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

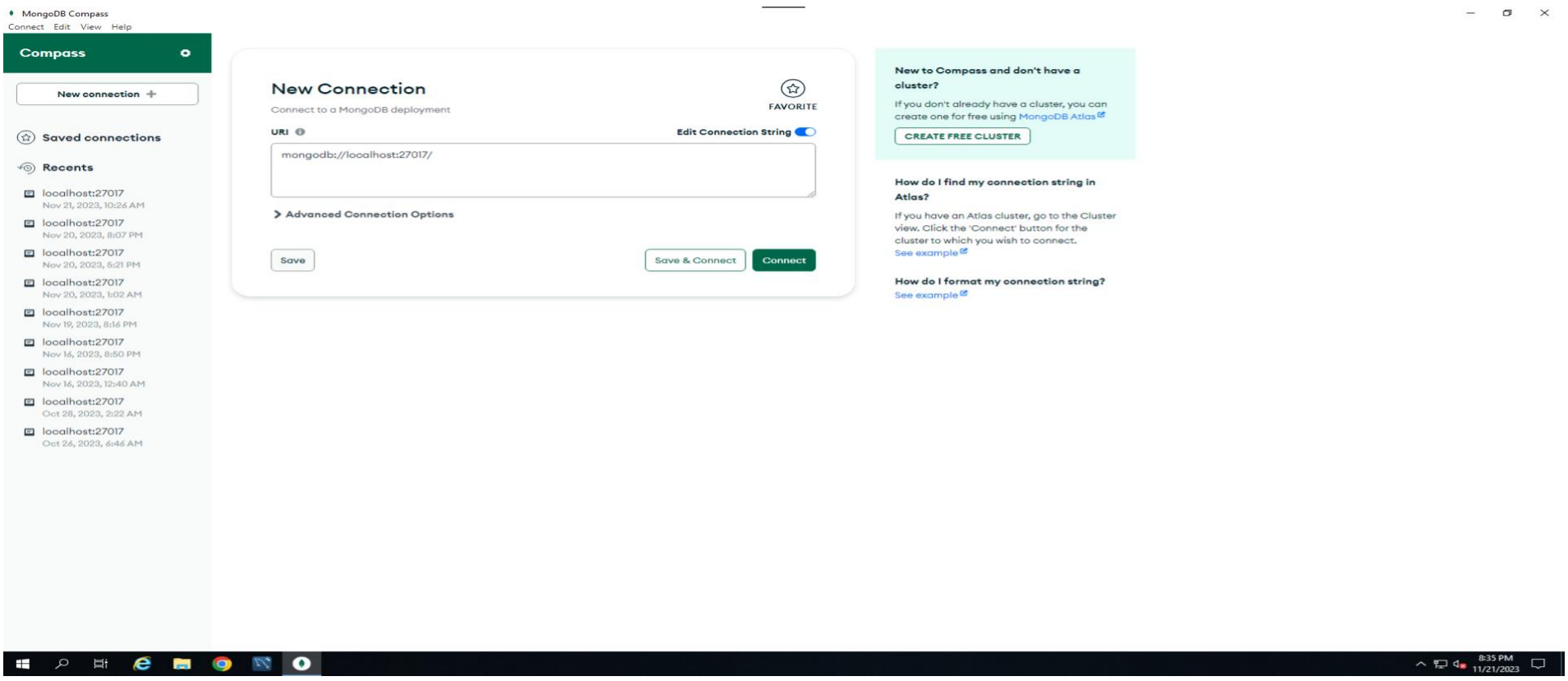


Fig:1

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:

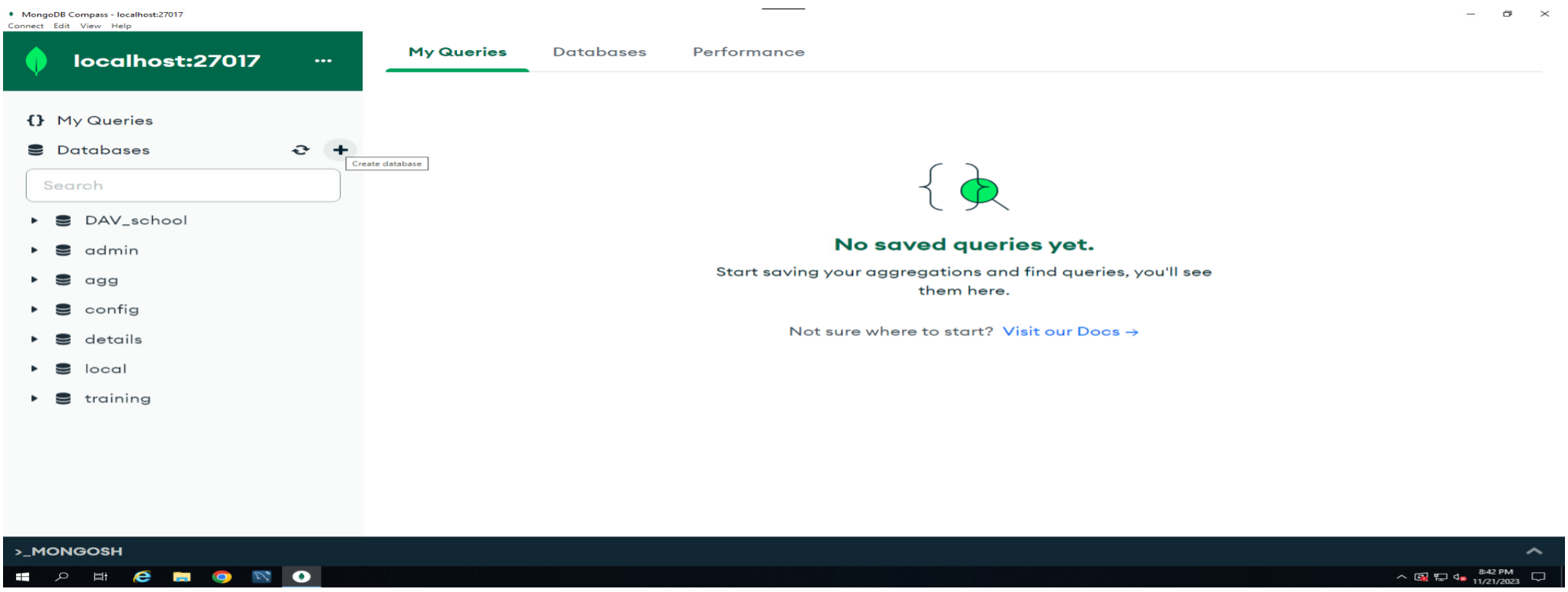


Fig:2

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

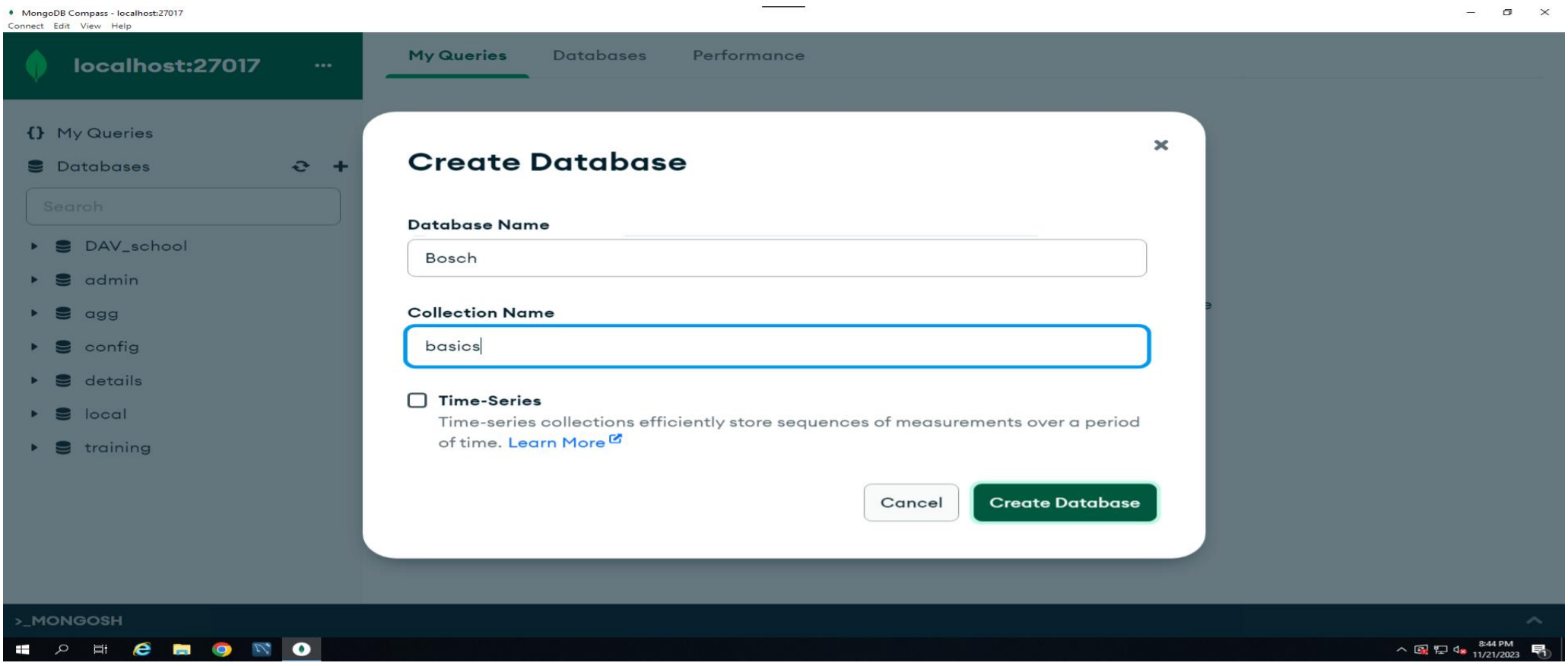


Fig:3

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

After entering required information enter “Create Database”.

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

Database creation in MongoDB SHELL :

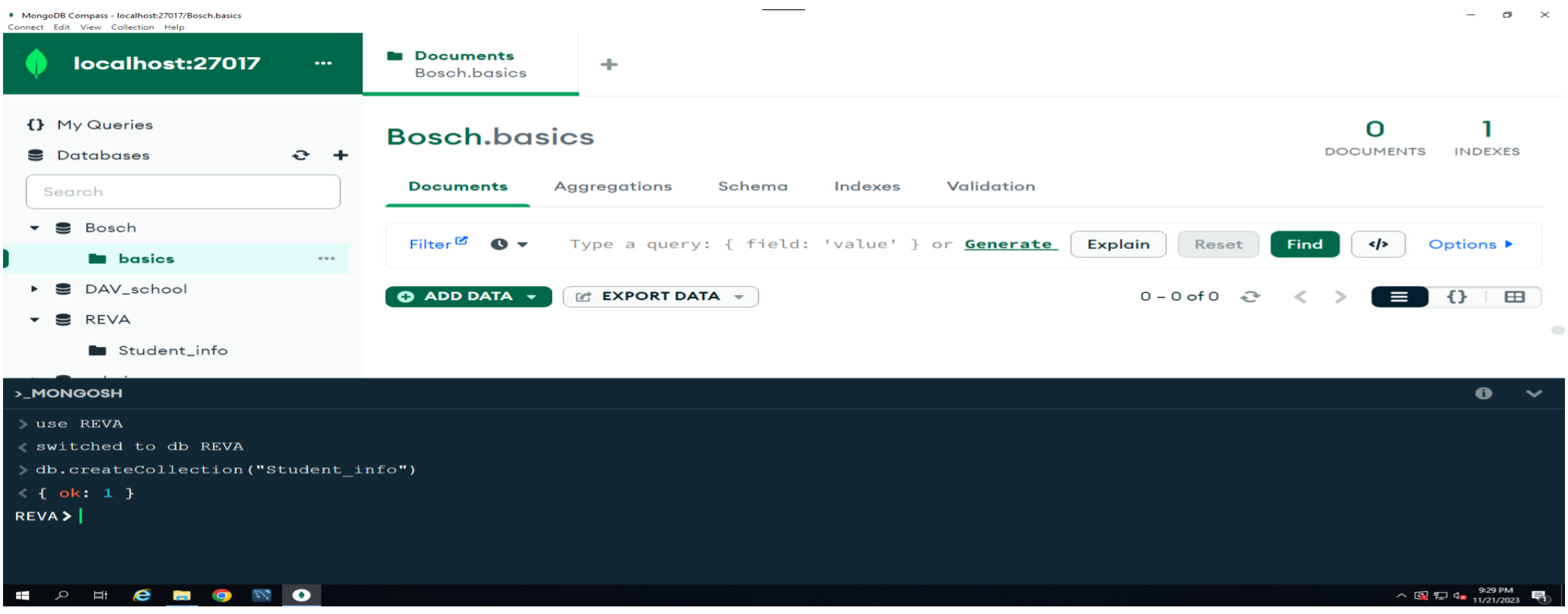


Fig:4

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:

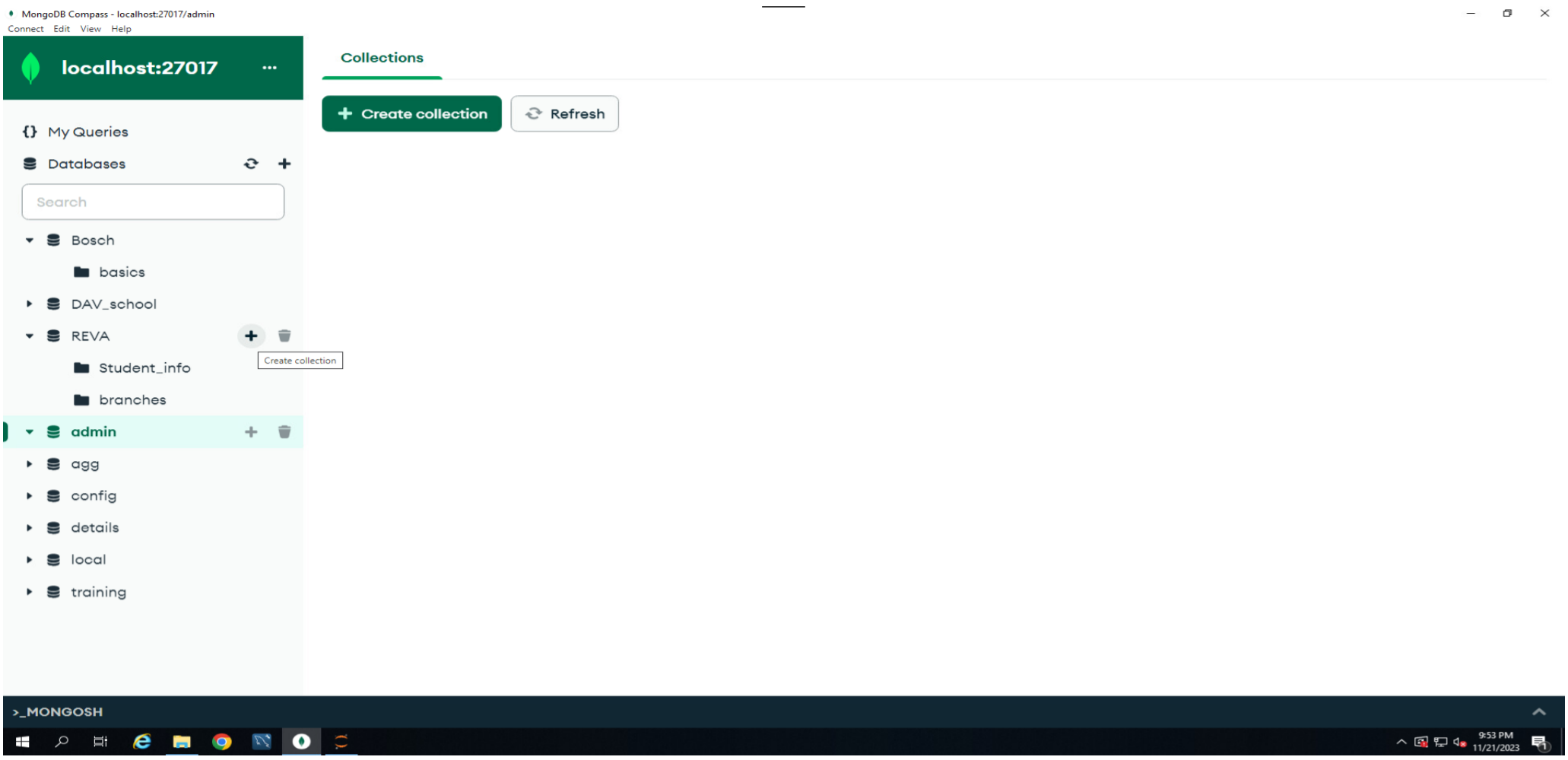


Fig:5

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.

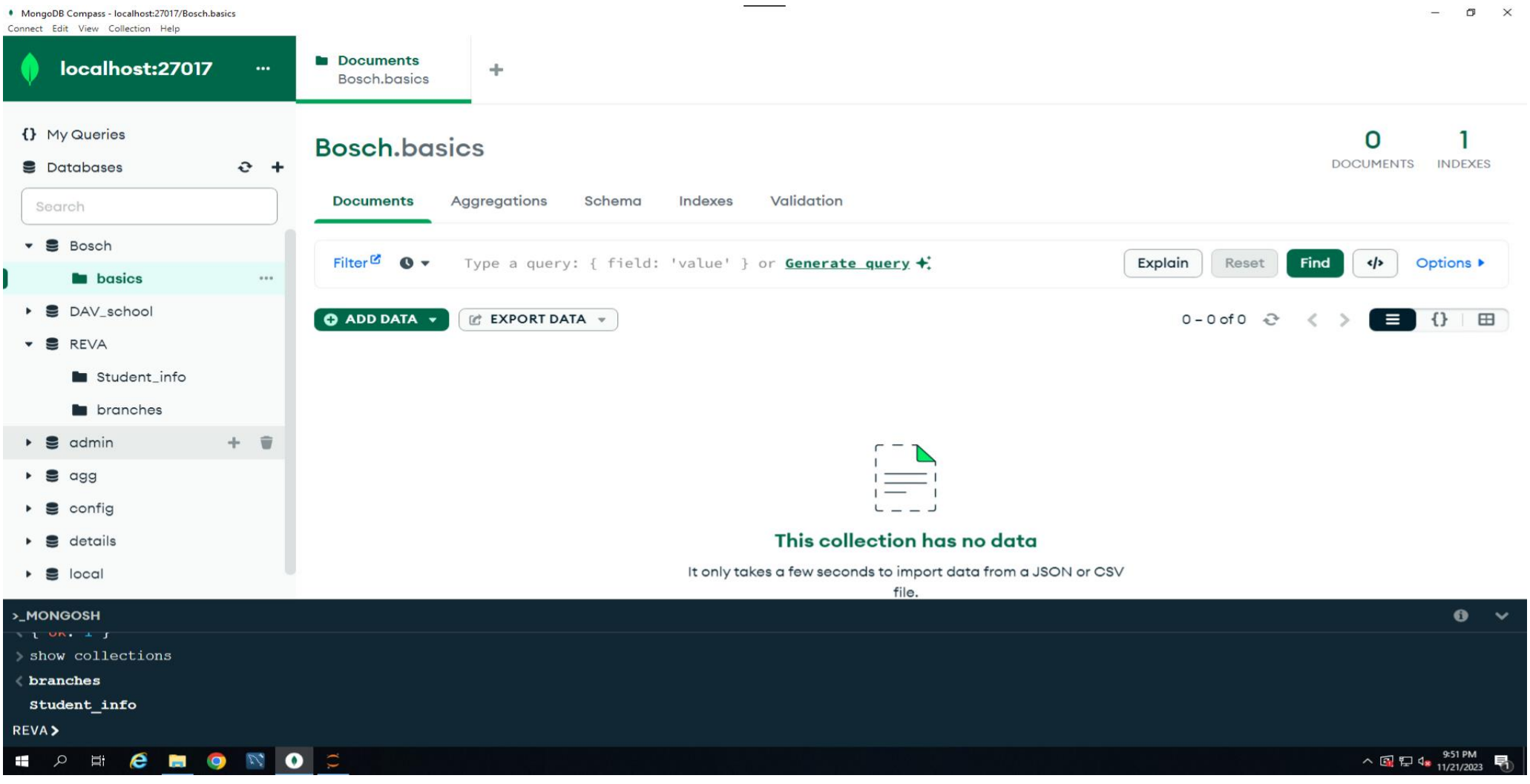


Fig:6

“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 :

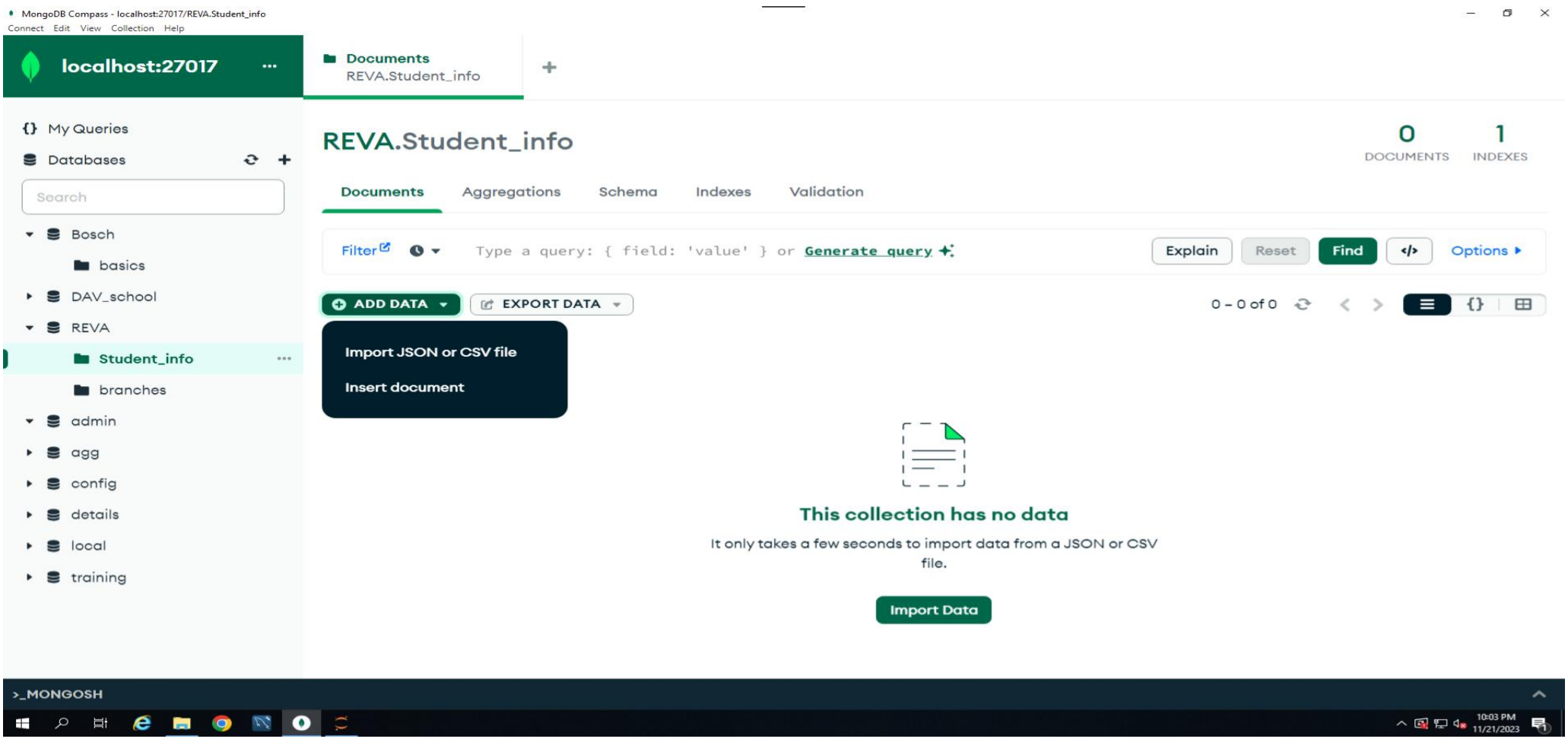


Fig:7

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.

