	22	"pdtr"	5	No field

Fig:28

Having document as shown in fig28.

The screenshot shows the MongoDB Compass interface after a document has been deleted. The left sidebar lists databases and collections, with 'training' selected and 'employee' highlighted. The main area displays the 'training.employee' collection with 4 documents. The MONGOSH terminal at the bottom shows the command `db.employee.deleteOne({name: "nithin"})` was run, resulting in 1 deleted document.

	name	age	email	emp_no	address
1	"sona"	22	"mial@gmail.com"	2	"pdtr"
2	"balu"	22	No field	3	"kadapa"
3	"suma"	22	No field	4	"kadapa"
4	"vasu"	22	No field	5	"pdtr"

Fig:29

Syntax: db.collection_name.deleteOne("field_name": "value")

Ex: db.employee.deleteOne({"name": "nithin"})

O/P: Nithin document is deleted as shown in fig 29.

deleteMany():

The screenshot shows the MongoDB Compass interface. On the left, the database structure is visible with databases like Bosch, DAV_school, REVA, admin, agg, config, details, and local. The main window displays the 'training.employee' collection with 5 documents and 1 index. One document is selected, showing fields: _id, name (sona), age (22), email (mial@gmail.com), emp_no (2), and address ("pdtr"). Below the interface is the MONGOSH shell. A command is run: db.employee.deleteMany({name: {\$in: ["summa", "vasu"]}}). The response shows an acknowledged true, deletedCount 2, and a timestamp of 11/23/2023 at 1:11 AM.

```
>_MONGOSH
>db.employee.deleteMany({name: {$in : ["summa", "vasu"]}})
<{
  acknowledged: true,
  deletedCount: 2
}
training>
```

Fig:30

Syntax: db.collection_name.deleteMany({“field_name”:{\$in :[“field_1”, “field_2”]}})

Ex: db.employee.deleteMany({“name”:{\$in :[“summa”, “vasu”]}})

O/P: summa and vasu documents are deleted from the collection employee as shown in fig30.

Remove database and collection in mongosh:

Drop():

The screenshot shows the MongoDB Compass interface. On the left, the database structure is visible with databases like DAV_school, REVA, admin, agg, config, details, and local. The main window displays the 'DAV_school.Student_marks' collection with 1 document and 1 index. One document is selected, showing fields: _id, 0, and 1. Below the interface is the MONGOSH shell. A command is run: db.Student_marks.drop(). The response shows an ok: 1 and dropped: 'Student_marks'. A timestamp of 11/23/2023 at 3:03 AM is also present.

```
>_MONGOSH
>use Bosch
< switched to db Bosch
>db.basics.drop()
<true
>show collections
<
>db.dropDatabase()
<{ ok: 1, dropped: 'Bosch' }
```

Fig:31

Syntax: db.collection_name.drop() ex : db.basics.drop() //drop collection from database

db.dropdatabase() database is deleted

AGGREGATION

Aggregation in MongoDB

Aggregation is a process of selecting data from a collection in MongoDB. It is a process multiple documents and returns computed results.

Aggregation in MongoDB Compass

\$match:

The screenshot shows the MongoDB Compass interface. On the left, the sidebar lists databases: Bosch, DAV_school, REVA, and others. The REVA database is selected, and its branches collection is highlighted. The main area is titled "REVA.branches" and shows the "Aggregations" tab is active. A message says "Your pipeline is currently empty. Need help getting started? Generate aggregation +". Below it, there's a "Pipeline" section with a "Smatch" button, a "SAVE" button, and a "CREATE NEW" button. To the right, there are buttons for "Explain", "Export", "Run", "More Options", "PREVIEW", "STAGES", and "TEXT". Under "Preview of documents", three documents are shown with their _id, branch_name, director, block_name, and no_sec fields. At the bottom, there's a "Learn more about aggregation pipeline stages" link and a "Add Stage" button.

Fig:32

Steps to use Aggregation in MongoDB compass:

Select database<<select collection<<open aggregation tab in MongoDB Compass<<click “add stage”<<select stage.

This screenshot shows the same MongoDB Compass interface as Fig:32, but with a pipeline added. The "Pipeline" dropdown now shows "Smatch". The "Stage 1 \$match" section contains the MQL query:

```
1 /**  
2  * query: The query in MQL.  
3 */  
4 {  
5   no_sec :{$gt :3}  
6 }
```

 To the right, the "Output after \$match stage (Sample of 2 documents)" shows two documents matching the criteria. The rest of the interface is identical to Fig:32.

Fig:33

REVA<<branches<<Aggregation<<AddStage<<select_stage1(\$match)

Syntax: stage1 : \$match :{//filter criteria//}

Greater than “gt”: operation.

Ex: {no_sec :{\$gt :3}}

\$match:

Less than “lt” operation.

The screenshot shows the MongoDB Compass interface. The left sidebar lists databases and collections, with 'REVA' and 'REVA.branches' selected. The main area displays an aggregation pipeline. Stage 1 is a \$match stage with the query `{no_sec: {$lt: 5}}`. The output pane shows a sample of one document: `_id: ObjectId('655da6dd92c2b2e30b12a5f7'), branch_name: 'Electronics', director: 'pavan', block_name: 'CVR', no_sec: 3`.

Fig:34

Ex: `{no_sec :{$lt:5}}`

O/P: displays documents having less than 5 in no_sec.

\$match:

Greater than or equals to “gte”

The screenshot shows the MongoDB Compass interface. The left sidebar lists databases and collections, with 'REVA' and 'REVA.branches' selected. The main area displays an aggregation pipeline. Stage 1 is a \$match stage with the query `{no_sec: {$gte: 5}}`. The output pane shows two documents: `_id: ObjectId('655dc4ce92c2b2e30b12a5f8'), branch_name: 'MBA', director: 'dev', block_name: 'RV', no_sec: 5` and `_id: ObjectId('655dc4ce92c2b2e30b12a5f9'), branch_name: 'Computers', director: 'dinesh', block_name: 'SSV', no_sec: 5`.

Fig:35

Ex: `{no_sec :{$gte :5}}`

O/P: displays the documents where no_sec are greater than equal to 5

\$match :

Less than or equal to “lte”.

The screenshot shows the MongoDB Compass interface. The left sidebar lists databases and collections, with 'REVA' and 'REVA.branches' selected. The main area is titled 'REVA.branches' and shows the 'Aggregations' tab selected. A pipeline is defined with a single stage: Stage 1 (\$match). The query is `{no_sec: {$lte: 3}}`. The output panel shows a sample of one document:

```
_id: ObjectId('655da6dd92c2b2e30b12a5f7')
branch_name: "Electronics"
director: "pavan"
block_name: "CVR"
no_sec: 3
```

Fig:36

Ex: {no_sec :{\$lte: 3}}

O/P: displays the documents where no_sec are less than equal to 3.

\$match:

Not equal to “ne”

The screenshot shows the MongoDB Compass interface. The left sidebar lists databases and collections, with 'REVA' and 'REVA.branches' selected. The main area is titled 'REVA.branches' and shows the 'Aggregations' tab selected. A pipeline is defined with a single stage: Stage 1 (\$match). The query is `{no_sec: {$ne: 3}}`. The output panel shows two samples of documents where no_sec is not 3:

```
_id: ObjectId('655dc4ce92c2b2e30b12a5f8')
branch_name: "MBA"
director: "dev"
block_name: "RV"
no_sec: 5
```

```
_id: ObjectId('655dc4ce92c2b2e30b12a5f9')
branch_name: "Computers"
director: "dinesh"
block_name: "SSV"
no_sec: 5
```

Fig:37

Ex: {no_sec :{\$ne:3}}

O/P: displays the documents where no_sec are not equal to 3

\$match:

Equal to “eq”

The screenshot shows the MongoDB Compass interface. On the left, the sidebar lists databases and collections, with 'REVA.branches' selected. In the main area, the 'Aggregations' tab is active. A pipeline is being built under the 'Untitled - modified' pipeline. The first stage is a '\$match' stage, which contains the MQL query: `1 /** 2 * query: The query in MQL. 3 */ 4 { 5 no_sec : {$eq : 3} 6 }`. To the right of the stage, the output is shown: 'Output after \$match stage (Sample of 1 document)' with one document: `_id: ObjectId('655da6dd92c2b2e30b12a5f7') branch_name: "Electronics" director: "pavan" block_name: "CVR" no_sec: 3`. Below the pipeline, there's a note: 'Learn more about aggregation pipeline stages'.

Fig:38

Ex:{no_sec :{\$eq :3}}

O/P: displays the documents where no_sec are equal to 3.

Aggregation \$group:

\$group: stage in MongoDB aggregation framework is used to group documents by a specified expression or field and perform aggregate operations on them.

Operations in \$group :SUM, AVG, MIN, MAX.

The screenshot shows the MongoDB Compass interface. The sidebar lists databases and collections, with 'training.sum_agg' selected. In the main area, the 'Documents' tab is active. There are four documents listed: 1. `_id: ObjectId('655d058410a2869aee85fdc2') name: "harshi" subject: "english" marks: 54`. 2. `_id: ObjectId('655d058410a2869aee85fdc3') name: "tillu" subject: "english" marks: 50`. 3. `_id: ObjectId('655d062210a2869aee85fdc5') name: "harshi" subject: "maths" marks: 54`. 4. `_id: ObjectId('655d065510a2869aee85fdc7') name: "tillu" subject: "maths" marks: 54`. The bottom status bar shows the date and time: 11/22/2023 8:38 PM.