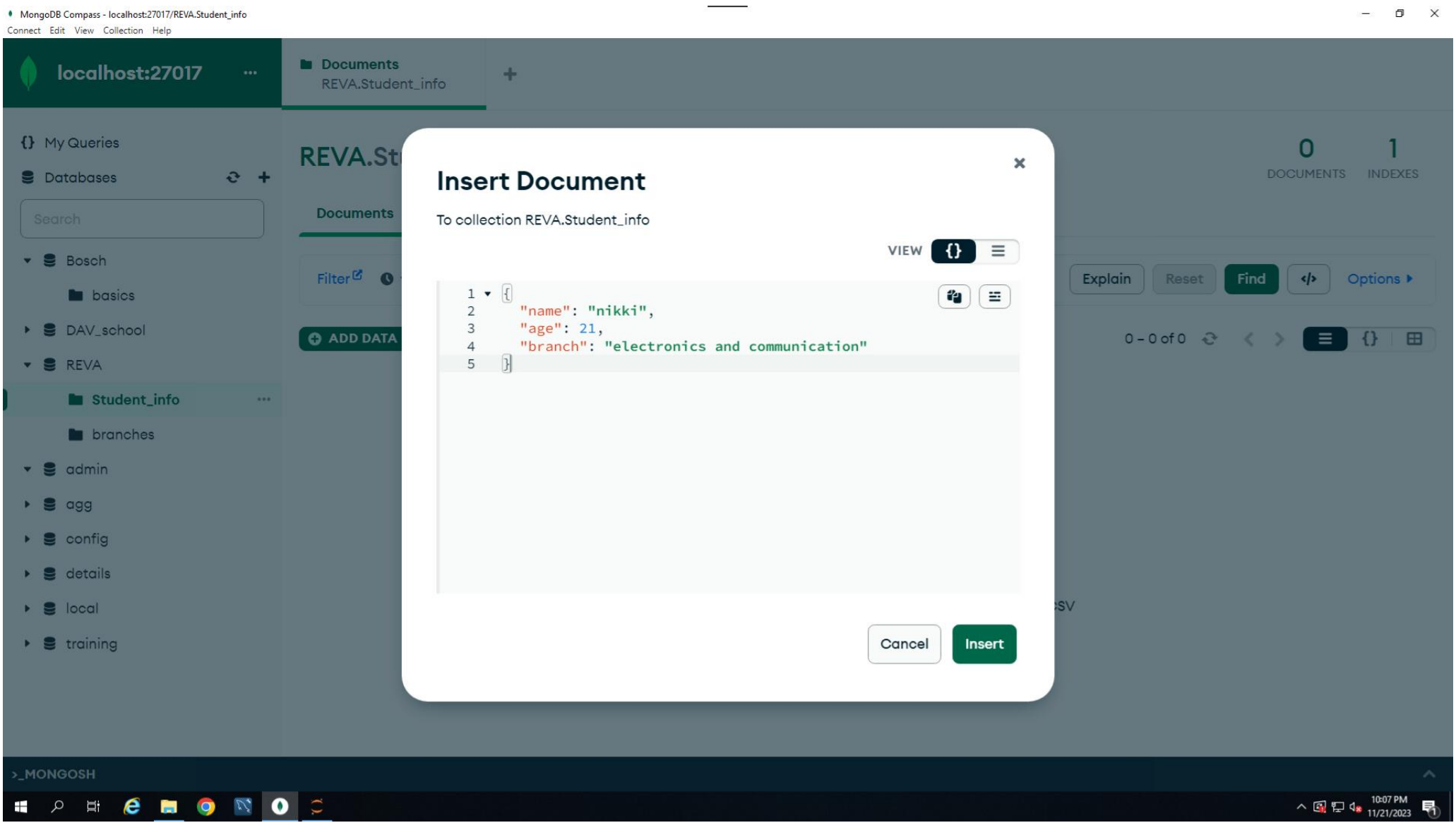


Fig:8

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:

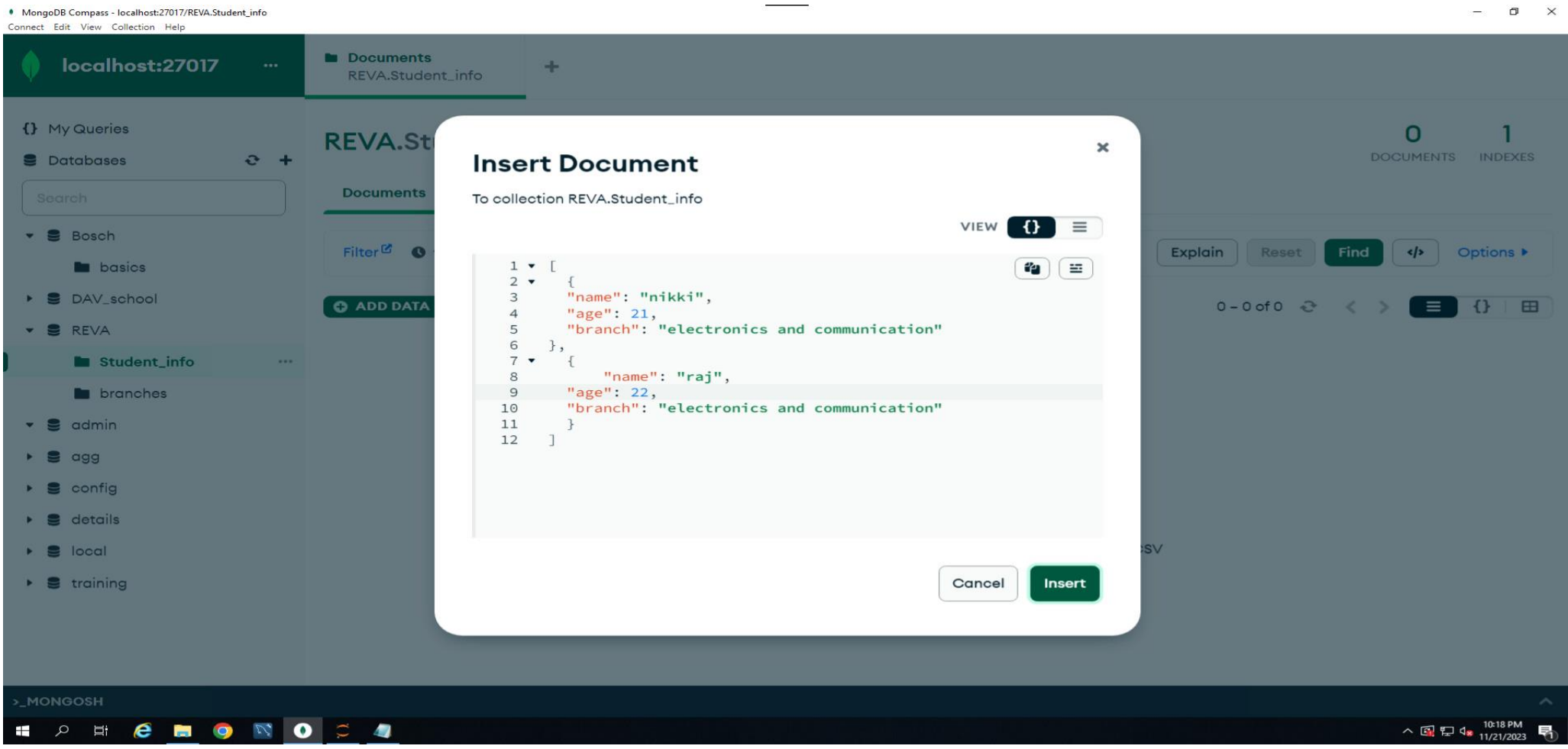


Fig:9

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:

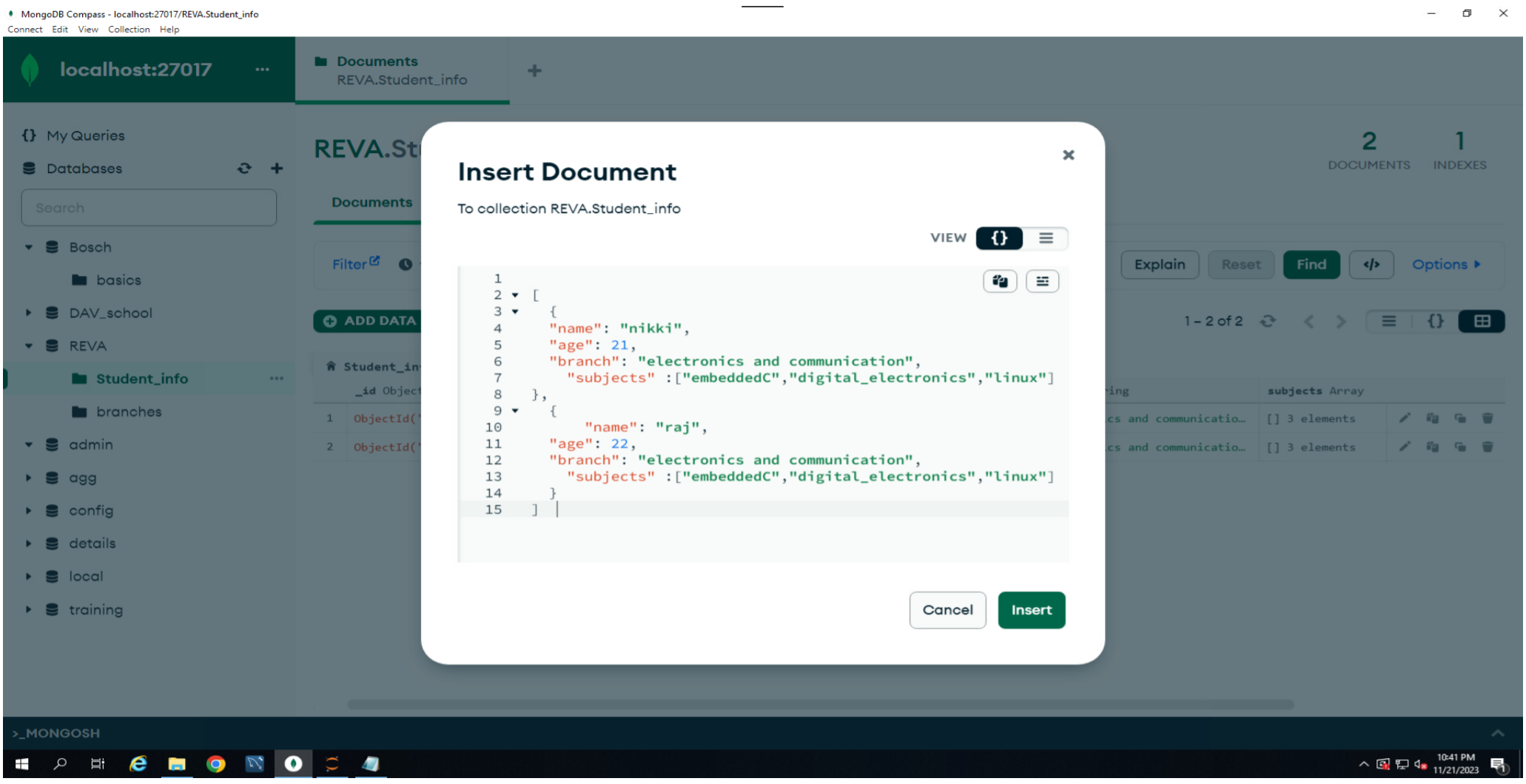


Fig:10

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 :

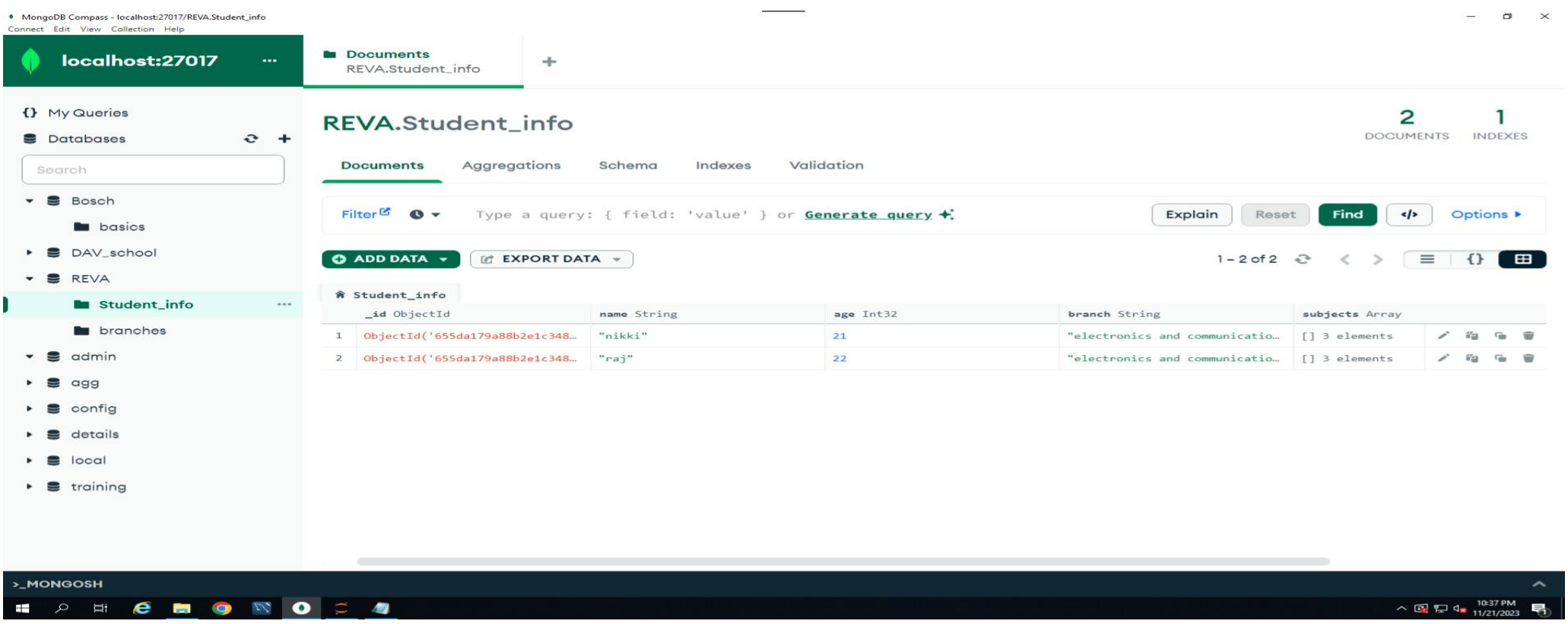


Fig:11

JSON view :

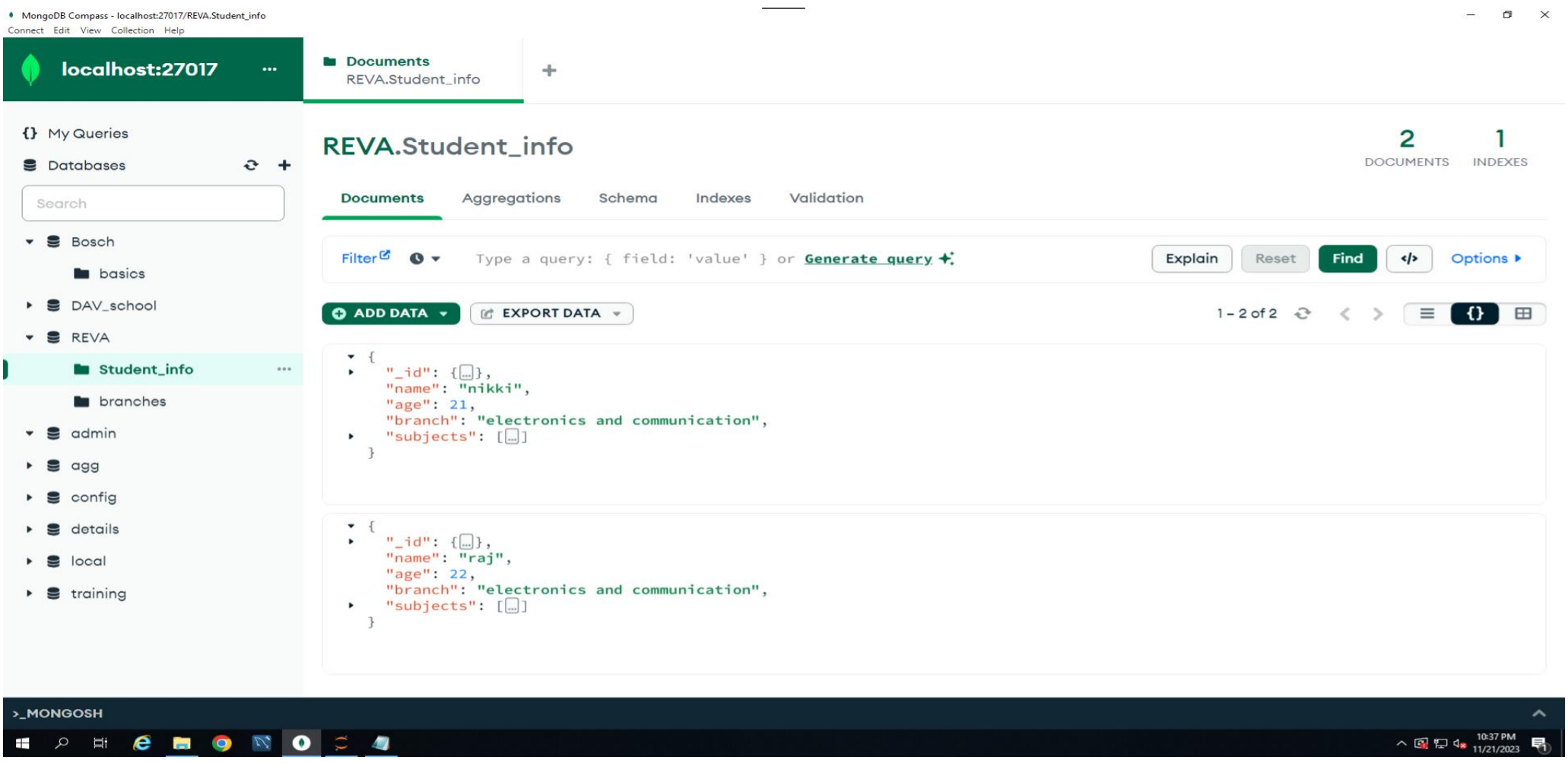


Fig:12

List view

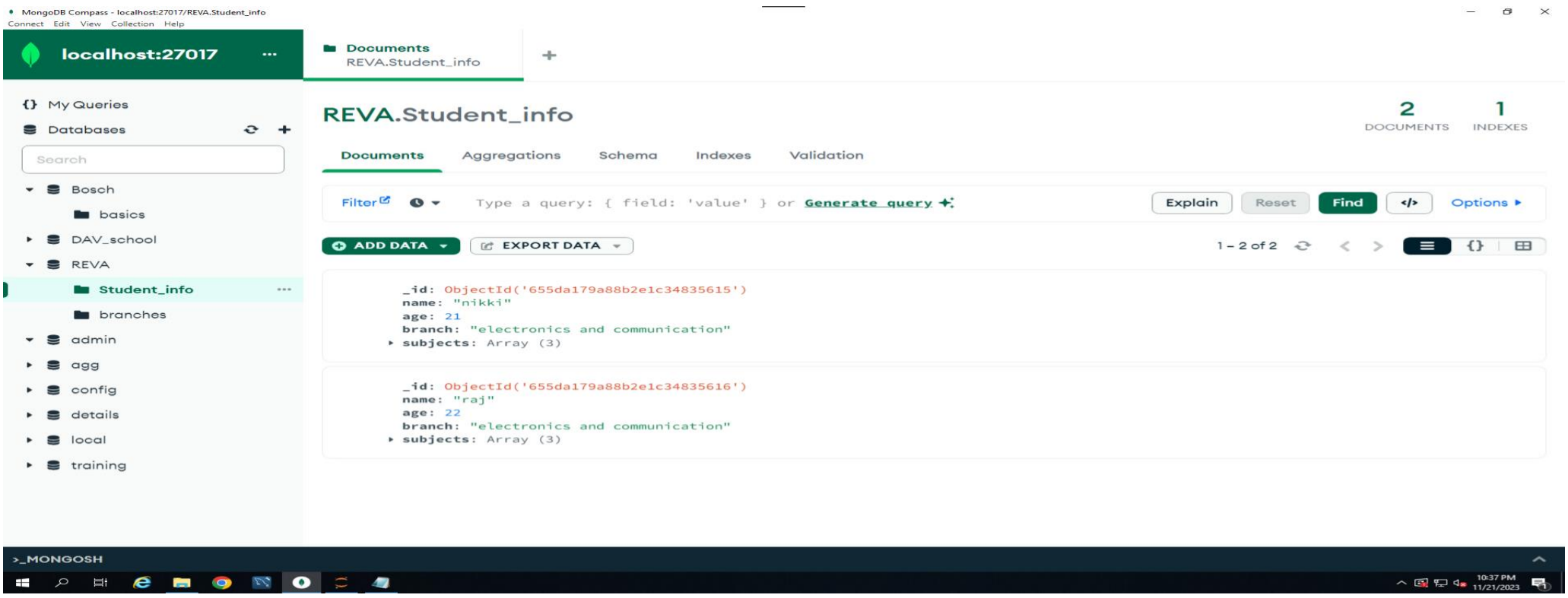


Fig:13

Insert dcoument 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(): :