Fig:39

Goto training<<collection sum_agg<<insert document as shown in fig39

localhost:27017

Aggregations training.sum_agg

4 DOCUMENTS 1 INDEXES

Documents Aggregations Schema Indexes Validation

Pipeline \$group Untitled - modified SAVE CREATE NEW EXPORT TO LANGUAGE PREVIEW STAGES TEXT More Options

Stage 1 \$group

```

1 /**
2  * _id: The id of the group.
3  * fieldN: The first field name.
4 */
5 {
6   _id: "$name", total_avg: {$avg: "$marks"}
7 }
  
```

Output after \$group stage (Sample of 2 documents)

_id: "tillu" total_avg: 52	_id: "harshi" total_avg: 54
-------------------------------	--------------------------------

Add Stage Learn more about aggregation pipeline stages

Fig:40

Syntax: \$GROUP AVG: Calculates the average value of specified numeric field within each group.

{_id: “field_name”, total_avg: {\$avg: “\$NumericField”}}

Ex: {_id: “\$name”,total_avg :{\$avg : “\$marks”}}

\$GROUP SUM: Calculates the sum of the specified numeric field within each group.

localhost:27017

Aggregations training.sum_agg

4 DOCUMENTS 1 INDEXES

Documents Aggregations Schema Indexes Validation

Pipeline \$group Untitled - modified SAVE CREATE NEW EXPORT TO LANGUAGE PREVIEW STAGES TEXT More Options

Stage 1 \$group

```

1 /**
2  * _id: The id of the group.
3  * fieldN: The first field name.
4 */
5 {
6   _id: "$name", total_sum: {$sum: "$marks"}
7 }
  
```

Output after \$group stage (Sample of 2 documents)

_id: "tillu" total_sum: 104	_id: "harshi" total_sum: 108
--------------------------------	---------------------------------

Add Stage Learn more about aggregation pipeline stages

Fig:41

Syntax: {_id : “\$field_name”,total_sum :{\$sum :”\$NumericField”}}

Ex: {_id : “\$name”,total_sum :{\$sum :”\$Marks”}}

\$group MIN: used to find the minimum value of a specified expression within a group.

The screenshot shows the MongoDB Compass interface. On the left sidebar, under the 'training' database, the 'sum_agg' collection is selected. In the main area, the 'Aggregations' tab is active. A pipeline is being built with one stage: '\$group'. The pipeline document is shown as:

```
1 /**
2  * _id: The id of the group.
3  * fieldN: The first field name.
4 */
5 {
6   _id: "$name", min: {$min: "$marks"}
7 }
```

The output after the '\$group' stage is displayed as a sample of two documents:

- For '_id: "tillu"', the 'min' value is 50.
- For '_id: "harshi"', the 'min' value is 54.

Fig:42

Syntax: `{_id : "$field_name", minimum :{$min :"$NumericField"} }`

Ex: `{_id : "$name",min:{$min:"$marks"} }`

\$group MAX: used to find the maximum value of a specified expression within a group.

The screenshot shows the MongoDB Compass interface, identical to Fig:42 but with a different output. The output after the '\$group' stage is displayed as a sample of two documents:

- For '_id: "tillu"', the 'max' value is 54.
- For '_id: "harshi"', the 'max' value is 54.

Fig:43

Syntax: `{_id : "$field_name", maximum :{$max :"$NumericField"} }`

Ex: `{_id : "$name",max:{$max:"$marks"} }`

\$limit: used to restrict the number of documents that passes through the pipeline. Often to employ optimize performance by limiting the amount of data processed.

The screenshot shows the MongoDB Compass interface for the 'student_data' collection. The 'Aggregations' tab is selected. A pipeline is defined with a single stage: Stage 1 (\$limit). The code for this stage is: `1 //** 2 * Provide the number of document 3 */ 4 2`. The output after the \$limit stage shows two sample documents:

```
_id: ObjectId('655ade748a4a0a3c454292e3')
name: "nithin"
class: 10
section: "a"
```

```
_id: ObjectId('655adf488a4a0a3c454292e6')
name: "sona"
class: 10
section: "b"
roll_num: 2
```

Fig:44

Syntax: {\$limit : value}

Ex: 2

\$Skip: used to skip a specified number of documents in the pipeline before passing the remaining documents to the next stage.

The screenshot shows the MongoDB Compass interface for the 'student_data' collection. The 'Aggregations' tab is selected. A pipeline is defined with a single stage: Stage 1 (\$skip). The code for this stage is: `1 //** 2 * Provide the number of documents to skip 3 */ 4 3`. The output after the \$skip stage shows one sample document:

```
_id: ObjectId('655ae1518a4a0a3c454292e9')
name: "vinod"
class: 10
section: "a"
roll_num: 3
```

Fig:45

Syntax: {\$Skip: "number"}

Ex: 3

\$unwind: used to deconstruct an array field, creating a separate document for each array element.

The screenshot shows the MongoDB Compass interface. The left sidebar lists databases and collections, with 'training.unwind_marks' selected. The main area shows the 'Documents' tab for 'training.unwind_marks'. Two documents are listed:

```
_id: ObjectId('655d10de10a2869aee85fdd4')
name: "nithin"
subjects: Array (2)
  0: "maths"
  1: "english"

_id: ObjectId('655d10de10a2869aee85fdd5')
name: "sona"
subjects: Array (2)
  0: "maths"
  1: "english"
```

At the bottom right, there are icons for edit, delete, and more options.

Fig:46

Consider fig46 documents field_name “subjects” consist of two arrays.

The screenshot shows the MongoDB Compass interface with the 'Aggregations' tab selected. An aggregation pipeline is displayed:

```
Stage 1: $unwind
```

The 'STAGE INPUT' section shows two sample documents:

```
_id: ObjectId('655d10de10a2869aee85fdd4')
name: "nithin"
subjects: Array (2)

_id: ObjectId('655d10de10a2869aee85fdd5')
name: "sona"
subjects: Array (2)
```

The 'STAGE OUTPUT' section shows the result of the \$unwind stage, which produces four documents:

```
_id: ObjectId('655d10de10a2869aee85fdd4')
name: "nithin"
subjects: "maths"

_id: ObjectId('655d10de10a2869aee85fdd4')
name: "nithin"
subjects: "english"

_id: ObjectId('655d10de10a2869aee85fdd5')
name: "sona"
subjects: "maths"

_id: ObjectId('655d10de10a2869aee85fdd5')
name: "sona"
subjects: "english"
```

Fig:47

Syntax \$unwind {path:” \$array_field}

Ex: {path: “\$subjects”}

O/P: separate documents are created for each array element.

\$addFields: used to add new field to the document in pipeline. It allows derived values based on existing fields.

\$addFields having operations SUM, AVG, MIN, MAX.

The screenshot shows the MongoDB Compass interface. The left sidebar lists databases and collections, with 'DAV_school' expanded to show 'Class_students', 'Nithin_marks', and 'Student_marks'. The main area is titled 'training.Aggregation' and shows a table with four documents. The columns are '_id ObjectId', 'name String', 'maths Int32', and 'english Int32'. The documents are:

_id	name	maths	english
ObjectId('655c4d6effb538f1770...')	"nithin"	45	35
ObjectId('655c4d6effb538f1770...')	"raju"	49	39
ObjectId('655c4d6effb538f1770...')	"sandeep"	50	50
ObjectId('655cf6e010a2869aee8...')	"dhanu"	44	55

The bottom left shows the aggregation pipeline stages:

- Stage 1: \$addFields
- Stage 2: \$group
- Stage 3: \$sort
- Stage 4: \$project

Fig:48

Consider fig48 is the document for addFields operations.

\$addFields \$sum:

The screenshot shows the MongoDB Compass interface with the aggregation pipeline editor open. The pipeline consists of four stages:

- Stage 1: \$addFields**:
Options: Enabled
Input: Sample of 4 documents
Output: Sample of 4 documents
Stage Input:
_id: ObjectId('655c4d6effb538f177074f61...')
name: "nithin"
maths: 45
english: 35

_id: ObjectId('655c4d6effb538f177074f62...')
name: "raju"
maths: 49
english: 39

_id: ObjectId('655c4d6effb538f177074f63...')
name: "sandeep"
maths: 50
english: 50

_id: ObjectId('655cf6e010a2869aee85fdb...')
name: "dhanu"
maths: 44
english: 55

Stage Output:
_id: ObjectId('655c4d6effb538f177074f61...')
name: "nithin"
maths: 45
english: 35
total_sum: 80

_id: ObjectId('655c4d6effb538f177074f62...')
name: "raju"
maths: 49
english: 39
total_sum: 88

_id: ObjectId('655c4d6effb538f177074f63...')
name: "sandeep"
maths: 50
english: 50
total_sum: 100

_id: ObjectId('655cf6e010a2869aee85fdb...')
name: "dhanu"
maths: 44
english: 55
total_sum: 99
- Stage 2: \$group**:
Group By: name
Initial Value: {
 '_id': null,
 'total_sum': 0
}
Final Value: {
 '_id': null,
 'total_sum': 0
}
- Stage 3: \$sort**:
Sort By: total_sum
- Stage 4: \$project**:
Fields:
 - name
 - maths
 - english
 - total_sum

Fig:49

Syntax: \$addFields {new_field :{\$sum :["field1","field2"]}}

Ex: \$addFields {total_marks :{\$sum :["maths","english"]}}

\$addFields \$avg :