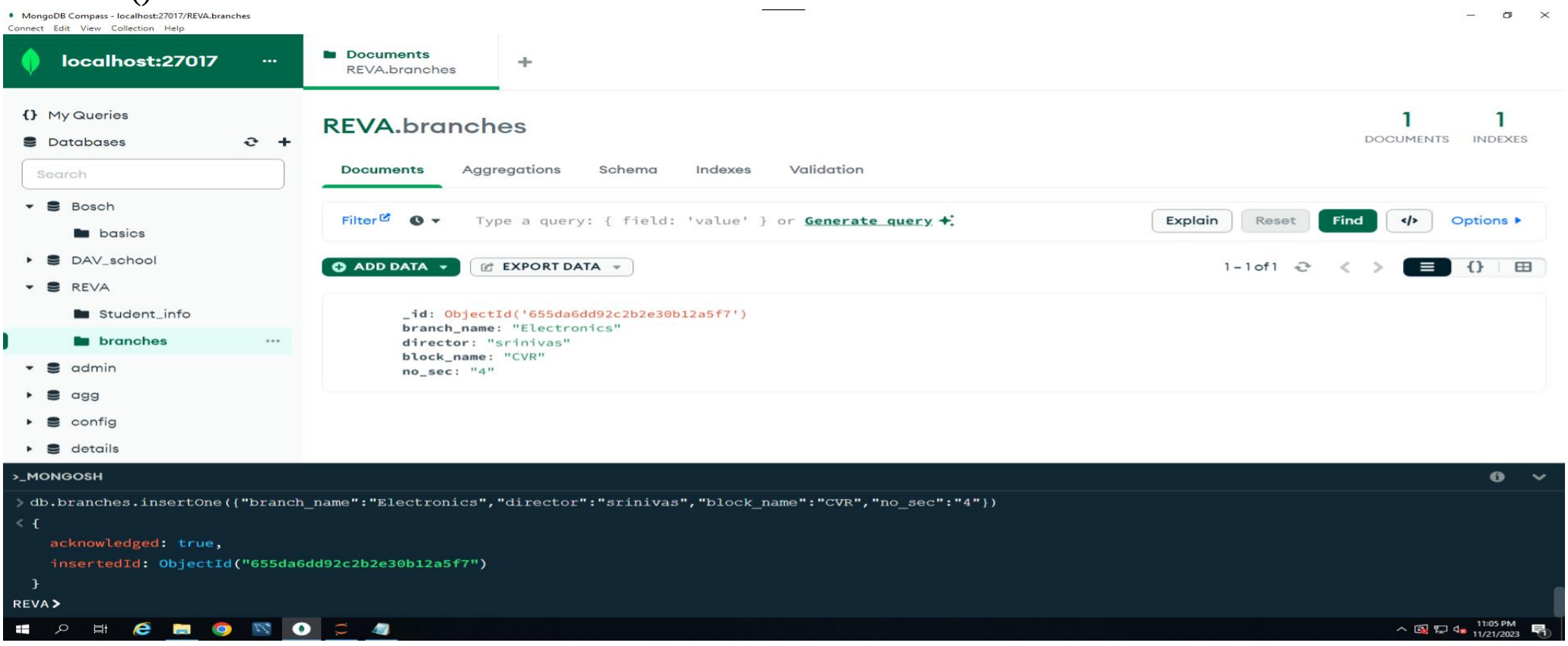


Fig:14

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 :

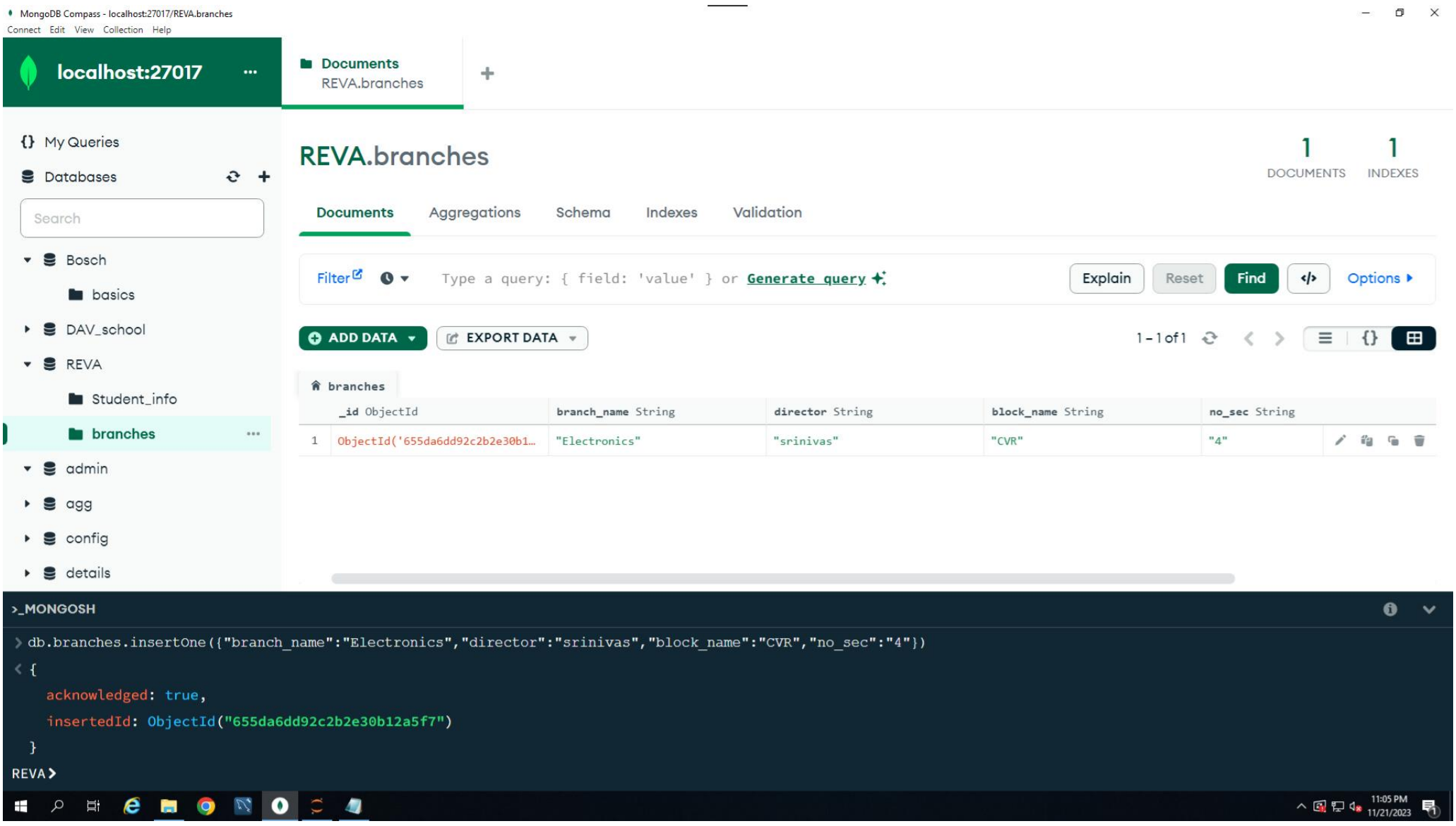


Fig:15

insertMany():

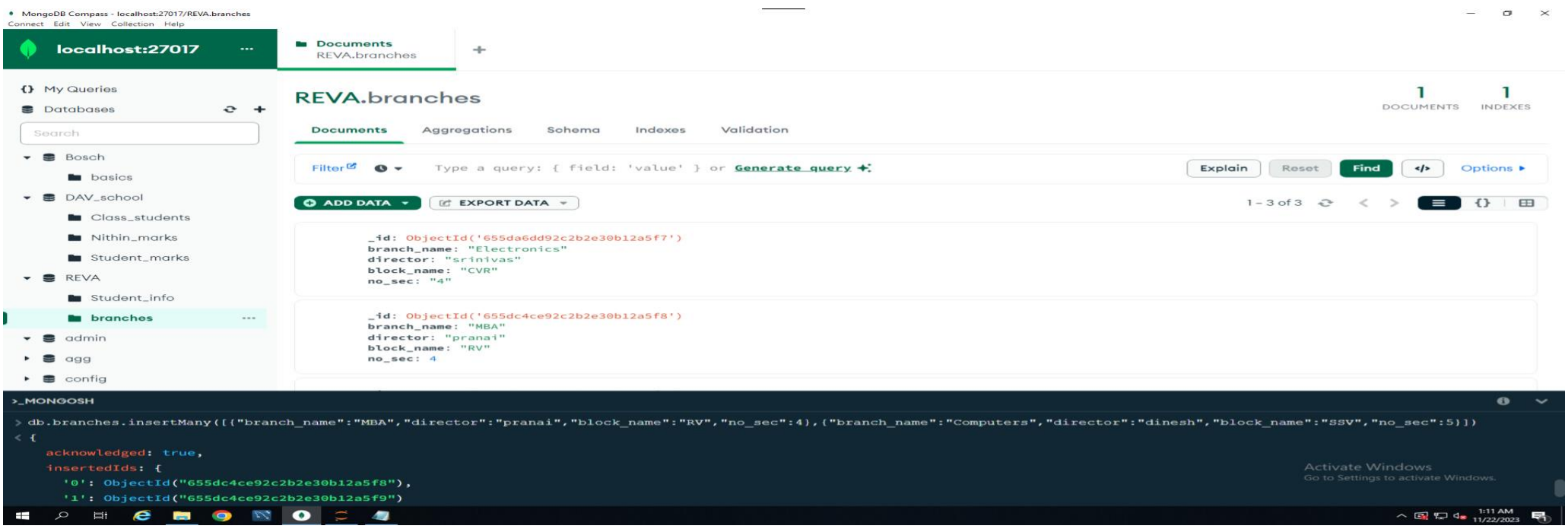


Fig:16

Syntax:

use<database_name>

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

O/P:

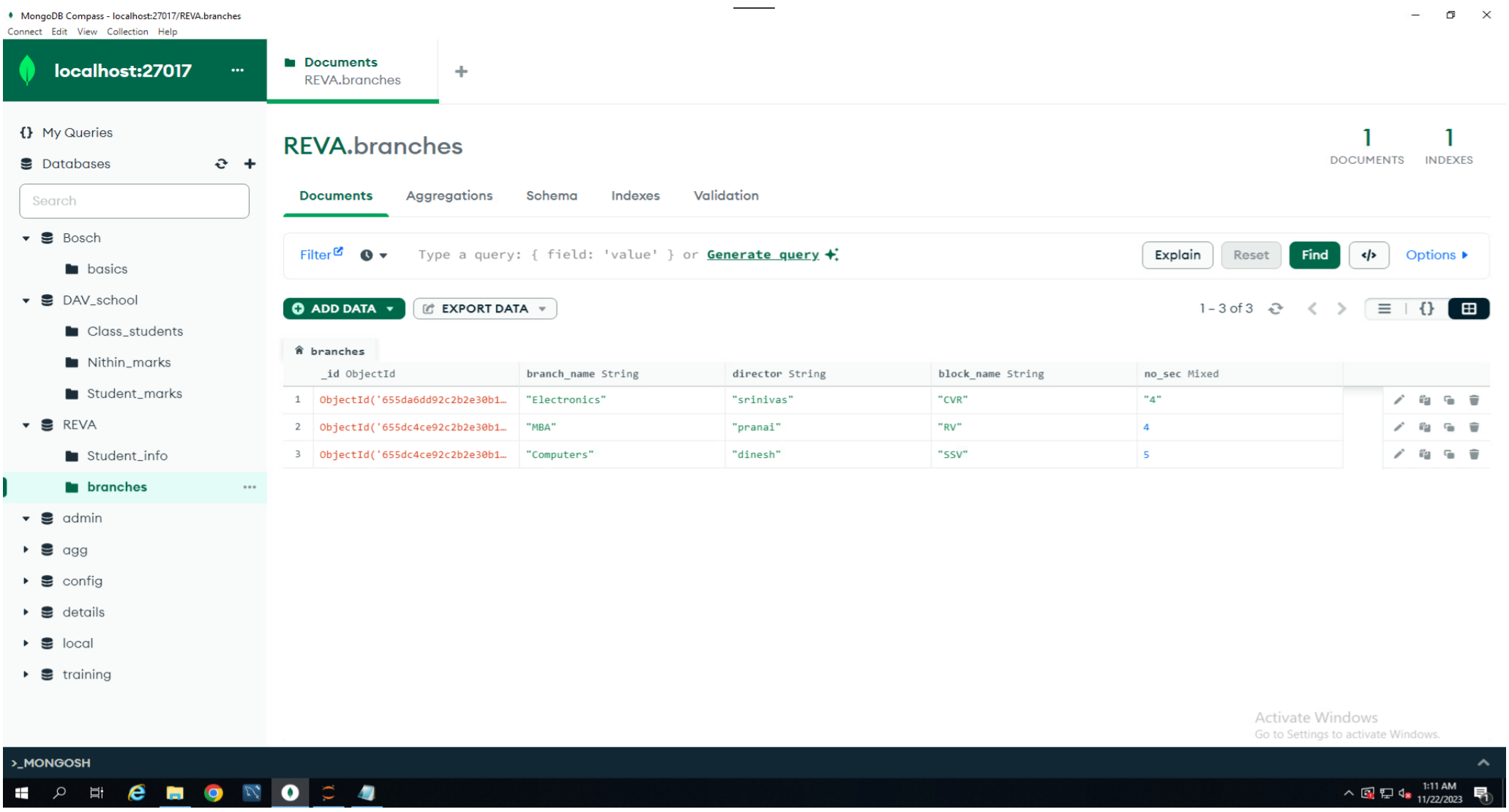


Fig:17

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 ():

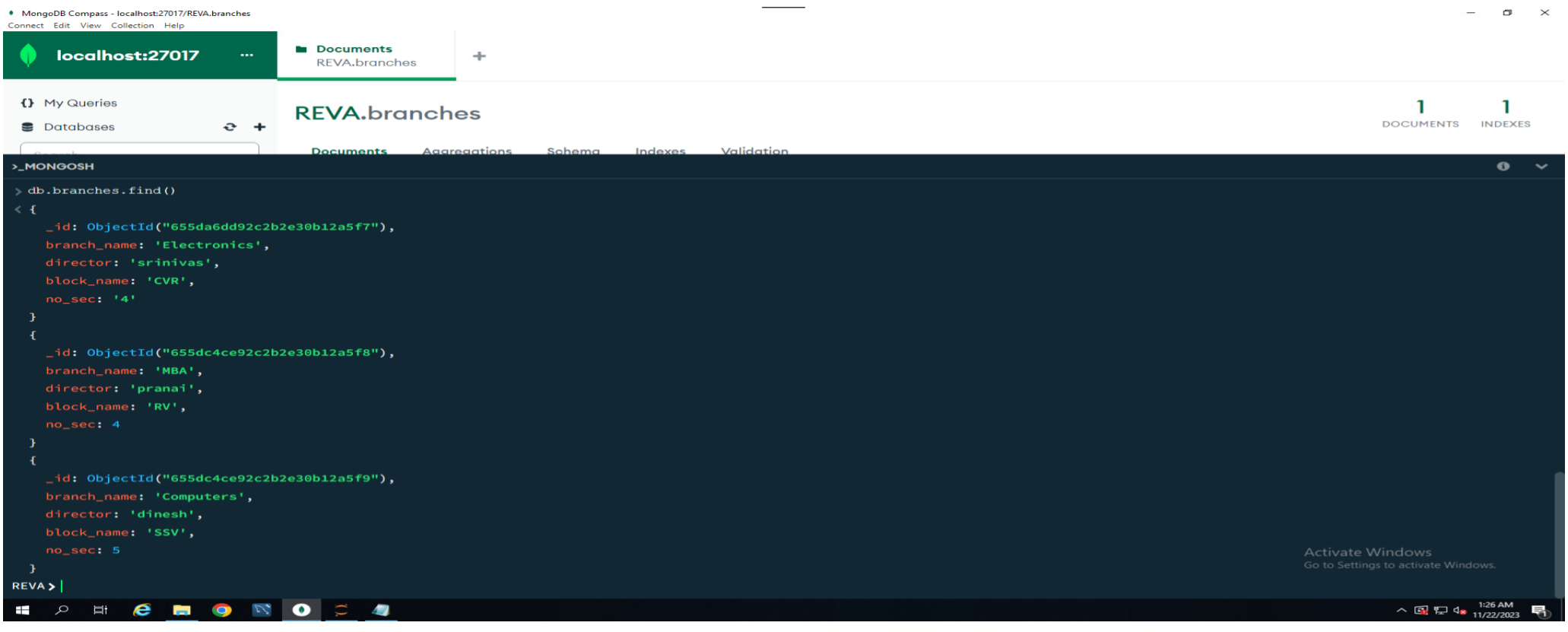


Fig:18

Syntax:

db.collection_name.find()

ex:db.branches.find()

findOne():

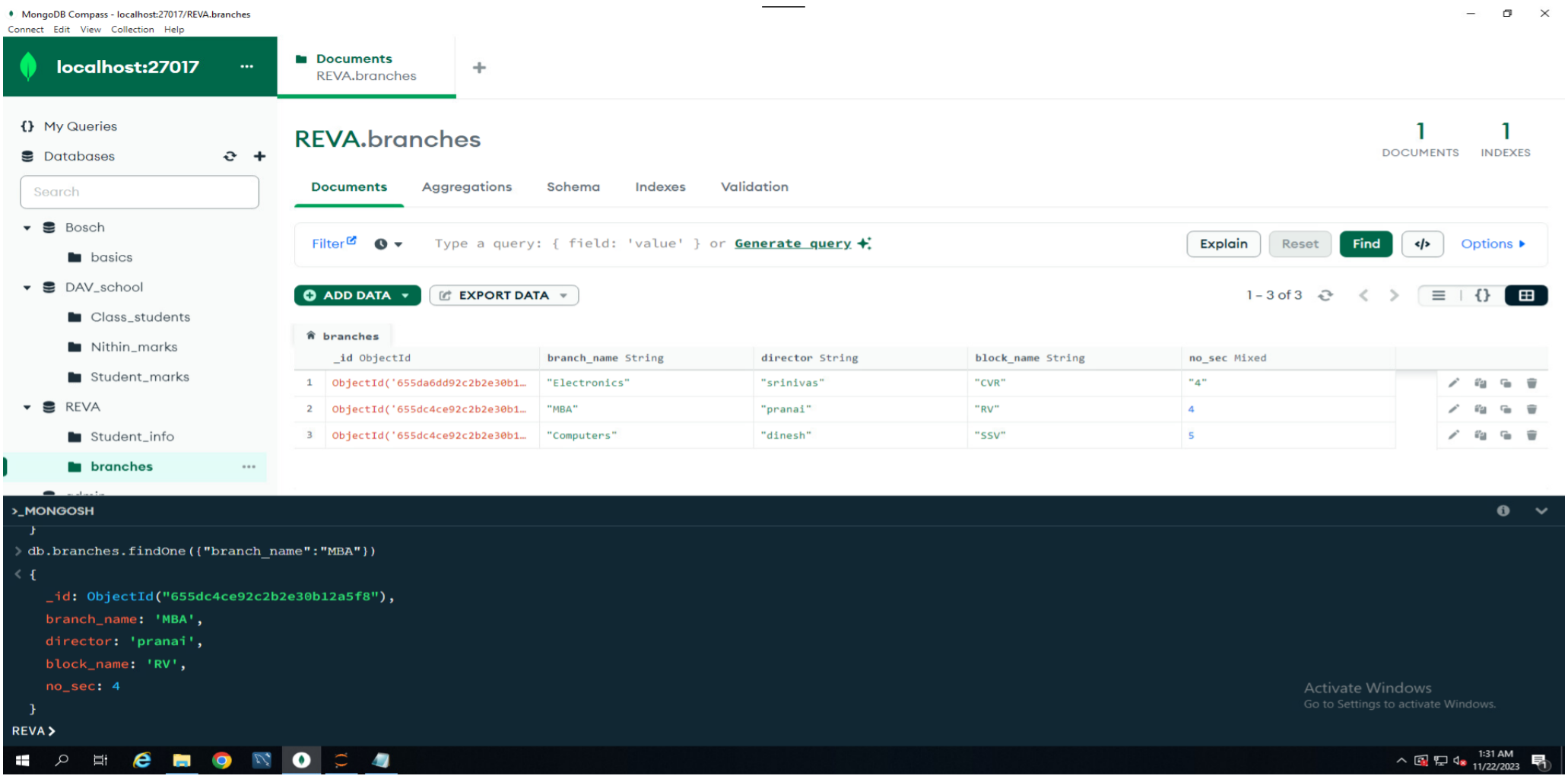


Fig:19

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)

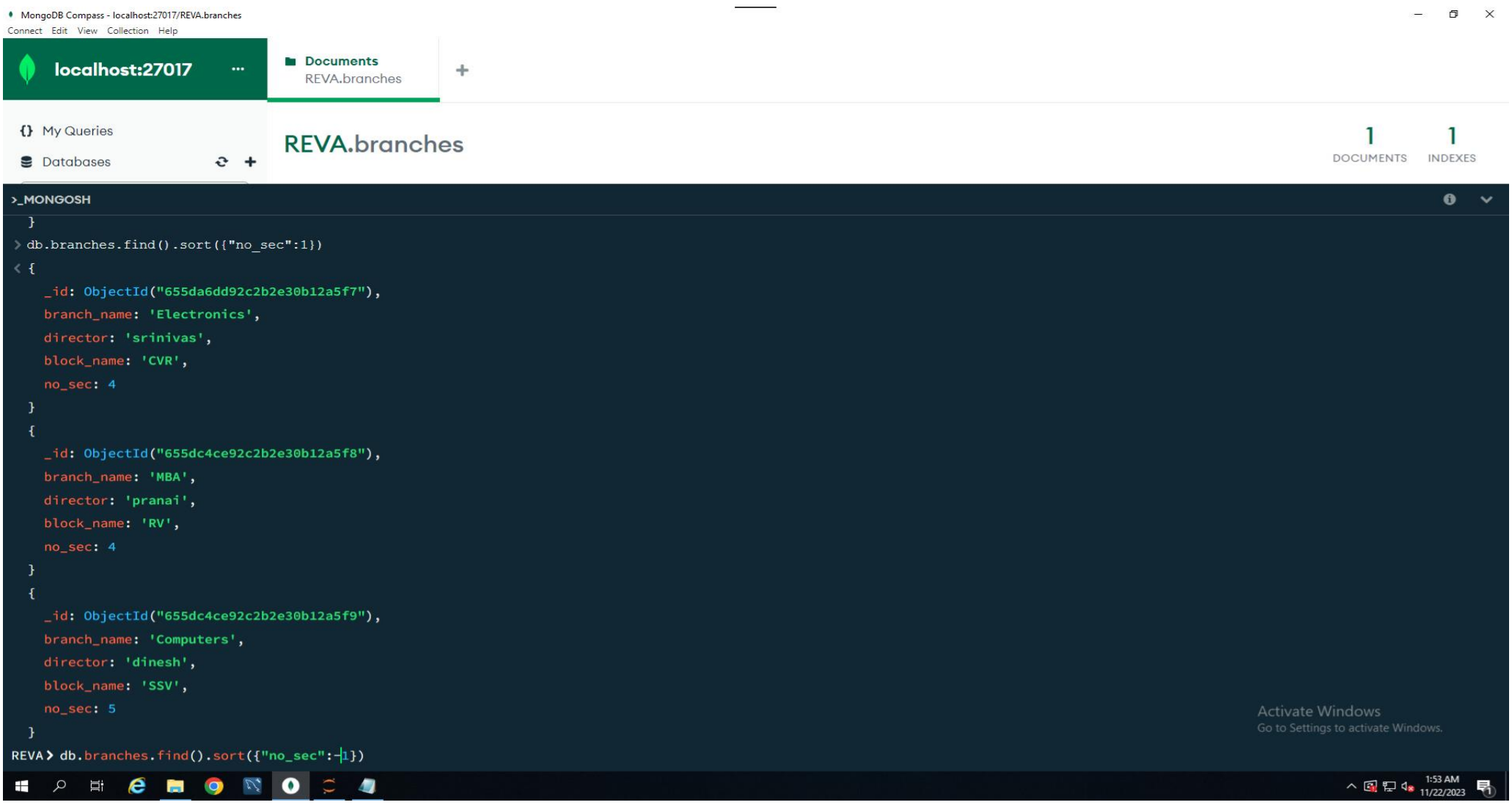


Fig:20

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

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

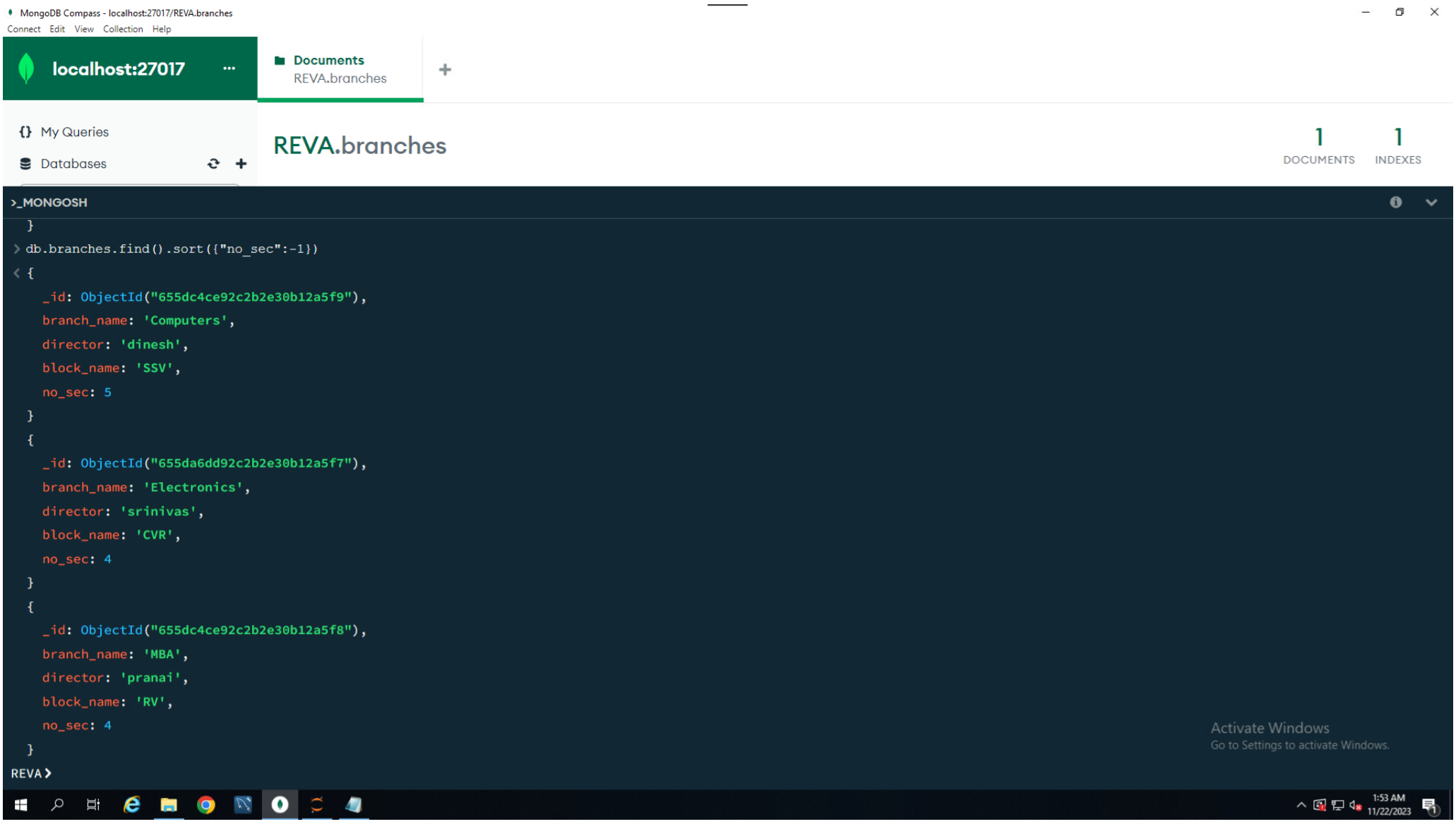


Fig:21

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:

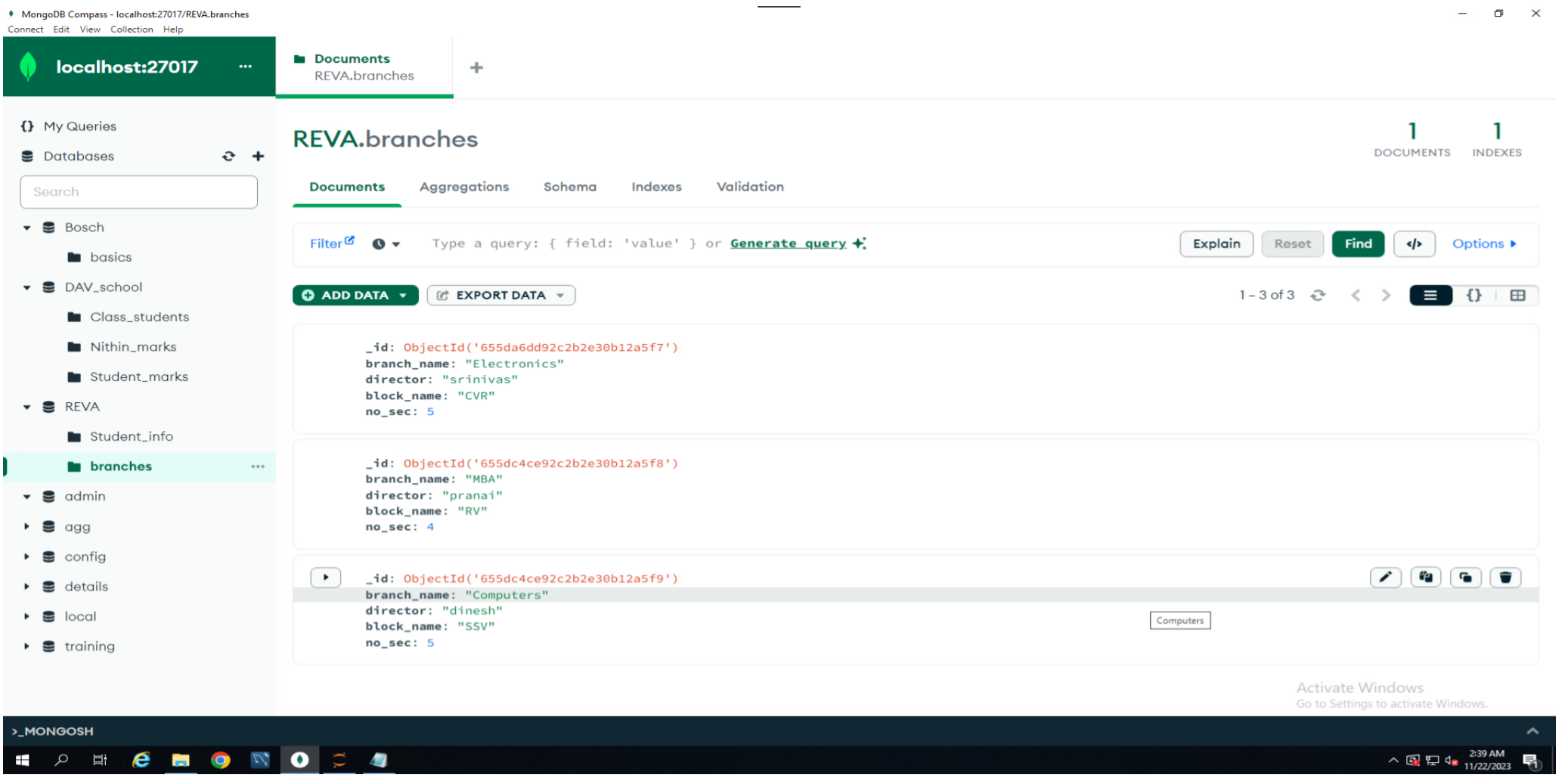


Fig:22

Steps:

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

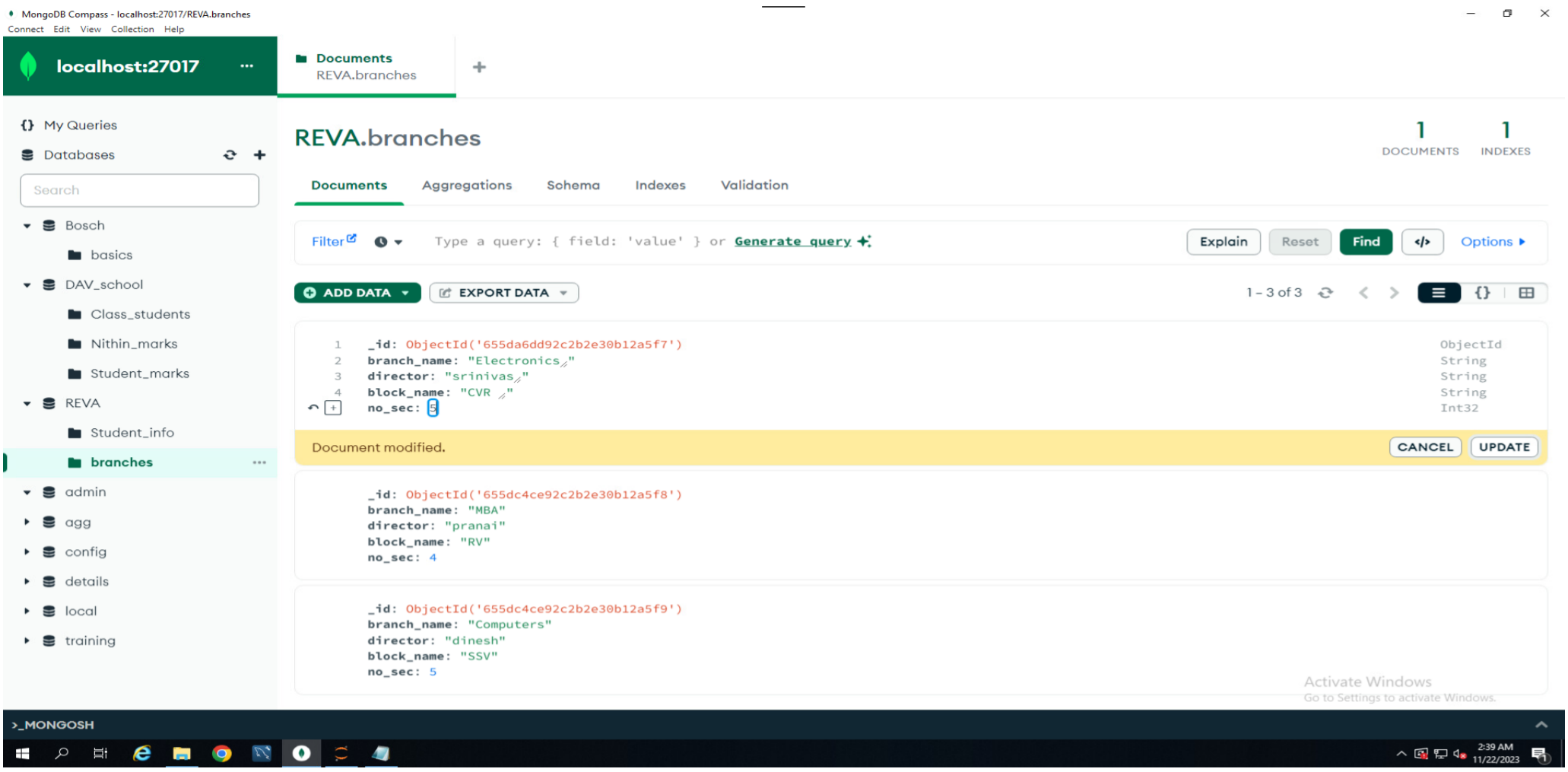


Fig:23

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

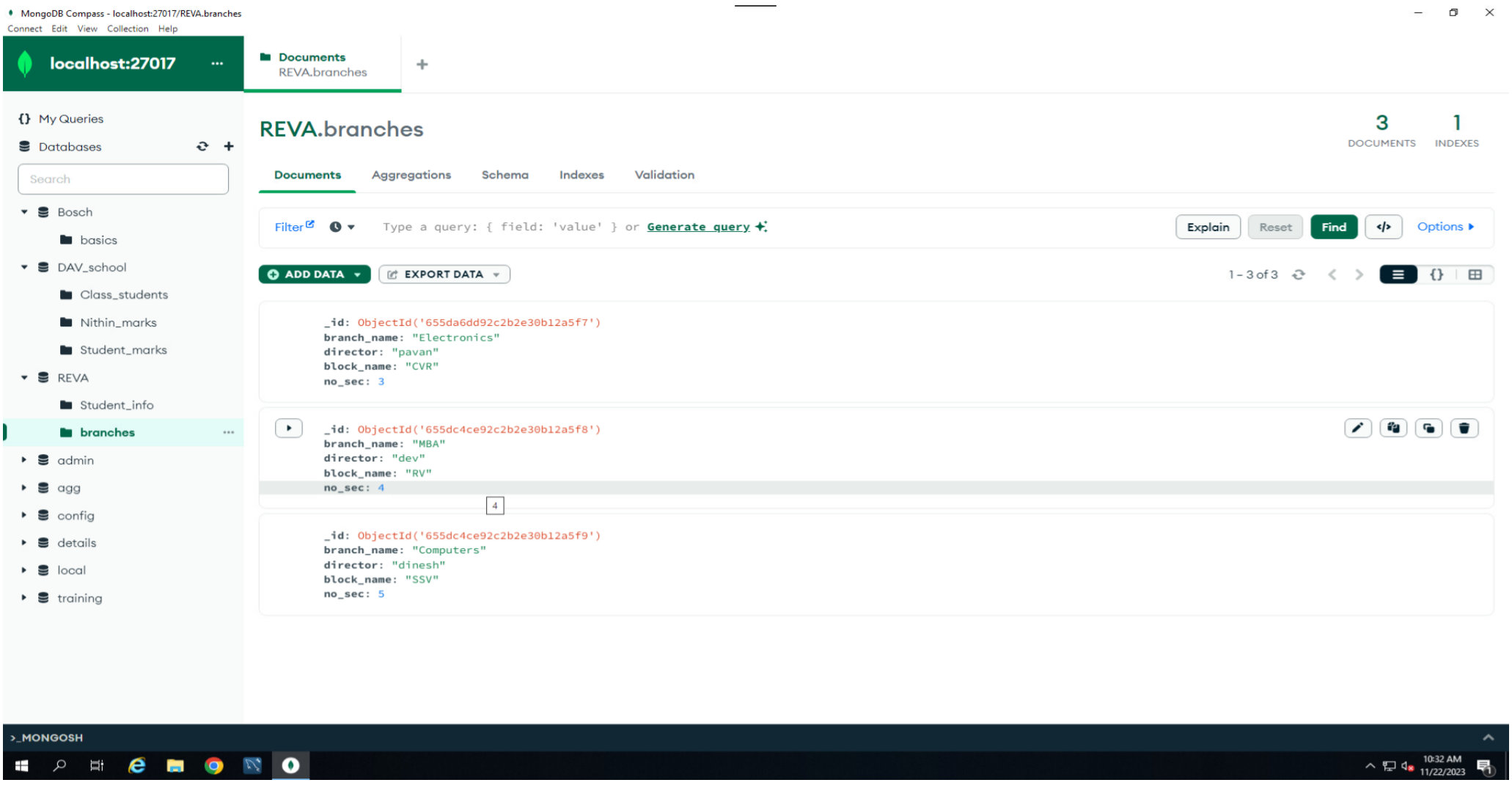


Fig:24

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

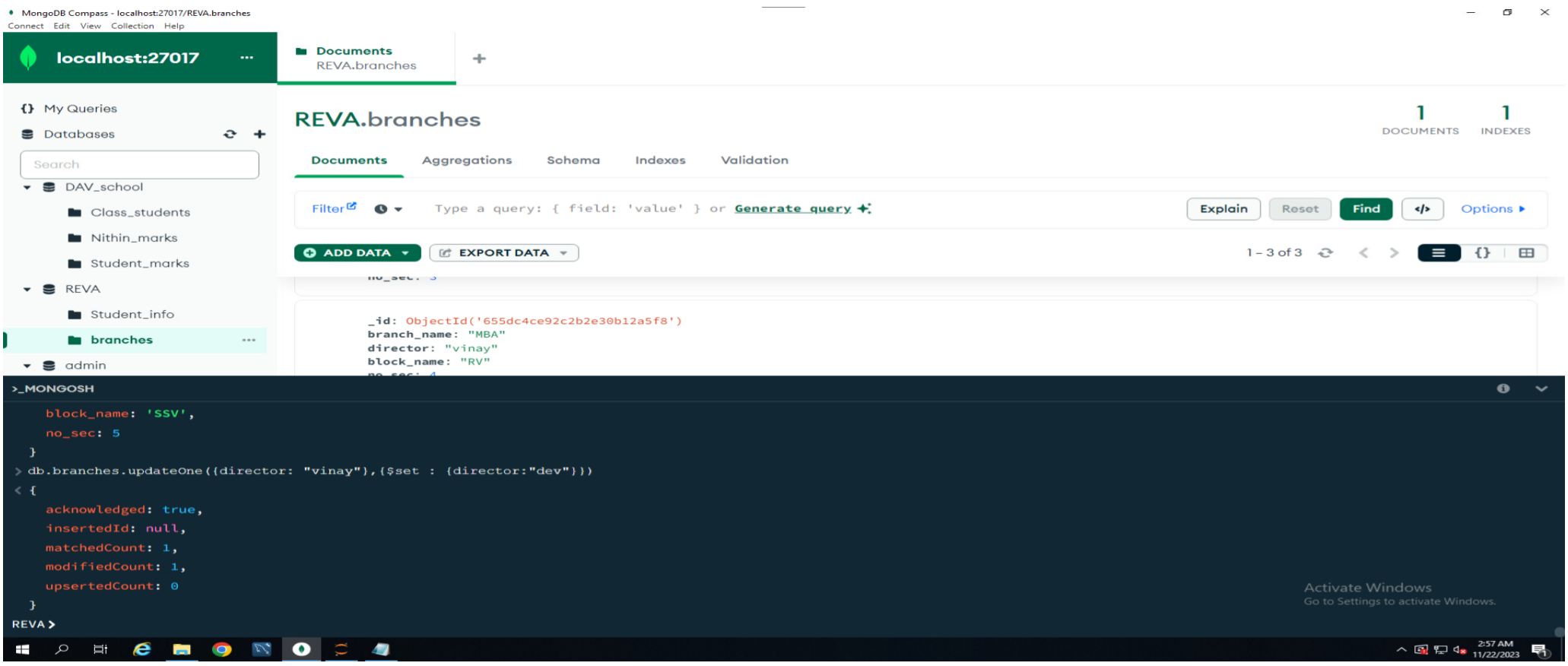


Fig:25

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

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

O/P:

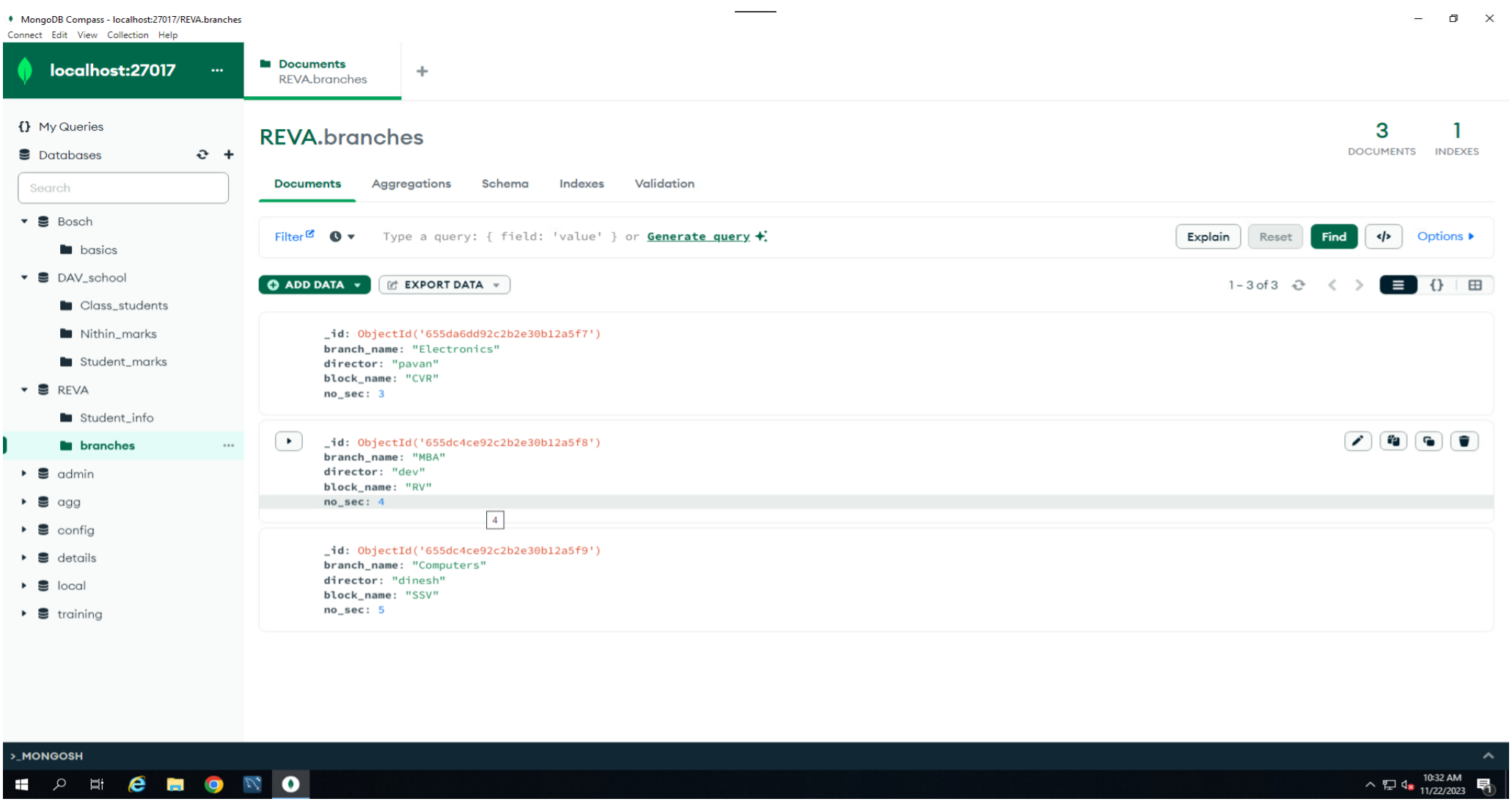


Fig:26

updateMany():

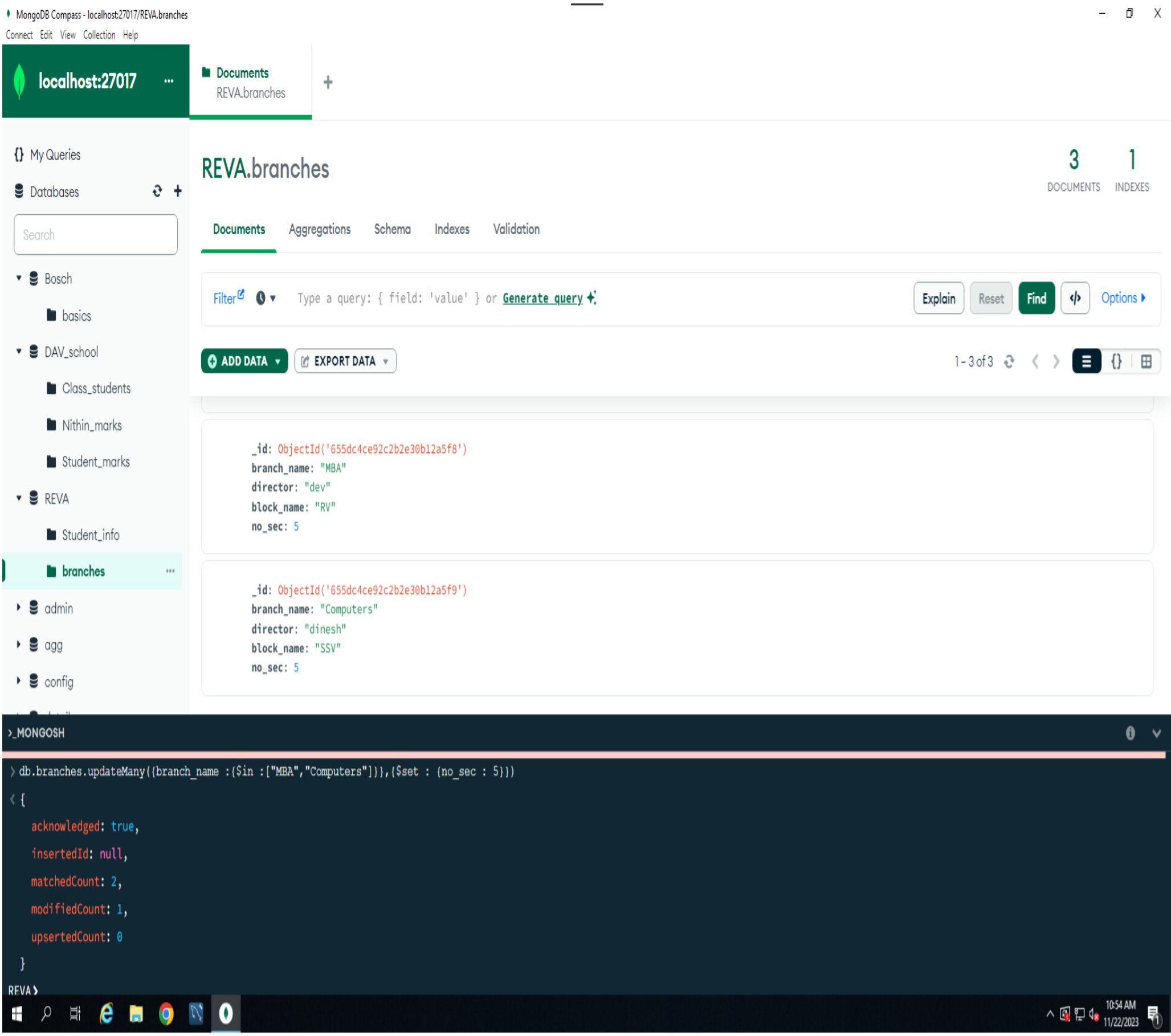


Fig:27

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():

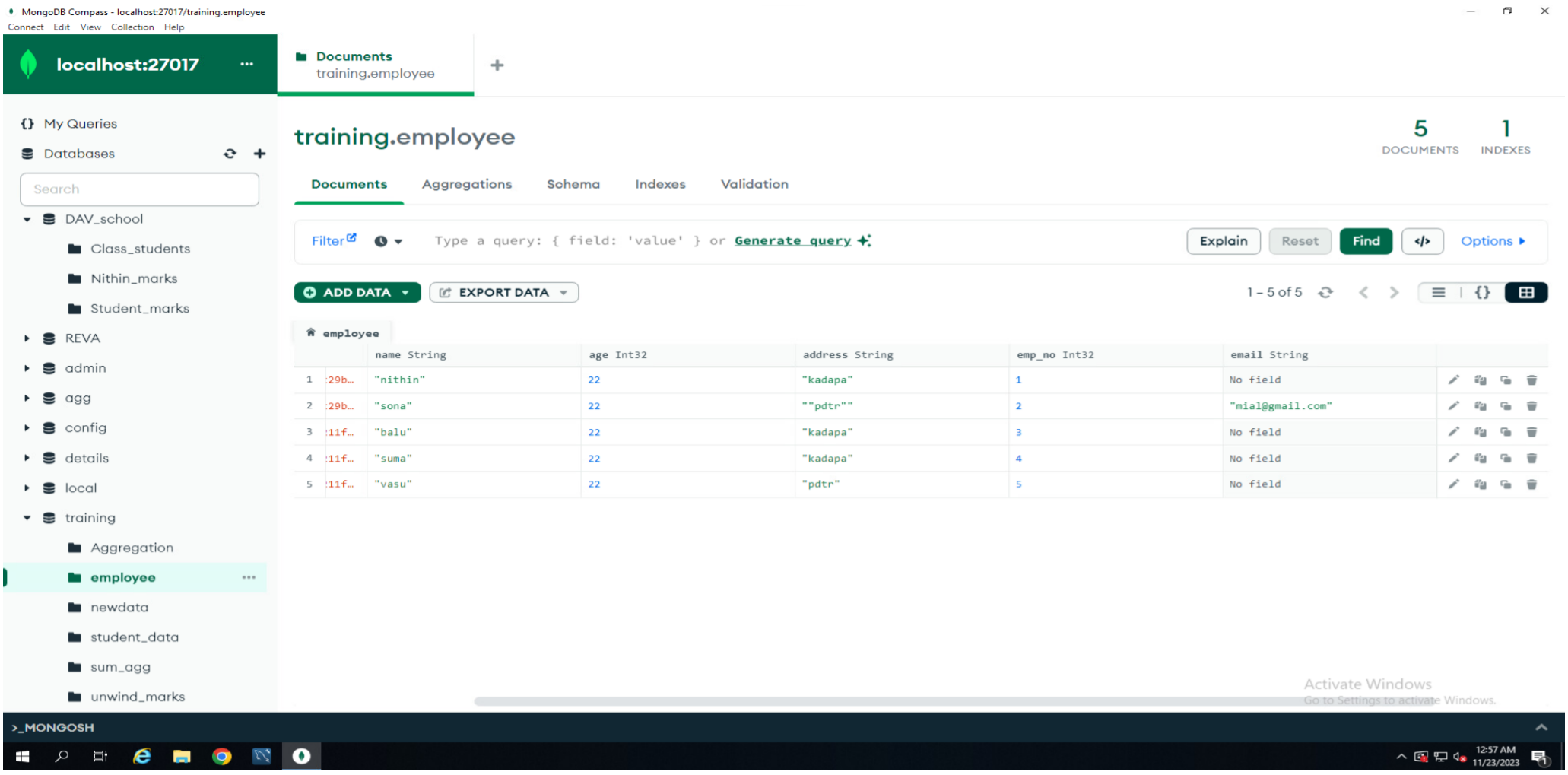


Fig:28

Having document as shown in fig28.

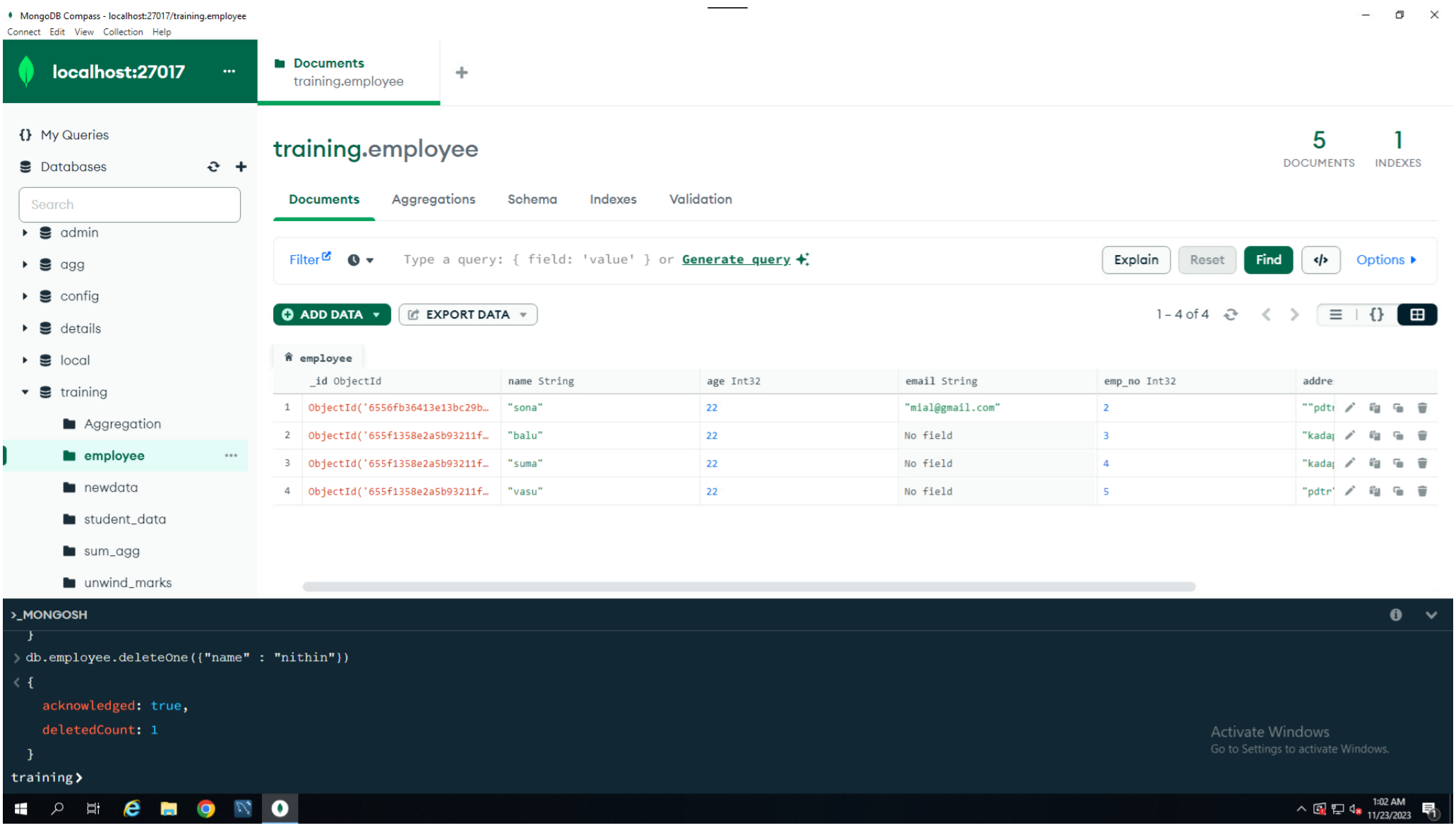


Fig:29

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():

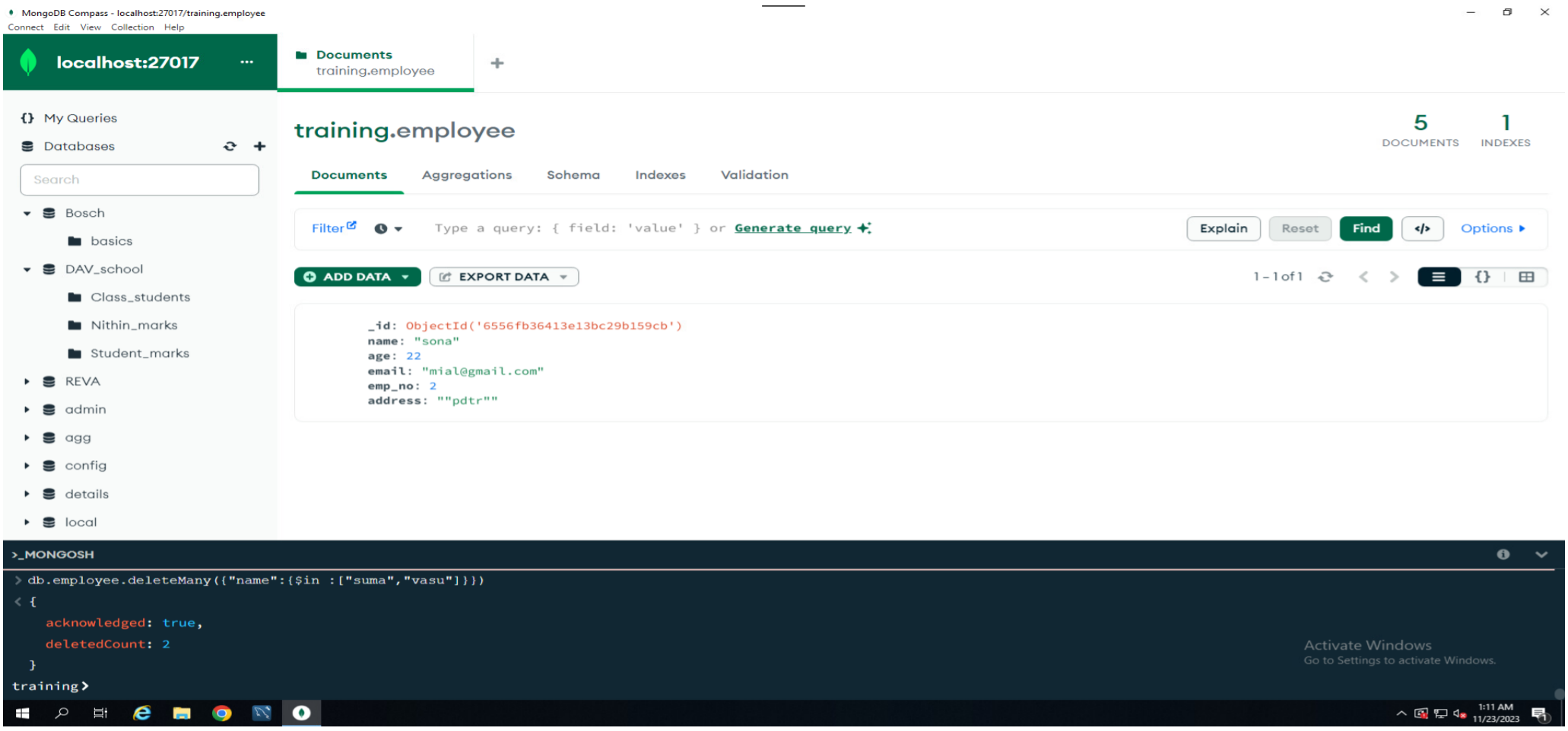


Fig:30

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

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

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

Remove database and collection in mongosh:

Drop():

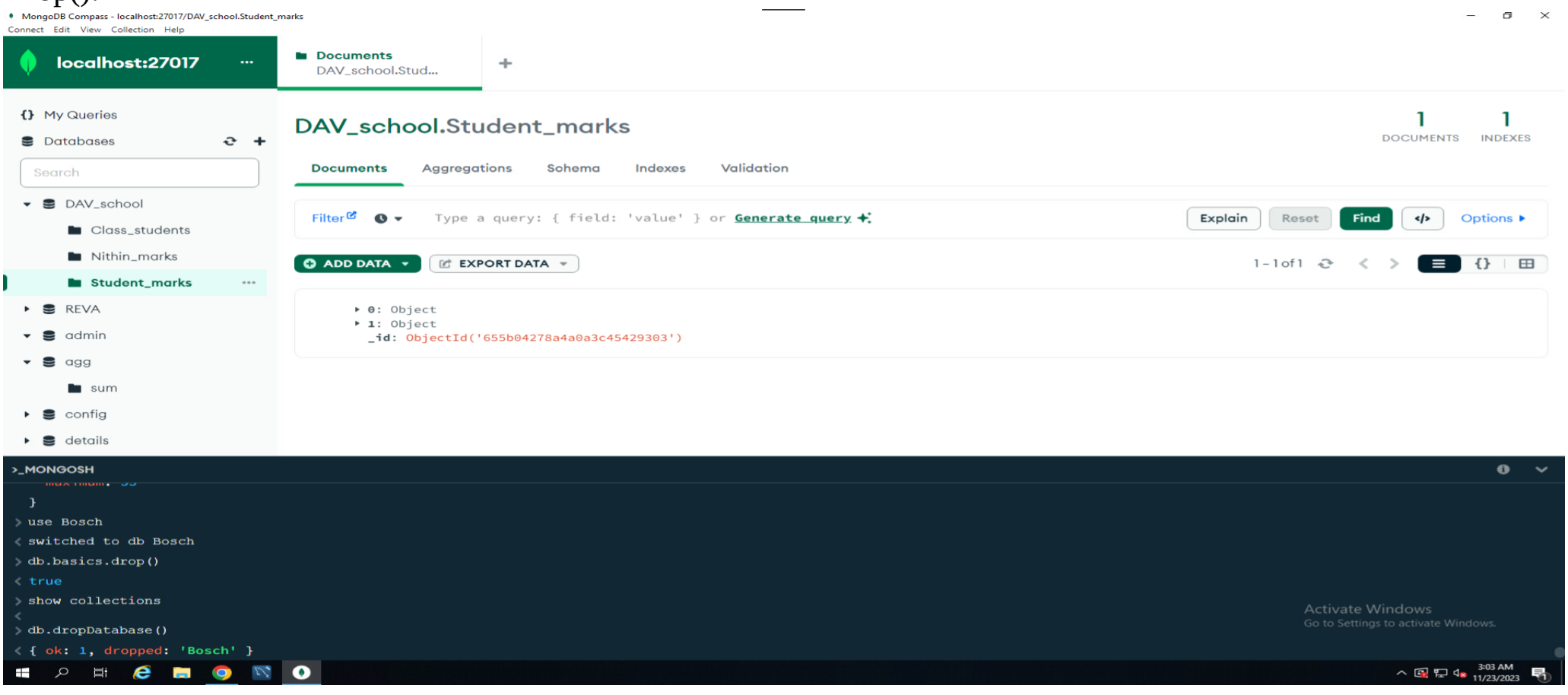


Fig:31

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:

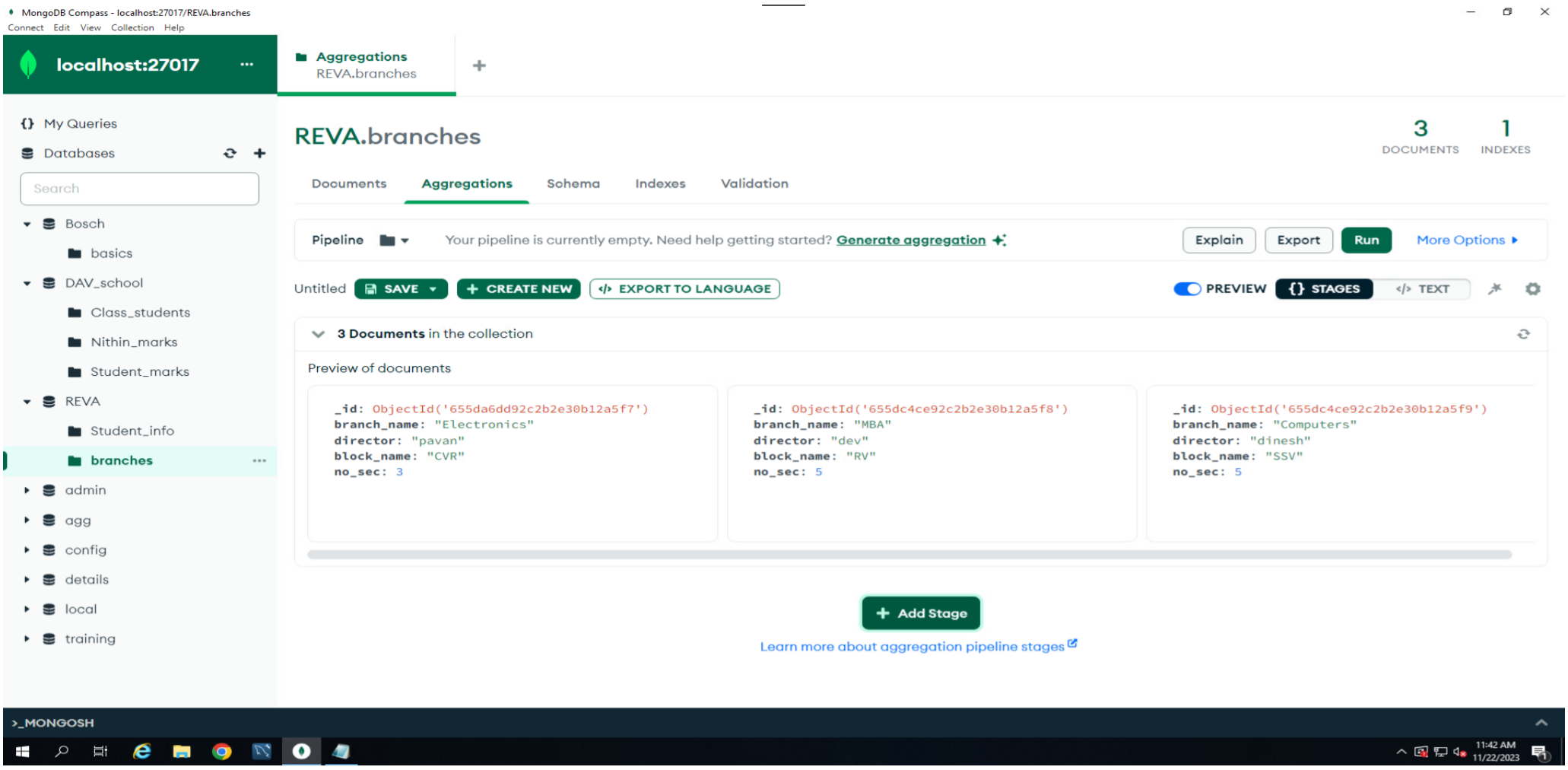


Fig:32

Steps to use Aggregation in MongoDB compass:

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

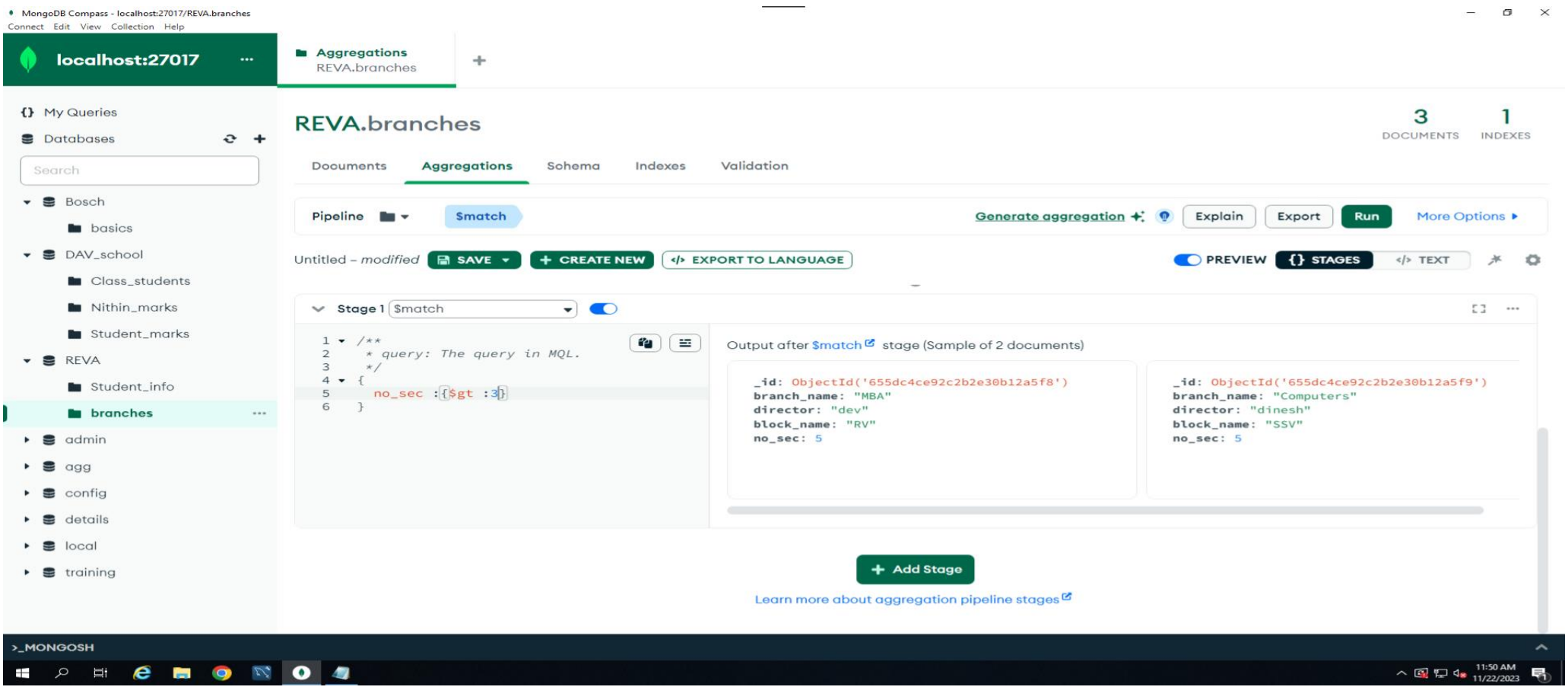


Fig:33

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.

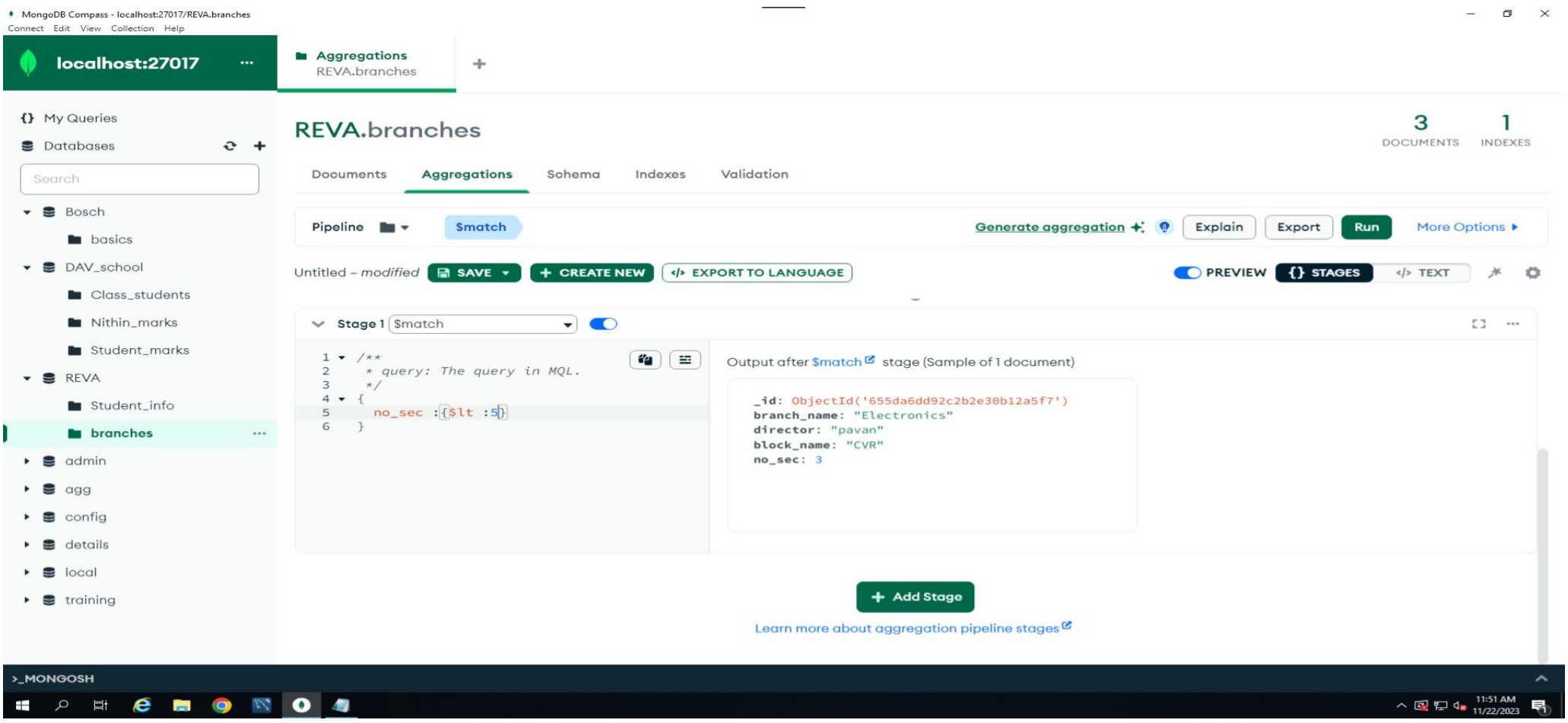


Fig:34

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

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

$match:

Greater than or equals to “gte”

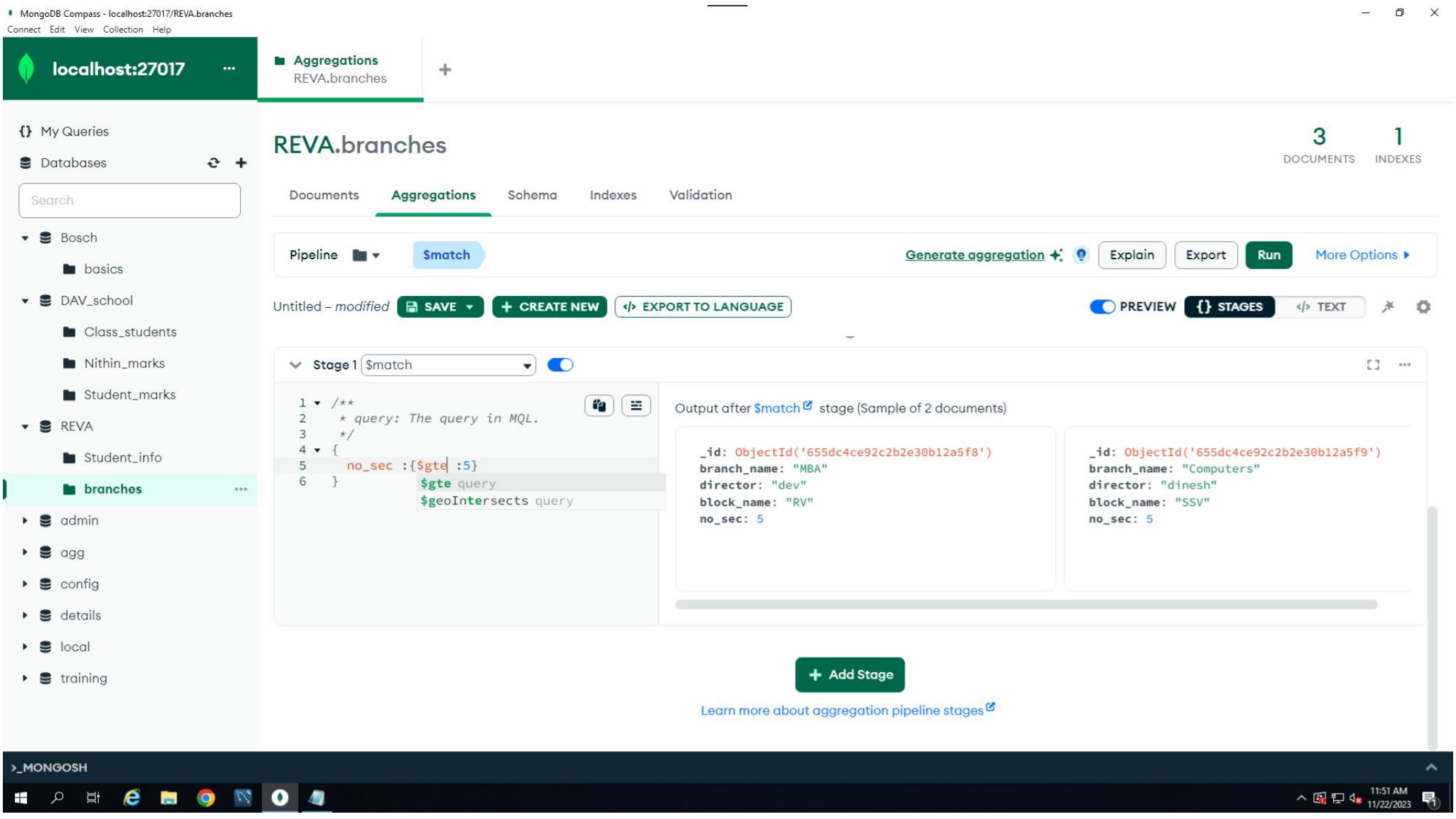


Fig:35

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”.

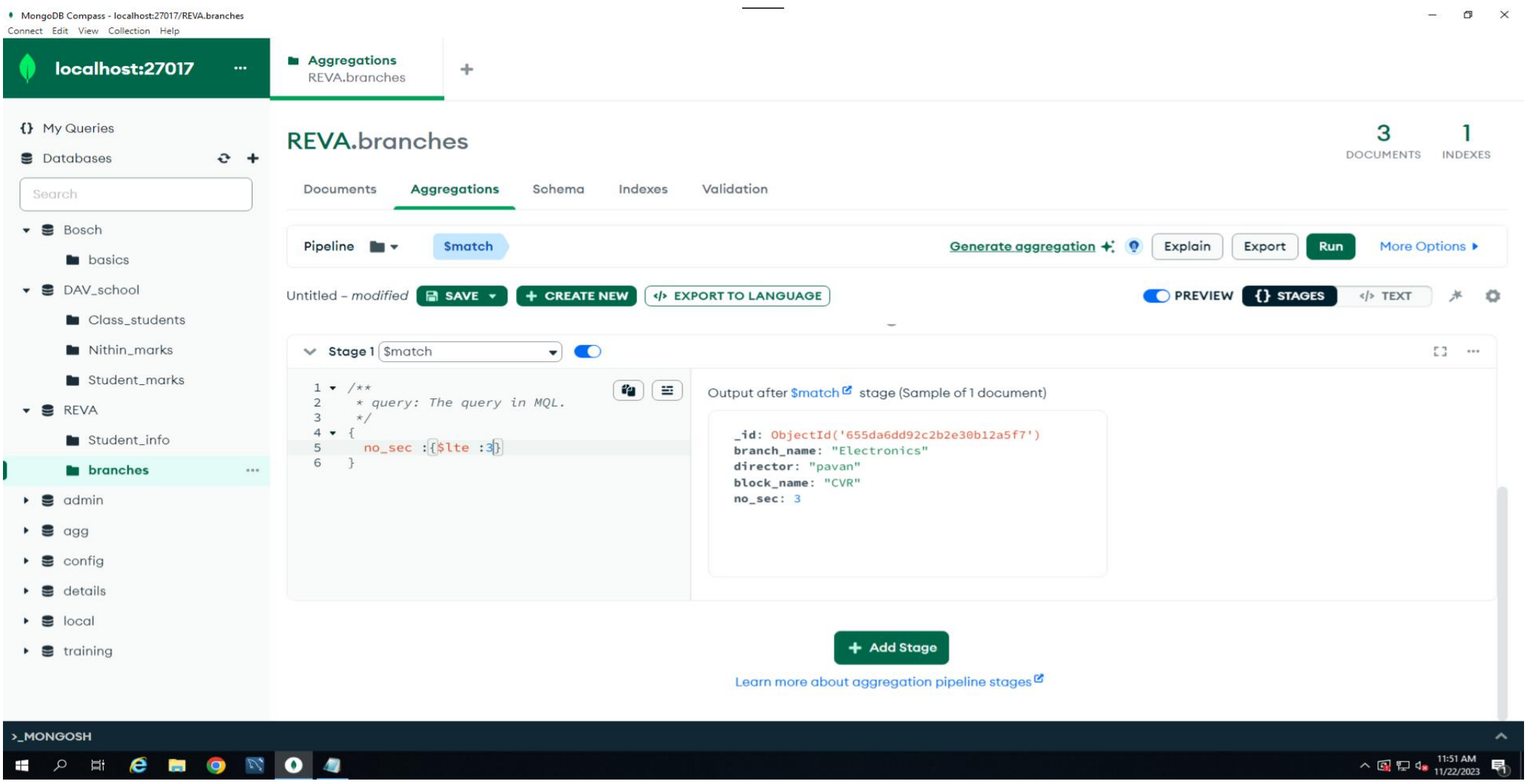


Fig:36

Ex:{no_sec :{$lte: 3}

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

$match:

Not equal to “ne”

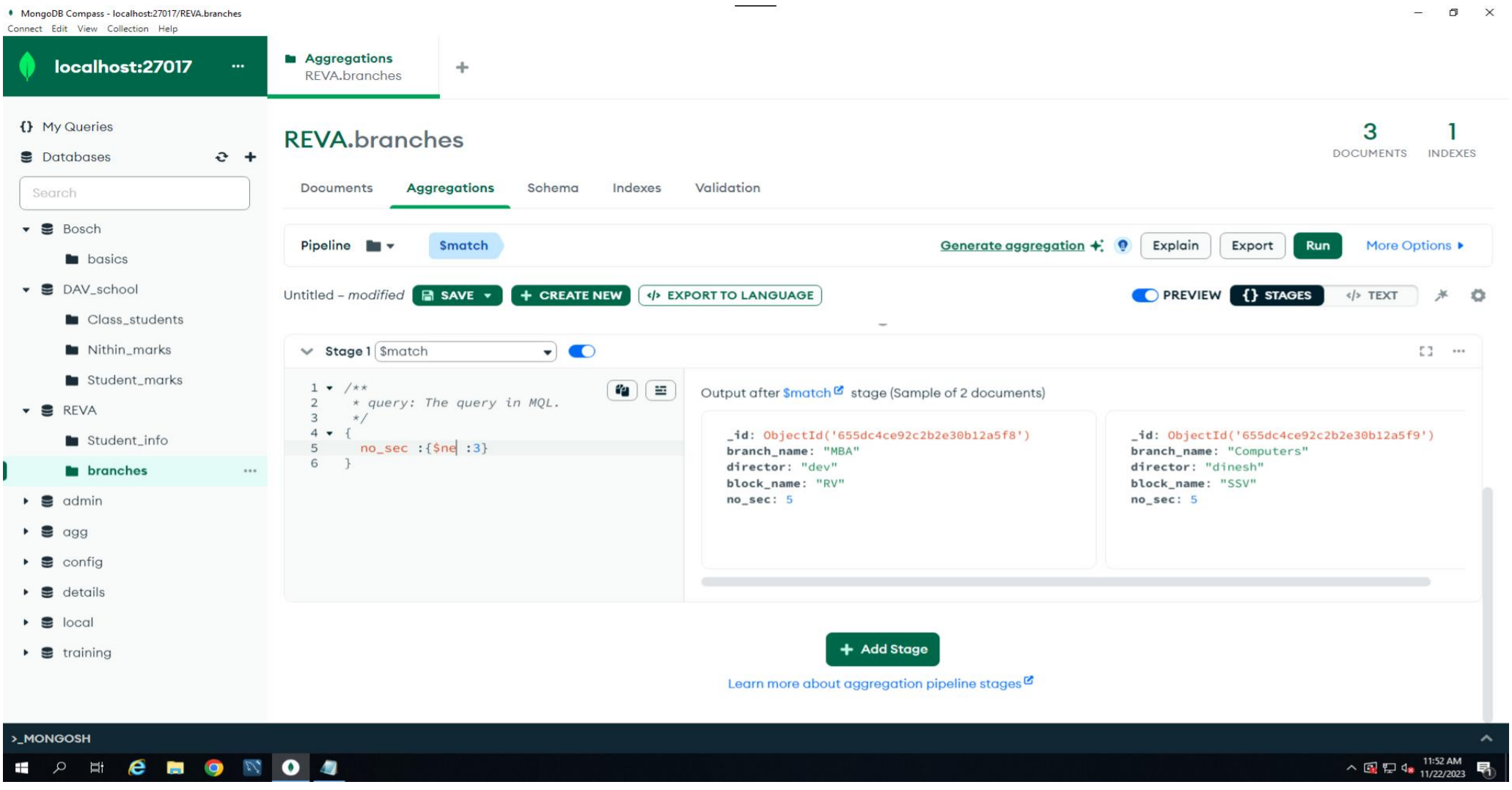


Fig:37

Ex: {no_sec :{$ne:3}

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

$match:

Equal to “eq”

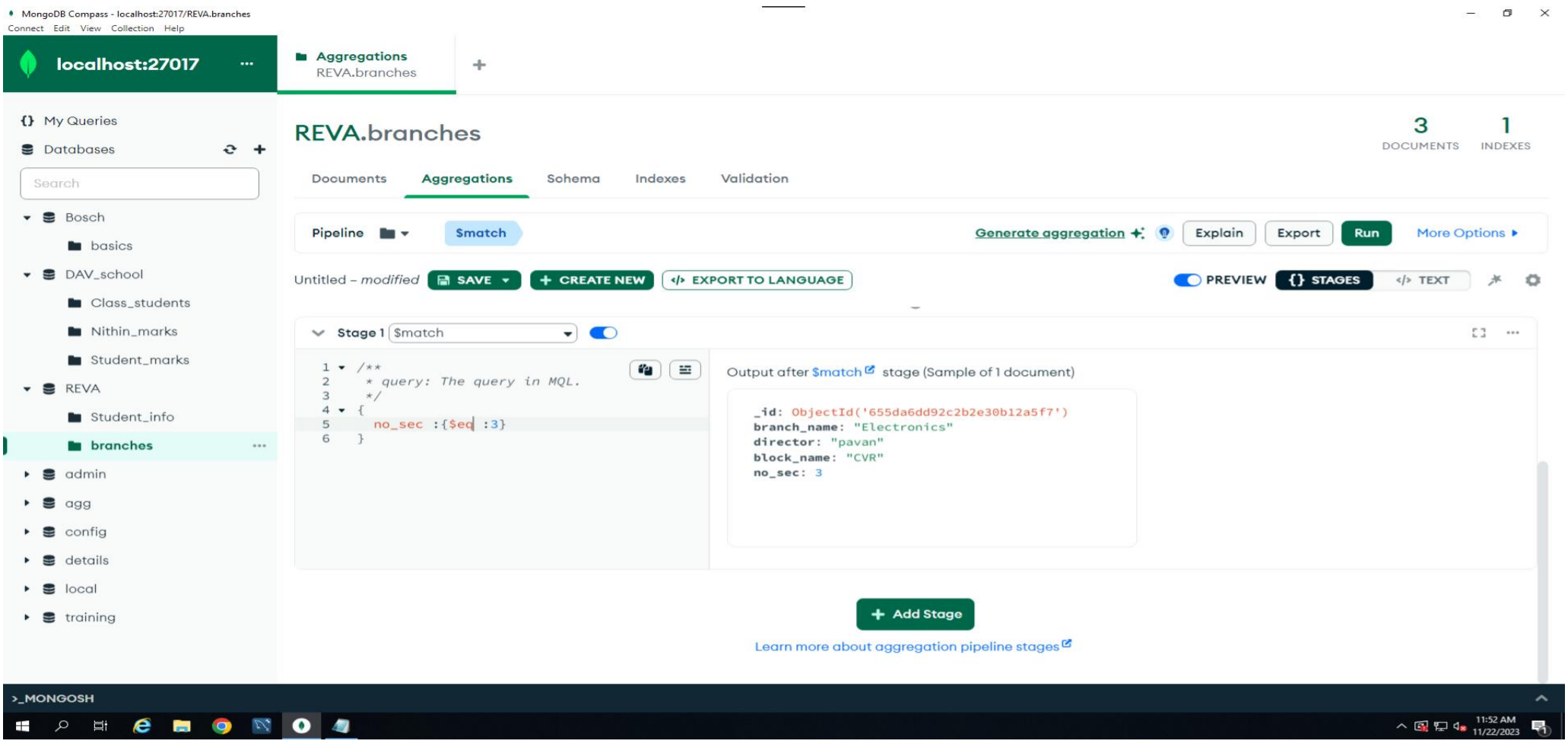


Fig:38

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.

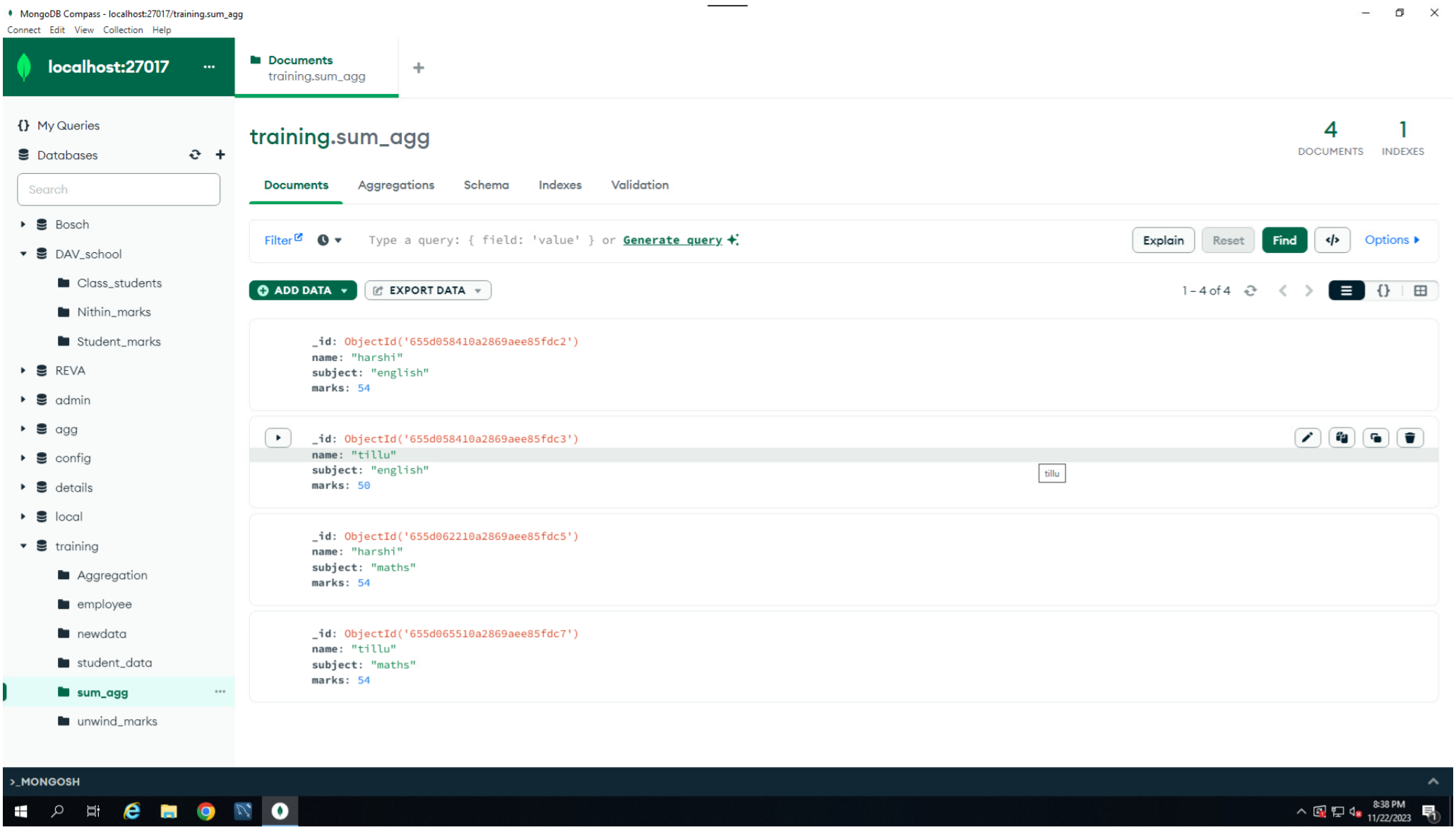


Fig:39

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

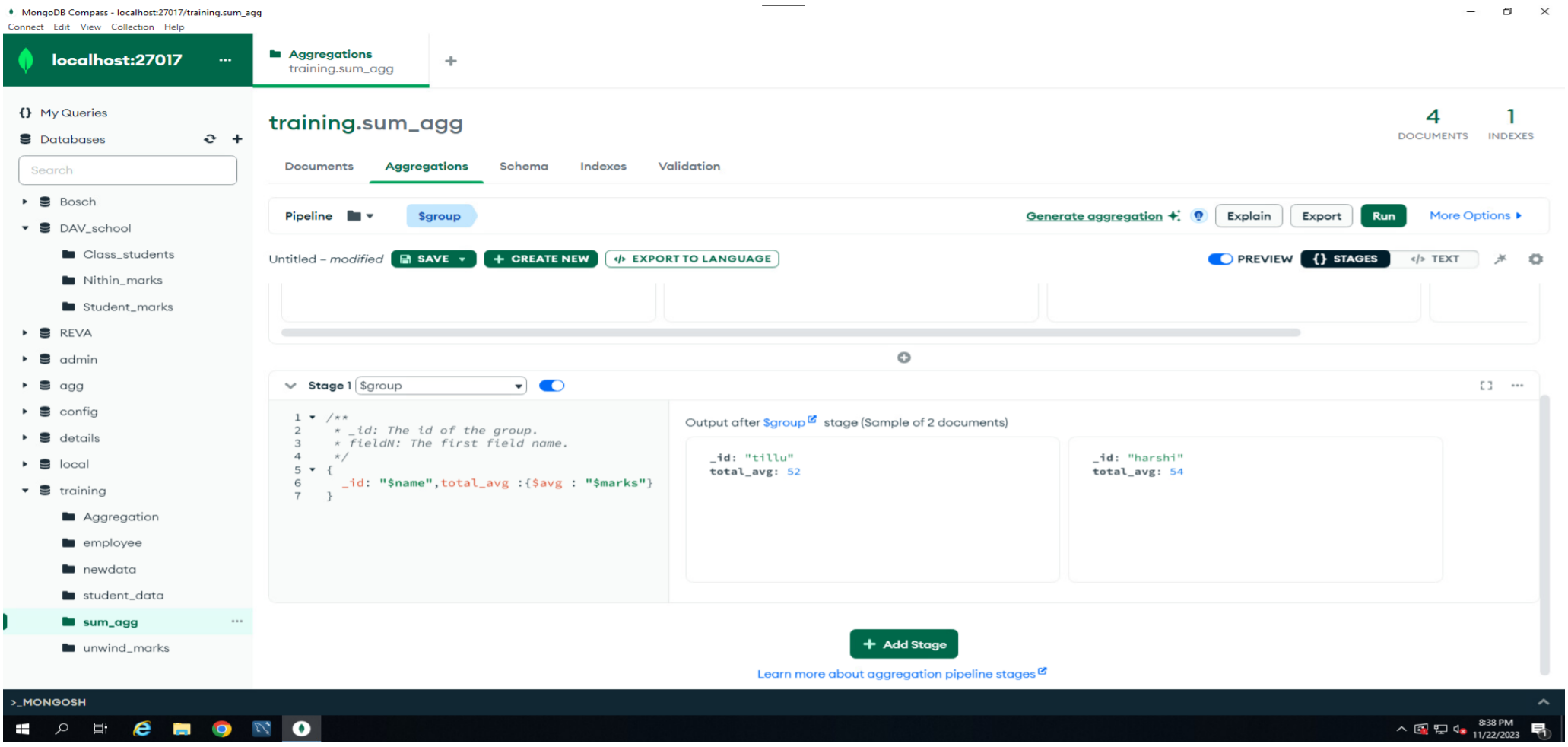


Fig:40

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.

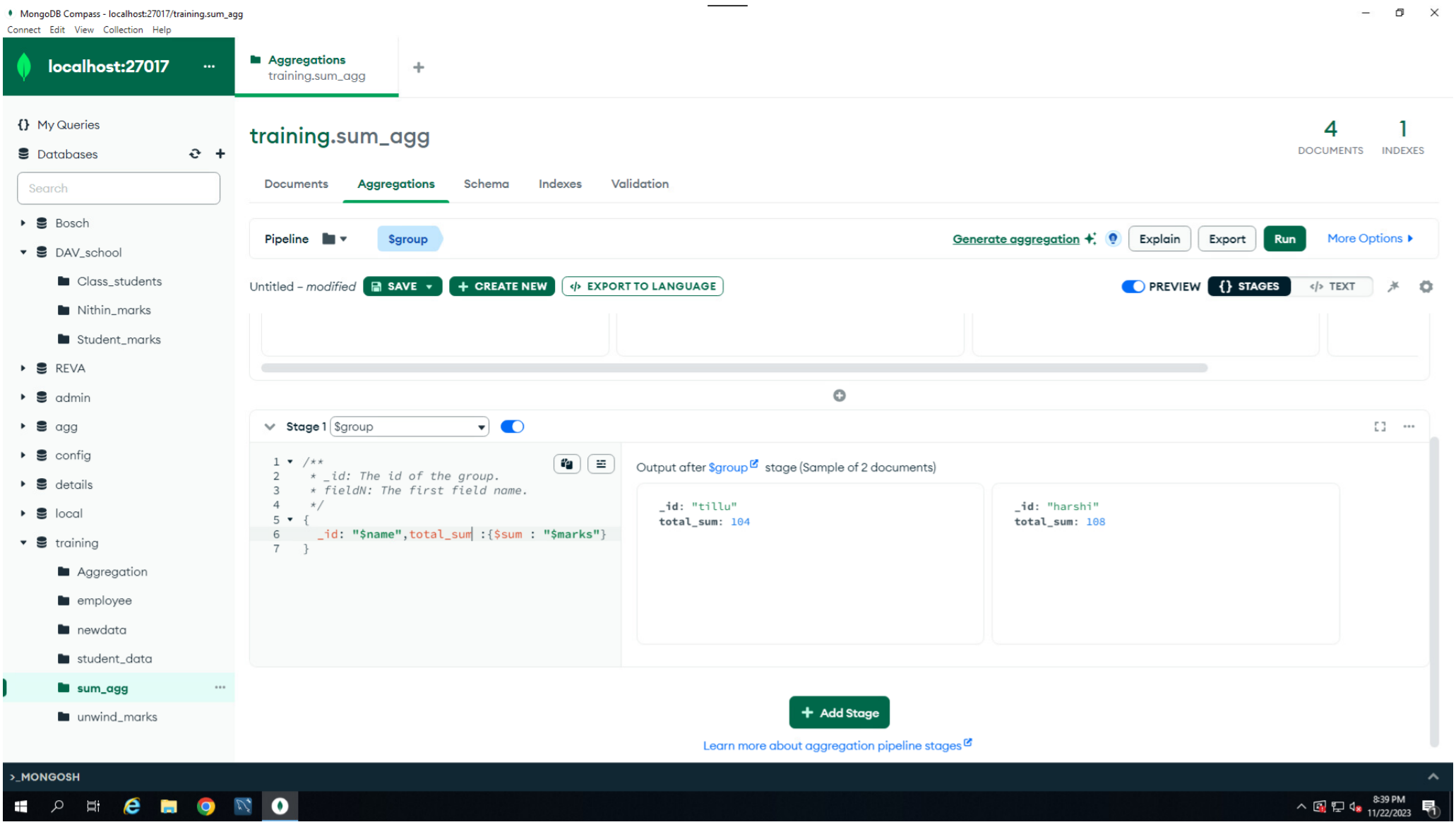


Fig:41

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.

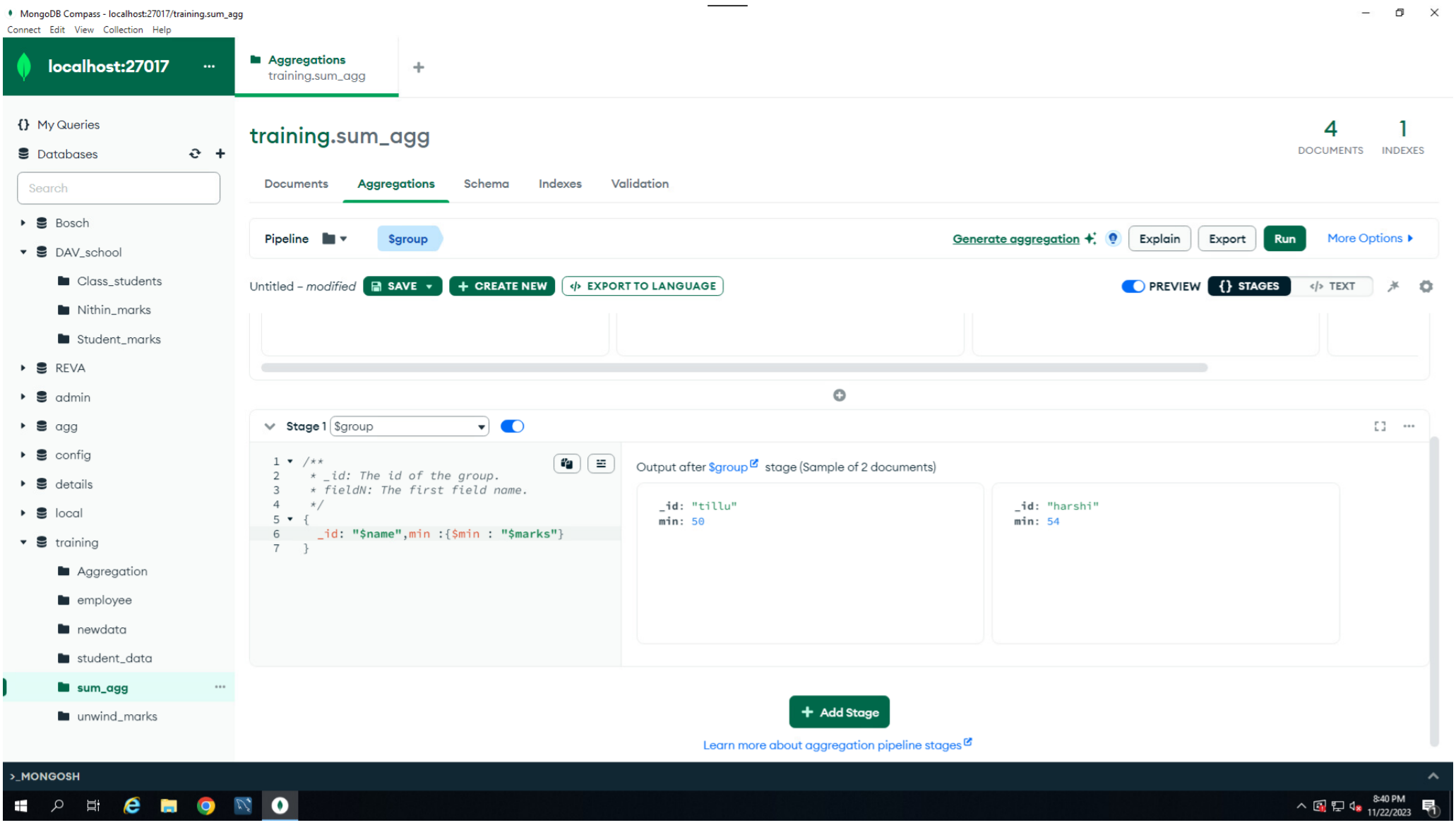


Fig:42

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.