The screenshot shows the MongoDB Compass interface with the aggregation pipeline editor. The stage input contains four documents with fields _id, name, maths, and english. The stage output shows the same documents with an additional field 'average' calculated as the average of 'maths' and 'english'. The aggregation pipeline stage is labeled '\$addFields'.

```
1  /**
2   * newField: The new field name.
3   * expression: The new field expression.
4   */
5  {
6    average: {$avg : ["$maths","$english"]}
7 }
```

STAGE INPUT
Sample of 4 documents

STAGE OUTPUT
Sample of 4 documents

Fig:50

Syntax: \$addFields {new_field :{\$vg :["field1","field2"]}}

Ex: \$addFields {average :{\$avg :["maths","english"]}}

\$addFields \$min

The screenshot shows the MongoDB Compass interface with the aggregation pipeline editor. The stage input contains four documents with fields _id, name, maths, and english. The stage output shows the same documents with an additional field 'minimum_marks' calculated as the minimum of 'maths' and 'english'. The aggregation pipeline stage is labeled '\$addFields'.

```
1  /**
2   * newField: The new field name.
3   * expression: The new field expression.
4   */
5  {
6    minimum_marks : {$min : ["$maths","$english"]}
7 }
```

STAGE INPUT
Sample of 4 documents

STAGE OUTPUT
Sample of 4 documents

Fig:51

Syntax: \$addFields {new_field :{\$min :["field1","field2"]}}

Ex: \$addFields {minimum_marks :{\$min :["maths","english"]}}

\$addField \$max:

The screenshot shows the MongoDB Compass interface for an aggregation pipeline. The pipeline consists of one stage, "Stage 1: \$addFields", which is currently enabled. The stage input shows four documents with fields: _id, name, maths, and english. The stage output shows the same four documents with an additional field, maximum_marks, which is the result of applying the \$max operator to the maths and english fields. The Compass interface includes a code editor at the top where the aggregation pipeline is defined.

Fig:52

Syntax: \$addFields {new_field :{\$max :["field1","field2"]}}

Ex: \$addFields {maximum_marks :{\$max :["maths","english"]}}

Aggregation in mongosh:

\$addFields: \$sum

The screenshot shows the Mongosh shell interface. A command is being run to aggregate documents in the "training.Aggregation" collection. The command uses the \$addFields stage to calculate the total_marks for each document by summing the maths and english fields. The resulting documents now include a total_marks field. The Mongosh interface also displays statistics for the collection: 4 documents and 1 index.

Fig:53

Syntax: db.Collection_name.aggregate({\$addFields :new_field :{\$sum :["\$field1","\$field2"]}}})

Ex: db.Aggregation.aggregate({\$addFields:{ total_marks :{\$sum :["\$maths","\$english"]}}})

\$addField \$avg:

The screenshot shows the MongoDB Compass interface. The top bar indicates "localhost:27017" and "Aggregations training.Aggregation". The main area displays the results of a query: "db.Aggregation.aggregate({\$addFields :{average :{\$avg :["\$maths","\$english"]}}})". The results show four documents with fields _id, name, maths, english, and average. The average field is calculated as the sum of maths and english divided by 2. The bottom right corner shows system status: "Activate Windows Go to Settings to activate Windows.", the date "11/23/2023", and the time "2:48 AM".

```
>_MONGOSH
> db.Aggregation.aggregate({$addFields :{average :{$avg :["$maths","$english"]}}})
< [
  {
    "_id": ObjectId("655c4d6effb538f177074f61"),
    "name": "nithin",
    "maths": 45,
    "english": 35,
    "average": 40
  },
  {
    "_id": ObjectId("655c4d6effb538f177074f62"),
    "name": "raju",
    "maths": 49,
    "english": 39,
    "average": 44
  },
  {
    "_id": ObjectId("655c4d6effb538f177074f63"),
    "name": "sandeep",
    "maths": 50,
    "english": 50,
    "average": 50
  },
  {
    "_id": ObjectId("655cf6e010a2869aee85fdb")
  }
]
```

Fig:54

Syntax: db.Collection_name.aggrigate({\$addFields :new_field :{\$avg :["\$field1","\$field2"]}}{})

Ex: db.Aggregation.aggrigate({\$addFields:{ average :{\$avg :["\$maths","\$english"]}}{} })

\$addField \$min:

The screenshot shows the MongoDB Compass interface. The top bar indicates "localhost:27017" and "Aggregations training.Aggregation". The main area displays the results of a query: "db.Aggregation.aggregate({\$addFields :{minimum :{\$min :["\$maths","\$english"]}}})". The results show four documents with fields _id, name, maths, english, and minimum. The minimum field is the lower of the two values. The bottom right corner shows system status: "Activate Windows Go to Settings to activate Windows.", the date "11/23/2023", and the time "2:49 AM".

```
>_MONGOSH
> db.Aggregation.aggregate({$addFields :{minimum :{$min :["$maths","$english"]}}})
< [
  {
    "_id": ObjectId("655c4d6effb538f177074f61"),
    "name": "nithin",
    "maths": 45,
    "english": 35,
    "minimum": 35
  },
  {
    "_id": ObjectId("655c4d6effb538f177074f62"),
    "name": "raju",
    "maths": 49,
    "english": 39,
    "minimum": 39
  },
  {
    "_id": ObjectId("655c4d6effb538f177074f63"),
    "name": "sandeep",
    "maths": 50,
    "english": 50,
    "minimum": 50
  },
  {
    "_id": ObjectId("655cf6e010a2869aee85fdb")
  }
]
```

Fig:55

Syntax: db.Collection_name.aggrigate({\$addFields :new_field :{\$min :["\$field1","\$field2"]}}{})

Ex: db.Aggregation.aggrigate({\$addFields:{ minimum_marks :{\$min :["\$maths","\$english"]}}{} })

\$addFields \$max:

The screenshot shows the MongoDB Compass interface with the database set to 'localhost:27017' and the collection to 'Aggregations'. The query entered is: `> db.Aggregation.aggregate({$addFields :{maximum :{$max :["$maths","$english"]}}})`. The results pane displays four documents, each with an '_id' field and a 'maximum' field containing the maximum value of 'maths' and 'english' for that student. The students listed are nithin, raju, sandeep, and sona.

```
minimum: 44
}
> db.Aggregation.aggregate({$addFields :{maximum :{$max :["$maths","$english"]}}})
< [
  {
    _id: ObjectId("655c4d6effb538f177074f61"),
    name: 'nithin',
    maths: 45,
    english: 35,
    maximum: 45
  }
  {
    _id: ObjectId("655c4d6effb538f177074f62"),
    name: 'raju',
    maths: 49,
    english: 39,
    maximum: 49
  }
  {
    _id: ObjectId("655c4d6effb538f177074f63"),
    name: 'sandeep',
    maths: 50,
    english: 50,
    maximum: 50
  }
  {
    _id: ObjectId("655cf6e010a2869aee85fdb")
  }
]
```

Fig:56

Syntax: `db.Collection_name.aggregate({$addFields :new_field :{$max :["$field1","$field2"]}}{})`

Ex: `db.Aggregation.aggregate({$addFields:{ maximum_marks :{$max :["$maths","$english"]}}})`

\$unwind in Mongosh:

The screenshot shows the Mongosh shell with the database set to 'localhost:27017/training'. The collection is 'unwind_marks'. The command run is: `> db.unwind_marks.aggregate({$unwind :"$subjects"})`. The results show the subjects for each student. The student nithin has two subjects: 'maths' and 'english'. The student sona also has two subjects: 'maths' and 'english'.

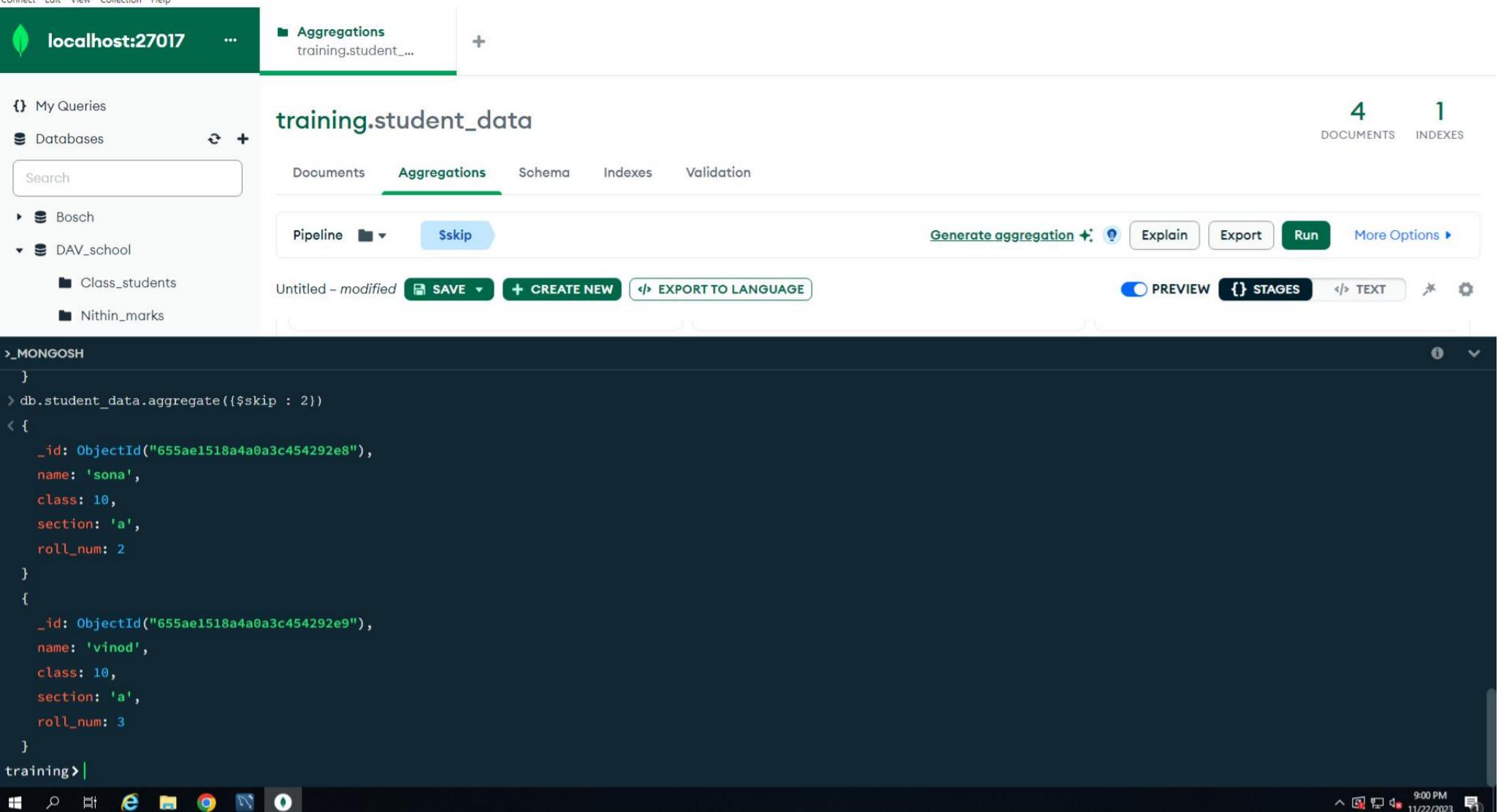
```
Documents Aggregations Schema Indexes Validation
2 1
DOCUMENTS INDEXES
>_MONGOSH
> db.unwind_marks.aggregate({$unwind :"$subjects"})
< [
  {
    _id: ObjectId("655d10de10a2869aee85fdd4"),
    name: 'nithin',
    subjects: 'maths'
  }
  {
    _id: ObjectId("655d10de10a2869aee85fdd4"),
    name: 'nithin',
    subjects: 'english'
  }
  {
    _id: ObjectId("655d10de10a2869aee85fdd5"),
    name: 'sona',
    subjects: 'maths'
  }
  {
    _id: ObjectId("655d10de10a2869aee85fdd5"),
    name: 'sona',
    subjects: 'english'
  }
]
```

Fig:57

Syntax: `db.collection_name.aggregate({$unwind :"$array_field"})`

Ex: `db.unwind_marks.aggregate({$unwind :"$subjects"})`

\$Skip in Mongosh:



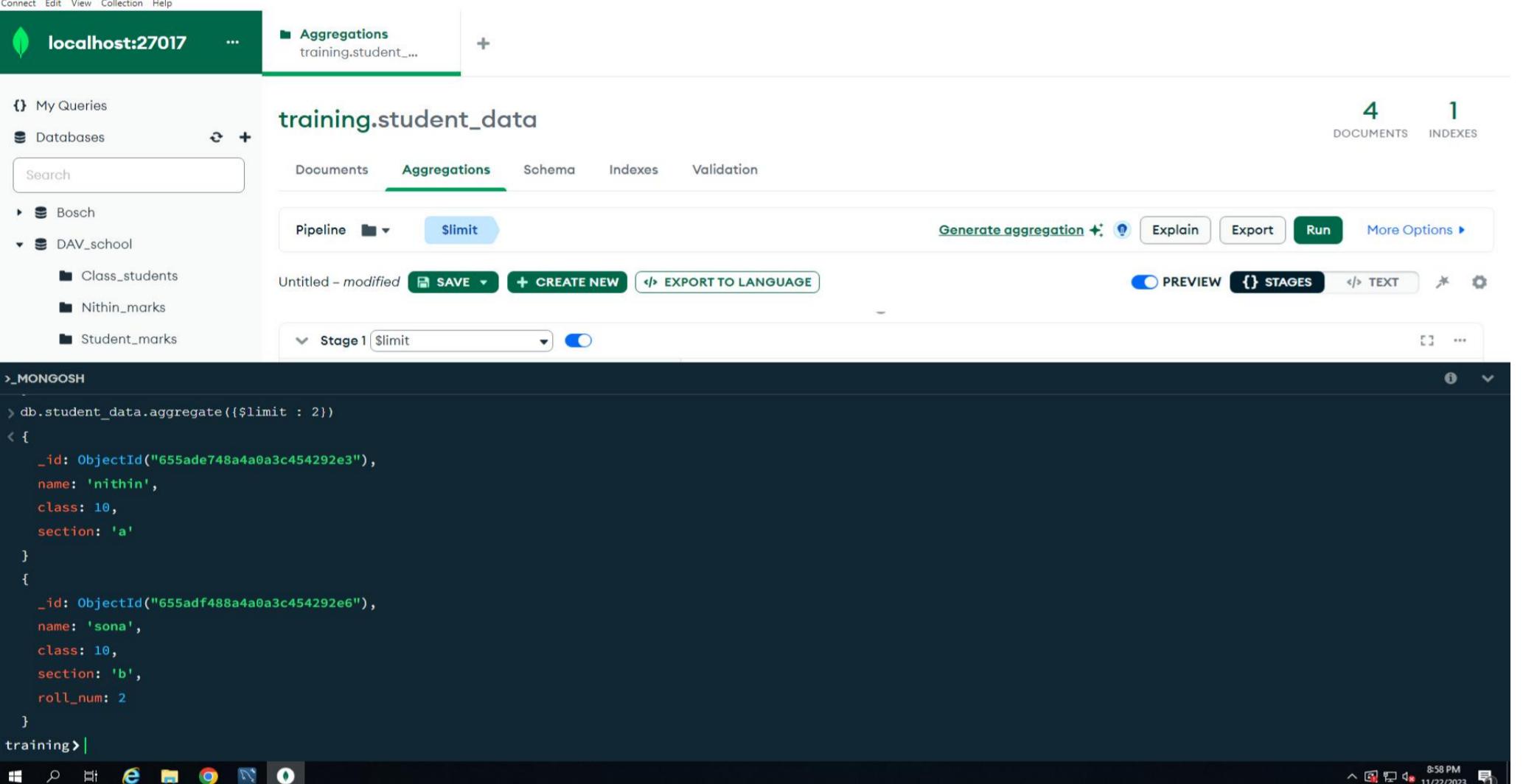
```
>_MONGOSH
}
> db.student_data.aggregate({$skip : 2})
< [
  {
    _id: ObjectId("655ae1518a4a0a3c454292e8"),
    name: 'sona',
    class: 10,
    section: 'a',
    roll_num: 2
  },
  {
    _id: ObjectId("655ae1518a4a0a3c454292e9"),
    name: 'vinod',
    class: 10,
    section: 'a',
    roll_num: 3
  }
]
training> |
```

Fig:58

Syntax: db.collection_name.aggregate({\$skip :number})

Ex: db.unwind_marks.aggregate({\$skip :2})

\$limit in Mongosh:



```
>_MONGOSH
}
> db.student_data.aggregate({$limit : 2})
< [
  {
    _id: ObjectId("655ade748a4a0a3c454292e3"),
    name: 'nithin',
    class: 10,
    section: 'a'
  },
  {
    _id: ObjectId("655adf488a4a0a3c454292e6"),
    name: 'sona',
    class: 10,
    section: 'b',
    roll_num: 2
  }
]
training> |
```

Fig:59

Syntax: db.collection_name.aggregate({\$unwind :"\$array_field"})

Ex: db.unwind_marks.aggregate({\$unwind :"\$subjects"})

\$group in Mongosh:

\$group \$sum :

MongoDB Compass - localhost:27017/training.sum_agg

localhost:27017

Documents training.sum_agg

4 DOCUMENTS 1 INDEXES

My Queries Databases

Search

REVA admin agg config details local training

Aggregation employee newdata

training.sum_agg

Documents Aggregations Schema Indexes Validation

Filter Type a query: { field: 'value' } or Generate query +.

EXPLAIN Reset FIND Options

ADD DATA EXPORT DATA

1 - 4 of 4

_id: ObjectId('655d058410a2869aee85fdc2')
name: "harshi"
subject: "english"
marks: 54

_id: ObjectId('655d058410a2869aee85fdc3')
name: "tillu"
subject: "english"
marks: 50

>_MONGOSH

>use training
< switched to db training
>db.sum_agg.aggregate({\$group :{_id :"\$name",total_marks :{\$sum : "\$marks"}}})

<{
 _id: 'harshi',
 total_marks: 108
}
{
 _id: 'tillu',
 total_marks: 104
}

training>

Fig:60

Syntax: db.collection_name.aggregate({\$group :{_id: “field_name”, total_marks: {\$sum: “\$NumericField”}}})

Ex: db.sum_agg.aggregate({\$group :{_id :"\$name",total_marks :{\$sum: "\$marks"}}})

\$group \$avg :

MongoDB Compass - localhost:27017/training.sum_agg

localhost:27017

Documents training.sum_agg

4 DOCUMENTS 1 INDEXES

My Queries Databases

Search

REVA admin agg config details local training

Aggregation employee newdata

training.sum_agg

Documents Aggregations Schema Indexes Validation

Filter Type a query: { field: 'value' } or Generate query +.

EXPLAIN Reset FIND Options

ADD DATA EXPORT DATA

1 - 4 of 4

_id: ObjectId('655d058410a2869aee85fdc2')
name: "harshi"
subject: "english"
marks: 54

_id: ObjectId('655d058410a2869aee85fdc3')
name: "tillu"
subject: "english"
marks: 50

>_MONGOSH

>total_marks: 104

}

>db.sum_agg.aggregate({\$group :{_id :"\$name",total_avg :{\$avg : "\$marks"}}})

<{
 _id: 'harshi',
 total_avg: 54
}
{
 _id: 'tillu',
 total_avg: 52
}

training>

Fig:61

Syntax: db.collection_name.aggregate({\$group :{_id: “field_name”, total_avg: {\$avg: “\$NumericField”}}})

Ex: db.sum_agg.aggregate({\$group :{_id :"\$name",total_avg :{\$avg: "\$marks"}}})

\$group \$min:

```
total_avg: 52
}
> db.sum_agg.aggregate({$group :{_id :"$name",minimum_marks :{$min : "$marks"}}})
<
{
  _id: 'harshi',
  minimum_marks: 54
}
{
  _id: 'tillu',
  minimum_marks: 50
}
training>
```

Fig:62

Syntax: db.collection_name.aggregate({\$group :{_id: “field_name”, minimum: {\$min: “\$NumericField”}}})

Ex: db.sum_agg.aggregate({\$group :{_id :"\$name”,minimum :{\$min: “\$marks”}}})

\$group \$max:

```
minimum_marks: 50
}
> db.sum_agg.aggregate({$group :{_id :"$name",maximum_marks :{$max : "$marks"}}})
<
{
  _id: 'tillu',
  maximum_marks: 54
}
{
  _id: 'harshi',
  maximum_marks: 50
}
training>
```

Fig:63

Syntax: db.collection_name.aggregate({\$group :{_id: “field_name”, total_marks: {\$sum: “\$NumericField”}}})

Ex: db.sum_agg.aggregate({\$group :{_id :"\$name”,total_marks :{\$sum: “\$marks”}}})

Schema

Schema in MongoDB serves as a crucial tool for structuring, organizing and maintaining the consistency and integrity of data within a collection.

The screenshot shows the MongoDB Compass interface for the REVA.Student_info collection. The left sidebar lists databases and collections, with 'Student_info' selected. The main area displays the schema for a single document. The 'Schema' tab is active, showing fields like '_id' (objectid), 'age' (int32), and 'branch' (string). The 'branch' field has a value of 'electronics and communication'. Below the schema, there are sections for 'Indexes' and 'Validation'. A status bar at the bottom indicates 2 documents and 1 index.

MongoDB Compass - localhost:27017/REVA.Student_info

localhost:27017

REVA.Student_info

My Queries

Databases

Search

Documents Aggregations Schema Indexes Validation

Filter Type a query: { field: 'value' } or [Generate query](#).

This report is based on a sample of 2 documents. [Learn more](#)

_id
objectid
S M T W T F S 0:00 6:00 12:00 18:00 23:00
inserted: 2023-11-22 06:36:41

age
int32 22 21

branch
string electronics and communication

2 DOCUMENTS 1 INDEXES

>_MONGOSH

MongoDB Compass - localhost:27017/REVA.Student_info

localhost:27017

REVA.Student_info

My Queries

Databases

Search

Documents Aggregations Schema Indexes Validation

Filter Type a query: { field: 'value' } or [Generate query](#).

This report is based on a sample of 2 documents. [Learn more](#)

branch
string electronics and communication

name
string nikki raj

subjects
array
string
Array lengths
min: 3
average: 3.0
max: 3

2 DOCUMENTS 1 INDEXES

>_MONGOSH

In MongoDB, a schema acts like a blueprint, providing guidelines for how data is structured and maintained within a collection.

With the features such as unique constraints, default values and supports for relationships between documents.

INDEX

Indexes are crucial for speeding up queries in a database. They acts like a roadmap, helping quickly locate and access data.

The screenshot shows the MongoDB Compass interface. The left sidebar lists databases and collections, with 'Aggregation' selected under the 'training' database. The main area displays the 'Documents' tab for the 'training.Aggregation' collection. A search bar at the top contains the query '{name: "dhanu"}'. Below it, four documents are listed, each with its _id, name, maths, and english scores. The total count is 4 documents and 1 index.

Fig:66

Select the data base “training”>>collection>>Aggregation>>write a queary as shown in fig66”name:dhanu”>>click on button “find”. New page is opens as shown in fig67

The screenshot shows the MongoDB Compass interface with the 'Explain Plan' dialog open. The dialog title is 'Explain Plan' and it explains that Explain provides key execution metrics to help diagnose slow queries and optimize index usage. It shows a 'Visual Tree' tab selected. The main section shows a 'COLLSCAN' operation with the following details: Returned 1 document, Execution Time 0 ms, and Documents Examined: 4. To the right, the 'Query Performance Summary' panel shows: 1 documents returned, 4 documents examined, 0 ms execution time, Is not sorted in memory, 0 index keys examined, and a note that No index available for this query.

Fig:67

To find document (name:"dhanu") documents examined 4 as shown without involvement of indexes.

Creating an Index in MongoDB compass:

The screenshot shows the MongoDB Compass interface. The left sidebar lists databases: DAV_school, REVA, and MONGOSH. Under REVA, the 'Student_info' collection is selected. The main area displays the 'Indexes' tab for the 'REVA.Student_info' collection. A single index, '_id_', is listed. It is a regular index (Type: REGULAR) with a size of 20.5 KB and usage since Sun Nov 26 2023. The index is marked as UNIQUE. The top right corner shows 2 DOCUMENTS and 1 INDEXES.

Fig:68

Go to Index>>click on create Index. New page is displayed as shown in fig 69.

The screenshot shows the MongoDB Compass interface. The left sidebar lists databases: REVA, training, and MONGOSH. Under training, the 'Aggregation' collection is selected. The main area displays the 'Indexes' tab for the 'training.Aggregation' collection. A modal dialog titled 'Create Index' is open. In the 'Index fields' section, 'name' is selected with type '1 (asc)'. The 'Create Index' button is visible at the bottom right of the dialog. The top right corner shows 4 DOCUMENTS and 1 INDEXES.

Fig:69

Select the required index fields and type as shown in fig69.

Selected field "name" and type "1asc".

Click on Create Index as show in fig69.

localhost:27017

Indexes
training.Aggregation

My Queries
Databases

Search

Documents Aggregations Schema Indexes Validation

Create Index Refresh

VIEWING INDEXES SEARCH INDEXES

Name and Definition Type Size Usage Properties

id REGULAR 36.9 KB 5 (since Sun Nov 26 2023) UNIQUE

name_1 REGULAR 20.5 KB 0 (since Sun Nov 26 2023)

REVA
Student_info
branches
admin
agg
config
details
local
training
Aggregation
employee
newdata
student_data
sum_agg
 unwind_marks

>_MONGOSH

Fig:70

Name_1 index is created as shown in fig70.

Follow the process as shown in fig66.

MongoDB Compass - localhost:27017/training.Aggregation

Documents

Explain Plan

Visual Tree Raw Output

FETCH
Returned 1 Execution Time 0 ms

IXSCAN
Returned 1 Execution Time 0 ms
Index Name: name_1
Multi Key Index: no

Query Performance Summary
1 documents returned
1 documents examined
0 ms execution time
Is not sorted in memory
1 index keys examined
Query used the following index:
name ↑

<_MONGOSH

Fig:71

The number of documents to examine are decreased. Which means time taking to find a document is decreases.

Dropping an Index in MongoDB compass:

MongoDB Compass - localhost:27017/training.Aggregation

localhost:27017 ...

Indexes training.Aggrega...

My Queries Databases Search

REVA Student_info branches admin agg config details local training Aggregation employee newdata student_data sum_agg unwind_marks

4 1 DOCUMENTS INDEXES

Documents Aggregations Schema Indexes Validation

Create Index Refresh

VIEWING INDEXES SEARCH INDEXES

Name and Definition	Type	Size	Usage	Properties
id	REGULAR	36.9 KB	5 (since Sun Nov 26 2023)	UNIQUE
name_1	REGULAR	20.5 KB	0 (since Sun Nov 26 2023)	

Drop Index name_1

_MONGOSH

Fig:72

Click on dropIndex as shown in fig72. New page is opens as shown in fig73.

MongoDB Compass - localhost:27017/training.Aggregation

localhost:27017 ...

Indexes training.Aggrega...

My Queries Databases Search

REVA Student_info branches admin agg config details local training Aggregation employee newdata student_data sum_agg unwind_marks

4 1 DOCUMENTS INDEXES

Documents Aggregations Schema Indexes Validation

Create Index Refresh

VIEWING INDEXES SEARCH INDEXES

Drop Index

Type the index name name_1 to drop

name_1

Cancel Drop

_MONGOSH

Fig:73

Enter the name which index wanted to drop “name_1”. Click on the Drop as show in fig73.

Index is dropped.

Creating an Index in Mongosh: Single Field Index

MongoDB Compass - localhost:27017/training.sum_agg
Connect Edit View Collection Help

localhost:27017 ... Documents training.sum_agg +

My Queries Databases Search Nithin_marks Student_marks REVA Student_info branches admin agg config details local training Aggregation employee newdata student_data sum_agg ... unwind_marks

training.sum_agg

Documents Aggregations Schema Indexes Validation

Filter ⓘ [name: "harshi"] Generate query Explain Reset Find Options

ADD DATA EXPORT DATA 1 - 4 of 4

`_id: ObjectId('655d058410a2869aee85fdc2')
name: "harshi"
subject: "english"
marks: 54`

`_id: ObjectId('655d058410a2869aee85fdc3')
name: "tillu"
subject: "english"
marks: 50`

`_id: ObjectId('655d062210a2869aee85fdc5')
name: "harshi"
subject: "maths"
marks: 54`

`_id: ObjectId('655d065510a2869aee85fdc7')
name: "tillu"
subject: "maths"
marks: 54`

> MONGOSH 12:01 PM 11/26/2023

Fig:74

Select database training <<collection “sum_agg”<<write query as shown {name:”harshi”}.

Click on explain new page is displayed as shown in fig75.

MongoDB Compass - localhost:27017/training.sum_agg
Connect Edit View Collection Help

localhost:27017 ... Documents

Explain Plan

Explain provides key execution metrics that help diagnose slow queries and optimize index usage. [Learn more](#)

Visual Tree Raw Output

COLLSCAN
Returned 2 Execution Time 0 ms
Documents Examined: 4

Query Performance Summary

- 2 documents returned
- 4 documents examined
- 0 ms execution time
- Is not sorted in memory
- 0 index keys examined

No index available for this query.

Close

> MONGOSH 12:01 PM 11/26/2023

Fig:75

The number of documents examined to find name are 4 as shown in fig 75. Without adding of Index.

localhost:27017

Documents training.sum_agg

training.sum_agg

4 1
DOCUMENTS INDEXES

Documents Aggregations Schema Indexes Validation

Filter { name: "harshi" } Generate query Explain Reset Find Options

ADD DATA EXPORT DATA

1 - 4 of 4

_id: ObjectId('655d058410a2869aee85fdc2')
name: "harshi"
subject: "english"
marks: 54

_id: ObjectId('655d058410a2869aee85fdc3')
name: "tillu"
subject: "english"
marks: 50

>_MONGOSH

```
> db.sum_agg.createIndex({ "name":1 });
< name_1
> db.sum_agg.getIndexes();
< [
  { v: 2, key: { _id: 1 }, name: '_id_' },
  { v: 2, key: { name: 1 }, name: 'name_1' }
]
training>
```

12:08 PM 11/26/2023

Fig:76

Go to mongosh shell.

Syntax to create a database:

```
db.collection_name.createIndex({field:value});
```

```
ex:db.sum_agg.createIndex({"name":1});
```

Syntax view created index:

```
db.collection_name.getIndexes();
```

```
ex:db.sum_agg.getIndexes();
```

Index is created to collection sum_agg as shown in fig76.

Explain Plan

Explain provides key execution metrics that help diagnose slow queries and optimize index usage. [Learn more](#)

Visual Tree Raw Output

FETCH Returned 2 Execution Time 0 ms

IXSCAN Returned 2 Execution Time 0 ms

Index Name: name_1 Multi Key Index: no

Query Performance Summary

- 2 documents returned
- 2 documents examined
- 6 ms execution time
- Is not sorted in memory
- 2 index keys examined

Query used the following index:
name ↑

Close

Fig:77

Examined documents to find name document are reduced to 2 as shown in fig77.

Creation of Index in mongosh: Compound Index.

The screenshot shows the MongoDB Compass interface. In the top right, it says "4 DOCUMENTS" and "2 INDEXES". The "Indexes" tab is selected. Below that, there's a "Filter" dropdown set to "{name: 'nithin', section: 'a'}. The "Find" button is highlighted in green. The bottom half of the screen is a terminal window titled "MONGOSH" showing the command: `> db.student_data.createIndex({name:1,section:1});`. The terminal also displays the results of the query and the creation of the index.

Fig:78

use database “training”collection “student_data”.

Syntax compound field:

`db.collection_name.createIndex({field_name:value,field_name2:value.....});`

ex:`db.student_data.createIndex({name:1,section:1});`

The screenshot shows the MongoDB Compass interface with the "Explain Plan" dialog open. The dialog title is "Explain Plan" and it says "Explain provides key execution metrics that help diagnose slow queries and optimize index usage." There are two tabs: "Visual Tree" (selected) and "Raw Output". The "Visual Tree" tab shows a tree structure with nodes: "IXSCAN" (Returned 1, Execution Time 0 ms, Index Name: name_1_section_1, Multi Key Index: no) and "FETCH" (Returned 1, Execution Time 0 ms). To the right is a "Query Performance Summary" panel with the following details: 1 documents returned, 1 documents examined, 0 ms execution time, Is not sorted in memory, 1 index keys examined. It also lists the query used the following index: "name ↑" and "section ↑". At the bottom right of the dialog is a "Close" button.

Fig:79

The examined document to find document are reduced to 1 as shown in fig79.

Creating a multikey indexes: used to create an index for array values.

The screenshot shows the MongoDB Compass interface. The left sidebar lists databases and collections, with 'unwind_marks' selected. The main area displays the 'Documents' tab for the 'unwind_marks' collection. Two documents are listed:

```
_id: ObjectId('655d10de10a2869aee85fdd4')
name: "nithin"
subjects: Array (2)
  0: "maths"
  1: "english"

_id: ObjectId('655d10de10a2869aee85fdd5')
name: "sona"
subjects: Array (2)
  0: "maths"
  1: "english"
```

At the bottom, a MONGOSH shell shows the command `db.unwind_marks.createIndex({ subjects: 1 })`.

Fig:80

Database:training<<Collection “unwind_marks”.

Consists of array for field “subjects”.

Syntax: db.collection_name.createIndex({field_name:value});

Ex: db.unwind_marks.createIndex({subjects:1});

The screenshot shows the MongoDB Compass interface. The left sidebar lists databases and collections, with 'unwind_marks' selected. The main area displays the 'Documents' tab for the 'unwind_marks' collection. A query filter is applied: `{subjects : "maths", name:"nithin"}`. At the bottom, a MONGOSH shell shows the command `db.unwind_marks.createIndex({ subjects: 1 })` being run.

```
name: 'nithin',
class: 10,
section: 'a'
}
> db.unwind_marks.createIndex({ "subjects":1 });
< subjects_1
> db.unwind_marks.getIndexes();
< [
  { v: 2, key: { _id: 1 }, name: '_id_' },
  { v: 2, key: { subjects: 1 }, name: 'subjects_1' }
]
> db.unwind_marks.find()
< {
  _id: ObjectId("655d10de10a2869aee85fdd4"),
  name: 'nithin',
  subjects: [
```

Fig:81

Drop an Index in Mongosh:

Syntax: db.collection_name.dropIndex({indexke:1});

The screenshot shows the MongoDB Compass interface. The left sidebar lists databases and collections, including 'REVA' and 'training'. The main area shows the 'Indexes' tab for the 'unwind_marks' collection. There are two indexes listed:

Name and Definition	Type	Size	Usage	Properties
id	REGULAR	20.5 KB	5 (since Sun Nov 26 2023)	UNIQUE
subjects_1	REGULAR	20.5 KB	2 (since Sun Nov 26 2023)	

The MONGOSH terminal at the bottom shows the command:

```
> db.unwind_marks.dropIndex({"subjects":1})
```

Fig:82

Ex:db.unwind_marks.drop({“subjects”:1})

The screenshot shows the MongoDB Compass interface. The left sidebar lists databases and collections, including 'REVA' and 'training'. The main area shows the 'Indexes' tab for the 'unwind_marks' collection. Only one index remains:

Name and Definition	Type	Size	Usage	Properties
id	REGULAR	20.5 KB	5 (since Sun Nov 26 2023)	UNIQUE

The MONGOSH terminal at the bottom shows the command and its execution result:

```
> db.unwind_marks.dropIndex({"subjects":1})
< { nIndexesWas: 2, ok: 1 }
>
```

Fig:83

Index is deleted for collection unwind_marks.

Pymongo

pymongo is a Python distribution that contains tools for interacting with [MongoDB](#) database from Python.

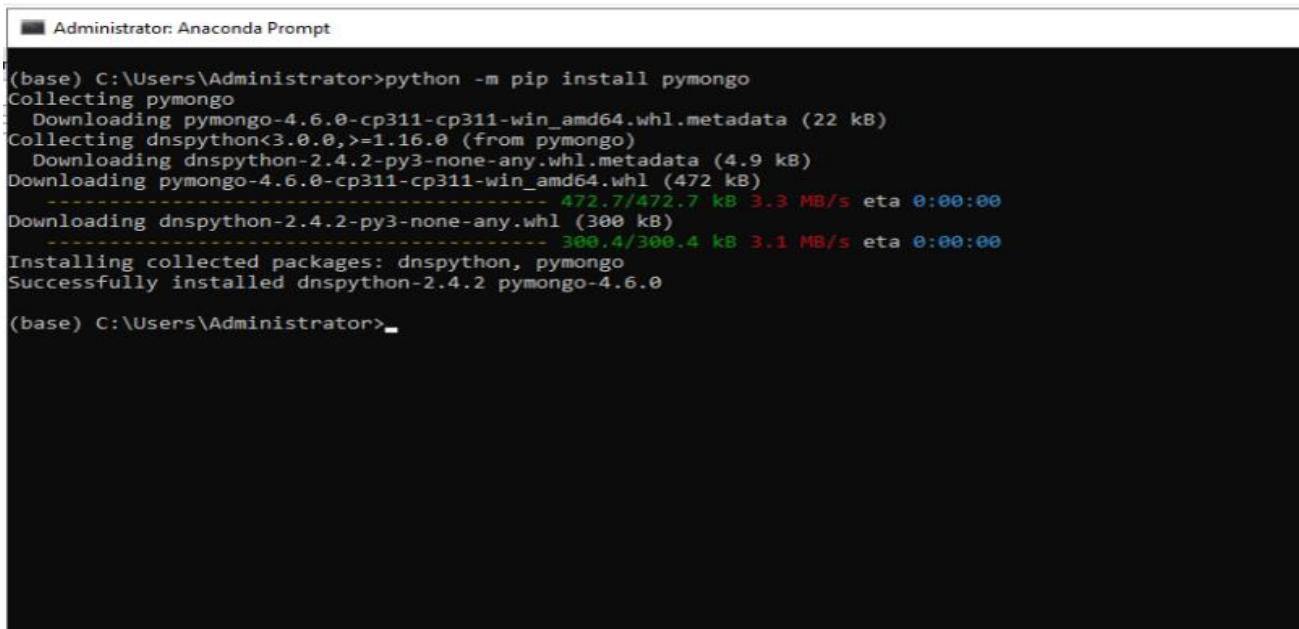
It is the official MongoDB driver for synchronous Python applications.

PyMongo, the standard [MongoDB driver library for Python](#), is easy to use and offers an intuitive API for accessing databases, collections, and documents. Objects retrieved from MongoDB through PyMongo are compatible with dictionaries and lists, so we can easily manipulate, iterate, and print them.

Pymongo Installation

- Python needs a MongoDB driver to access the MongoDB database.
 - here we will use the MongoDB driver "PyMongo".
 - We will you use PIP to install "PyMongo" in Anaconda prompt.
 - PIP is most likely already installed in your Python environment.
 - Navigate your command line to the location of PIP, as shown in the following picture.
- **Command to install pymongo in anaconda prompt.**

Python -m pip install pymongo.



```
(base) C:\Users\Administrator>python -m pip install pymongo
Collecting pymongo
  Downloading pymongo-4.6.0-cp311-cp311-win_amd64.whl.metadata (22 kB)
Collecting dnspython<3.0.0,>=1.16.0 (from pymongo)
  Downloading dnspython-2.4.2-py3-none-any.whl.metadata (4.9 kB)
  Downloading pymongo-4.6.0-cp311-cp311-win_amd64.whl (472 kB)
    472.7/472.7 kB 3.3 MB/s eta 0:00:00
  Downloading dnspython-2.4.2-py3-none-any.whl (300 kB)
    300.4/300.4 kB 3.1 MB/s eta 0:00:00
Installing collected packages: dnspython, pymongo
Successfully installed dnspython-2.4.2 pymongo-4.6.0

(base) C:\Users\Administrator>
```

- Installation of pymongo in anaconda prompt.

Python MongoDB Create Database

To create database in pymongo, first we need create a new folder in your desktop, then we should create a new text document and the must be save with extension of **.py** while saving the file in folder as shown in the pictures below.

To create a database in mongodb by using pymongo, start by creating a mongo client object, then specify a connection URL with the correct Ip address and the name of the database you want to create. MongoDB will create the new database if it does not exist and make a connection to it.

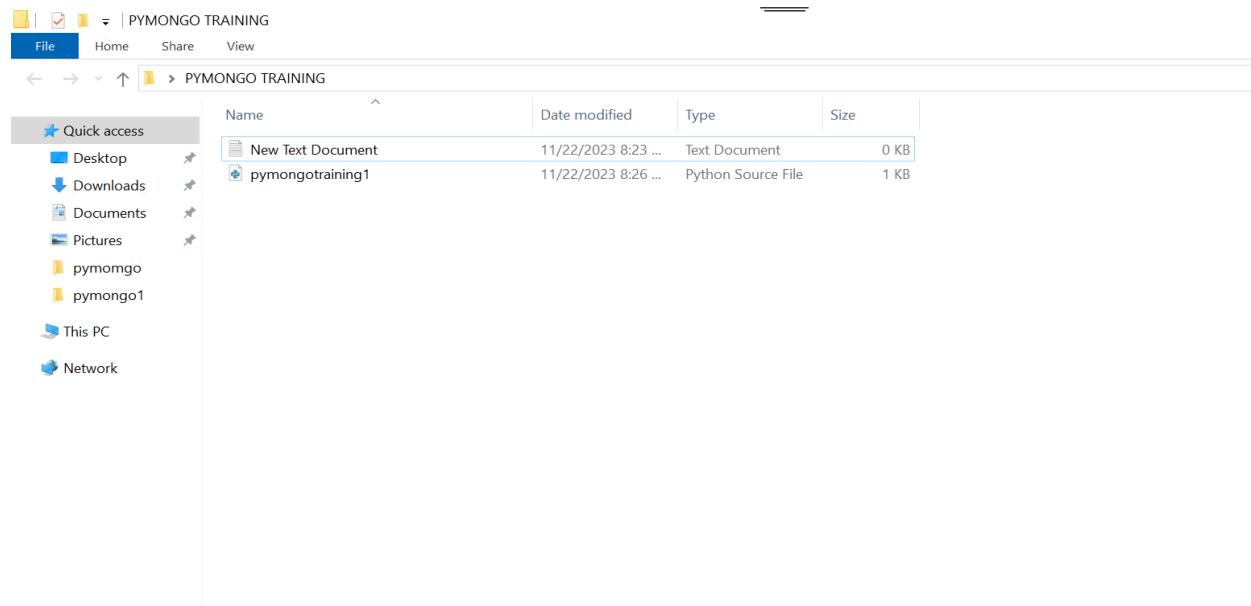
- The basic Syntax to write code in a created file.

Import pymongo

```
myclient = pymongo.MongoClient("mongodb://localhost:27017/")

mydb = myclient["newdatabasename"]
rec = [
    give the input
]
Information.insert_many(rec)
```

The above syntax must be written in a created file to create a database in pymongo tool.



- A folder is created.

Example:

Step 1:

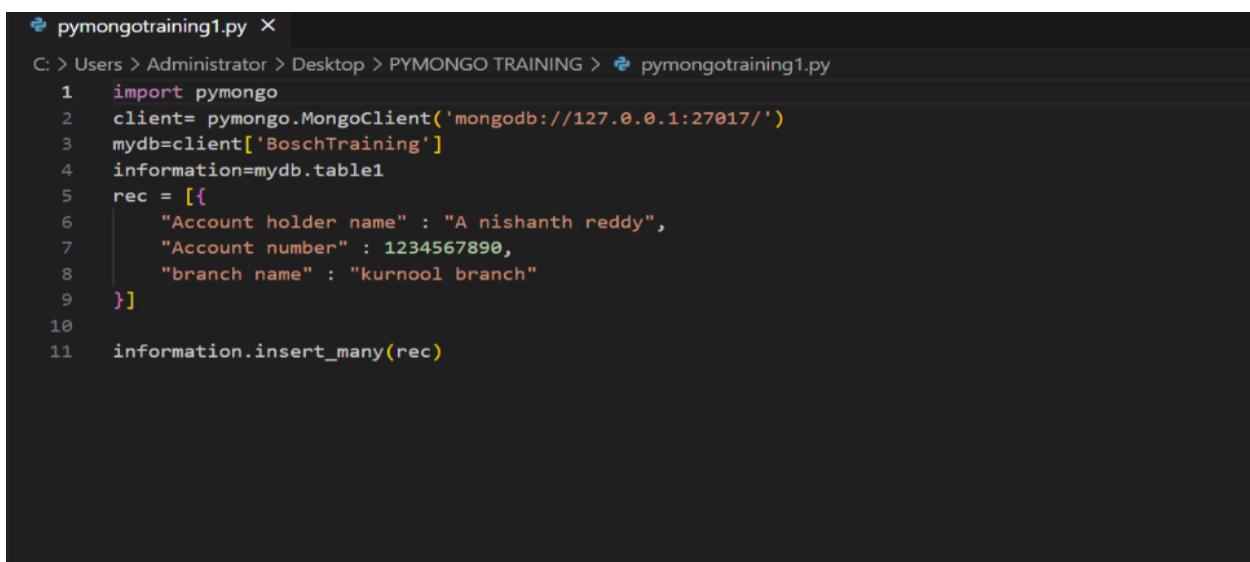
Here I have created a new folder in the name of **PYMONGOTRAINING** in that folder. I have created a new file like **pymongotraining1** text document and then I had written code to create database called **“BOSCH TRAINING”**.

Create a pymongotraining1.py

Code:

```
Import pymongo  
  
client= pymongo.MongoClient('mongodb://127.0.0.1:27017/')  
  
mydb=client['Bosch Training']  
  
information=mydb.table1  
  
rec = [{  
    "Account holder name" : "A Nishanth Reddy",  
    "Account number" : 1234567890,  
    "branch name" : "Kurnool branch"  
}]  
  
Information.insert_many(rec)
```

The above code must be saved in the file with the extension of .py as we told before then it automatically saves in python description. like shown below picture.

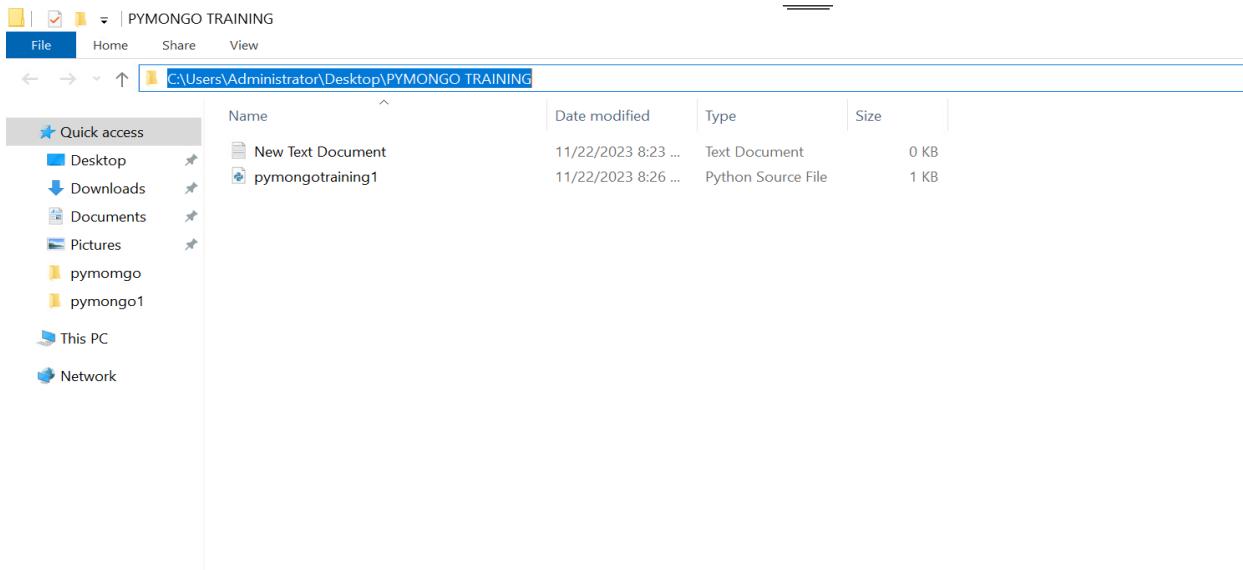


A screenshot of a terminal window titled "pymongotraining1.py". The window shows the Python code for inserting a document into a MongoDB database. The code imports pymongo, connects to the local MongoDB instance at port 27017, selects the 'Bosch Training' database, and inserts a single document into the 'table1' collection. The document contains three fields: 'Account holder name', 'Account number', and 'branch name'. The code is numbered from 1 to 11.

```
C: > Users > Administrator > Desktop > PYMONGO TRAINING > pymongotraining1.py  
1 import pymongo  
2 client= pymongo.MongoClient('mongodb://127.0.0.1:27017/')  
3 mydb=client['BoschTraining']  
4 information=mydb.table1  
5 rec = [{  
6     "Account holder name" : "A nishanth reddy",  
7     "Account number" : 1234567890,  
8     "branch name" : "kurnool branch"  
9 }]  
10  
11 information.insert_many(rec)
```

Step 2:

Write the above code and execute it using anaconda prompt. Here we created a file and a directory, then declared the path to execute the created Pymongo file.



- A file is Created in the folder.

```
(base) C:\Users\Administrator>pip install pymongo
Requirement already satisfied: pymongo in c:\programdata\anaconda3\lib\site-packages (4.6.0)
Requirement already satisfied: dnsPYTHON<3.0.0,>=1.16.0 in c:\programdata\anaconda3\lib\site-packages (from pymongo) (2.4.2)

(base) C:\Users\Administrator>cd C:\Users\Administrator\Desktop\PYMONGO TRAINING

(base) C:\Users\Administrator\Desktop\PYMONGO TRAINING>
```

The screenshot shows a terminal window titled 'Administrator: Anaconda Prompt'. It displays the command 'pip install pymongo' being run, which outputs that the requirement is already satisfied. Then, the command 'cd C:\Users\Administrator\Desktop\PYMONGO TRAINING' is run to change the current working directory to the specified folder.

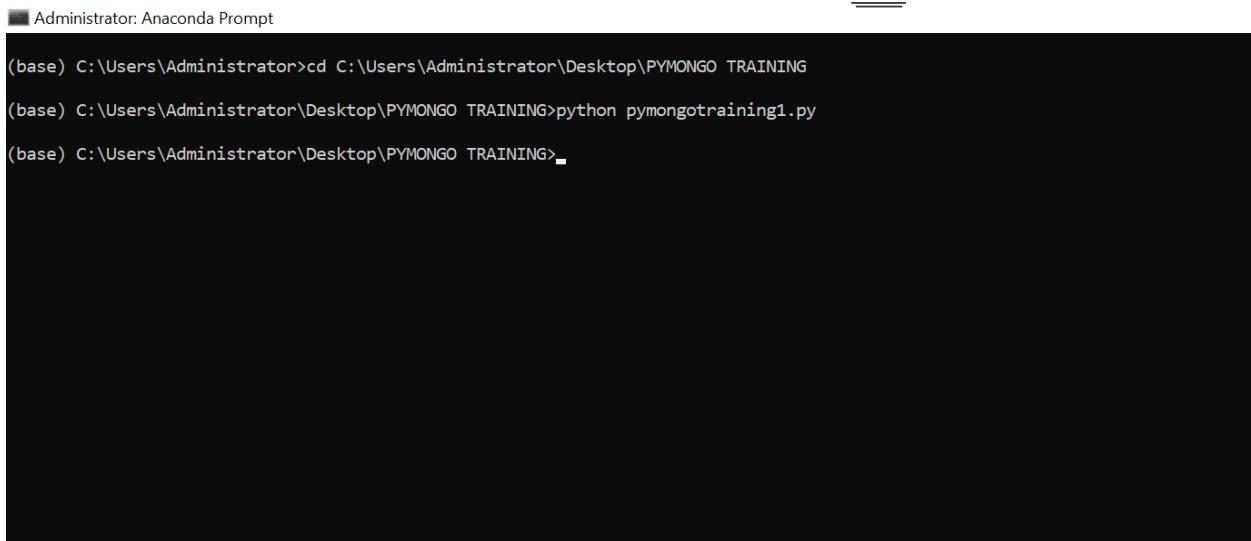
- Changed to Desktop directory form user.

Step 3:

Execute the file with the anaconda prompt.

- Syntax to execute a file in pymongo.

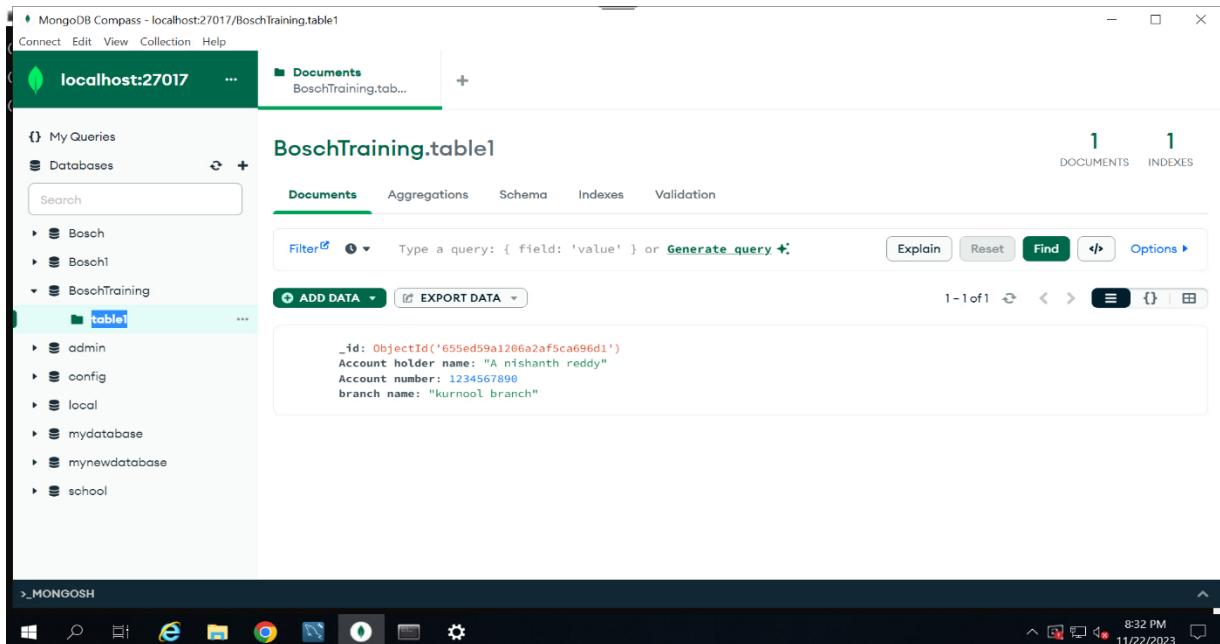
Python Filename.py



```
(base) C:\Users\Administrator>cd C:\Users\Administrator\Desktop\PYTHON TRAINING
(base) C:\Users\Administrator\Desktop\PYTHON TRAINING>python pymongotraining1.py
(base) C:\Users\Administrator\Desktop\PYTHON TRAINING>
```

Output:

Open the Mongo Compass and see if the ‘Bosch Training’ is being created.



MongoDB Compass - localhost:27017/BoschTraining.table1

localhost:27017

Documents

BoschTraining.table1

1 DOCUMENTS 1 INDEXES

Documents Aggregations Schema Indexes Validation

Filter Type a query: { field: 'value' } or Generate query + Explain Reset Find Options

Add DATA Export DATA 1-1 of 1

`_id: ObjectId('655ed59a1206a2af5ca696d1')
Account holder name: "A nishanth reddy"
Account number: 1234567890
branch name: "kurnool branch"`