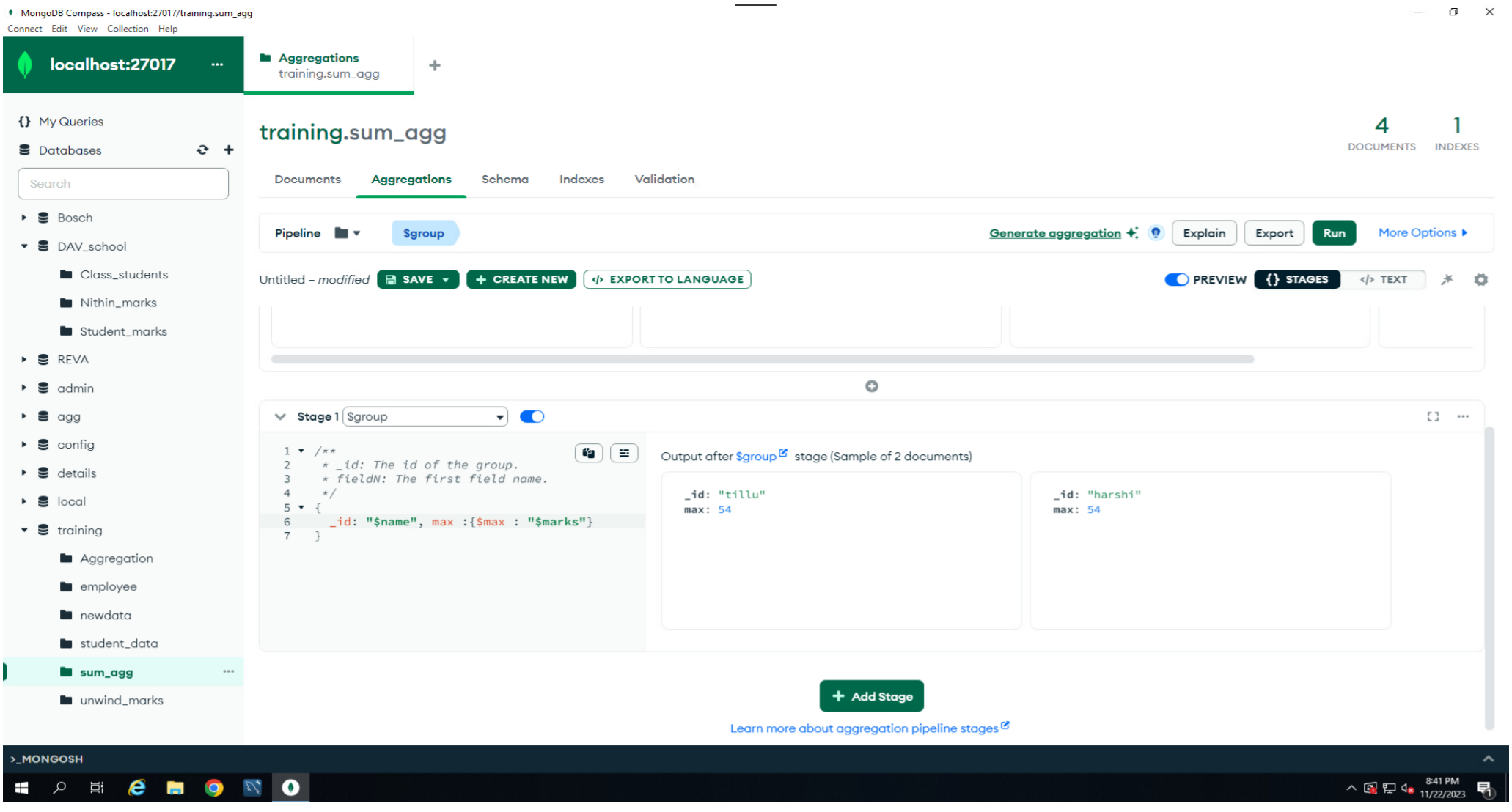


Fig:43

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 employe optimize performance by limiting the amount of data processed.

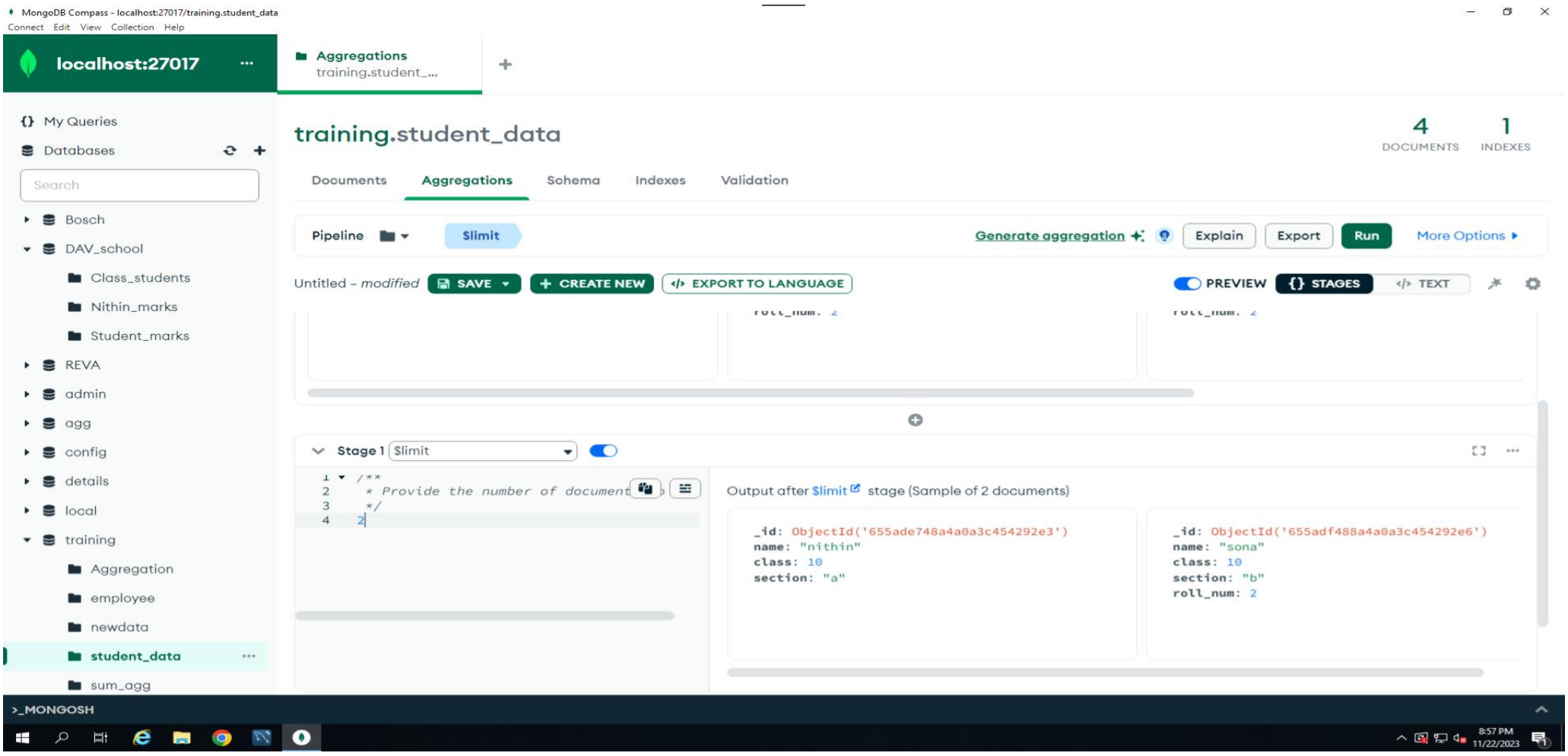


Fig:44

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.

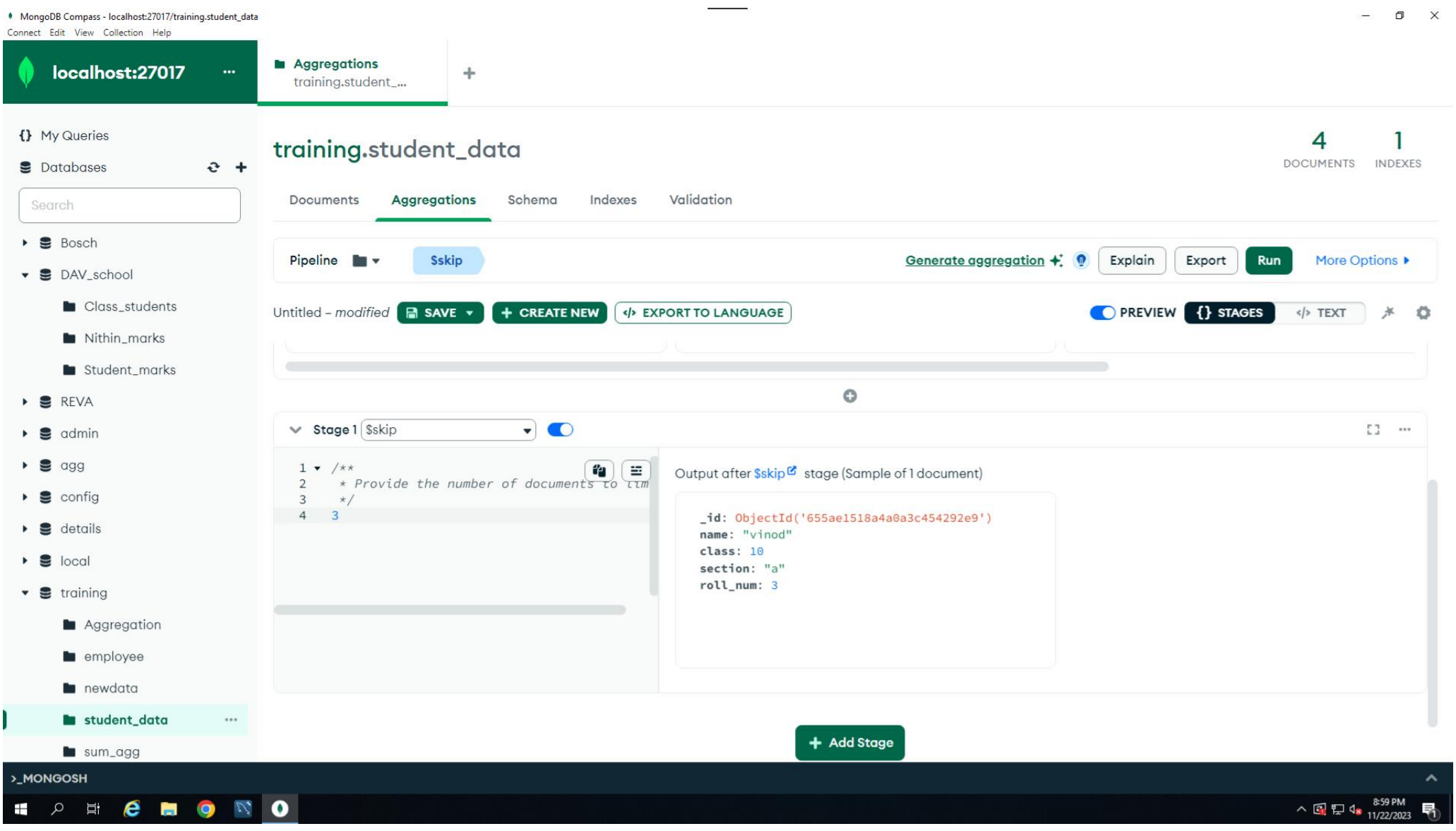


Fig:45

Syntax: {$Skip: “number”}

Ex: 3

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

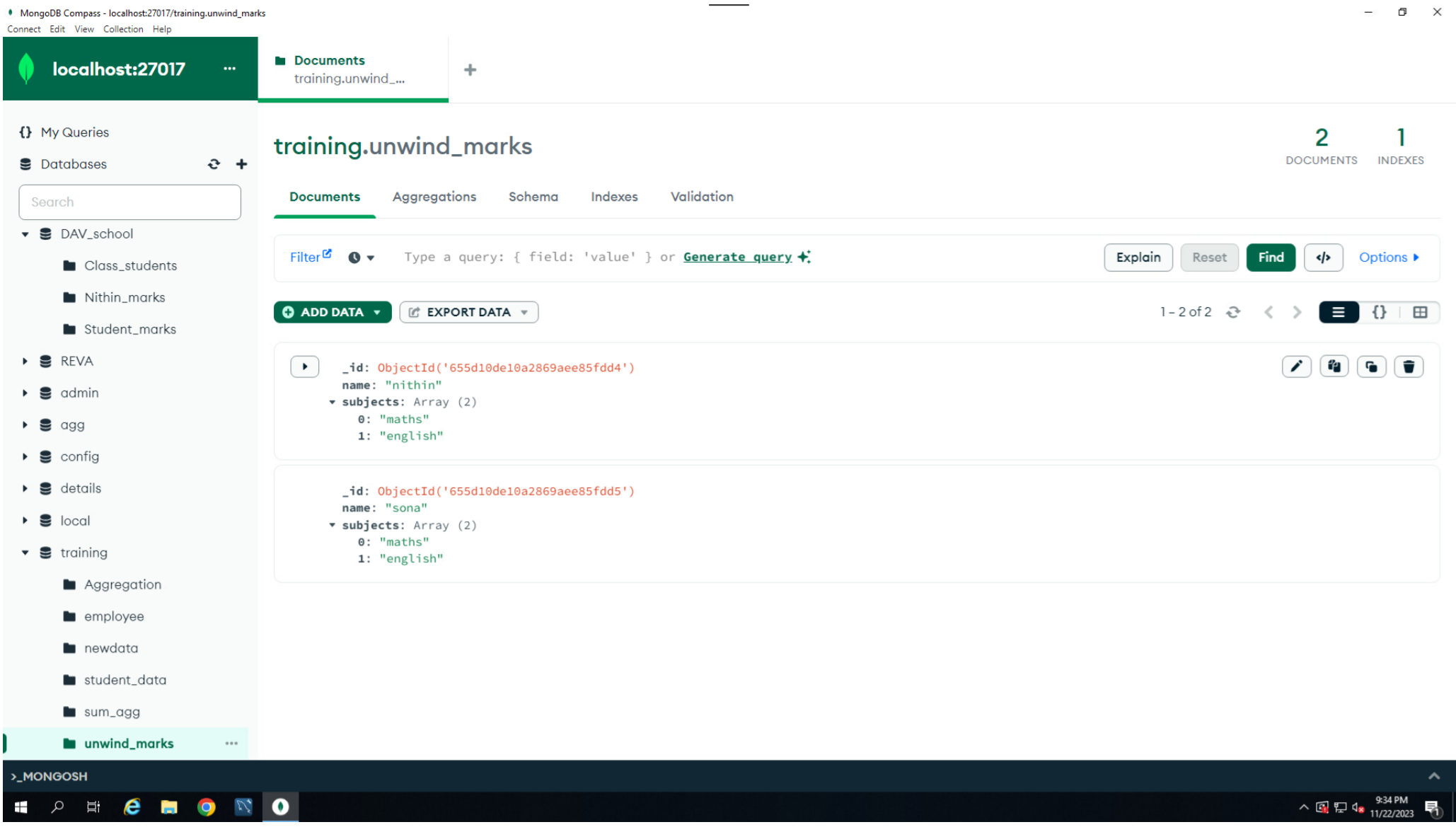


Fig:46

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

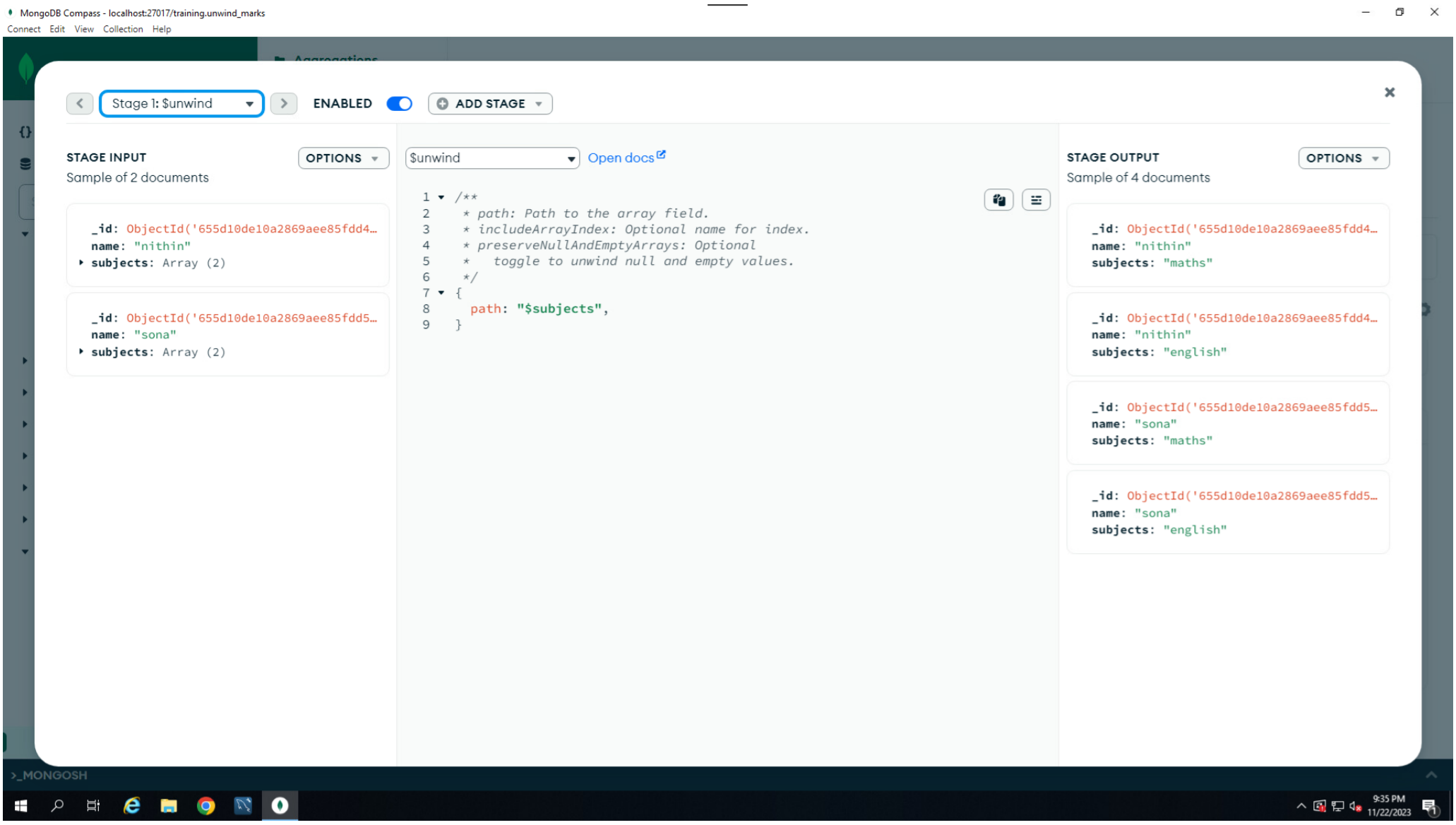


Fig:47

Syntax $unwind {path:” $array_field}

Ex: {path: “$subjects”}

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

$addFileds: 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.

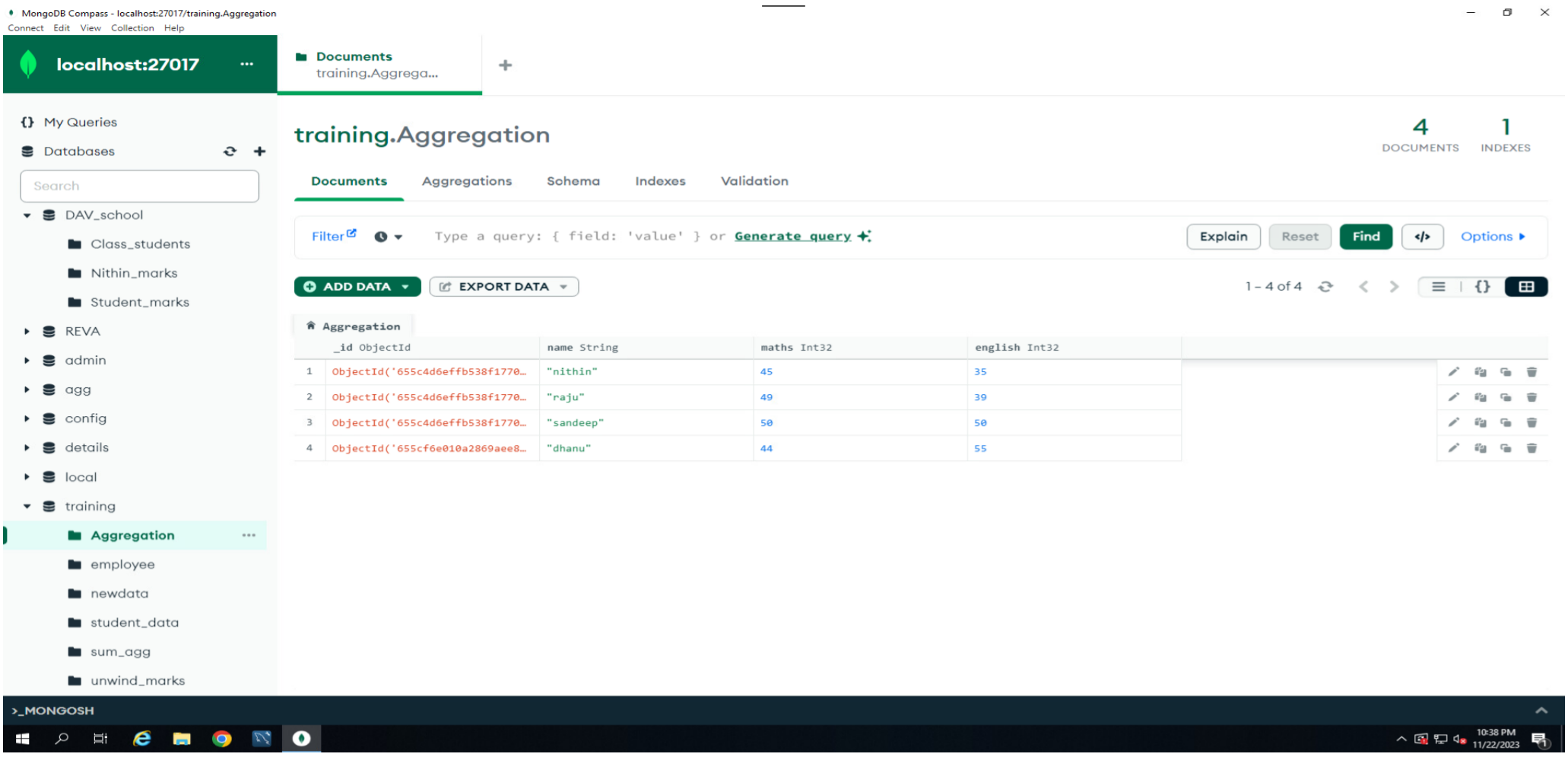


Fig:48

Consider fig48 is the document for addFields operations.

$addFields $sum:

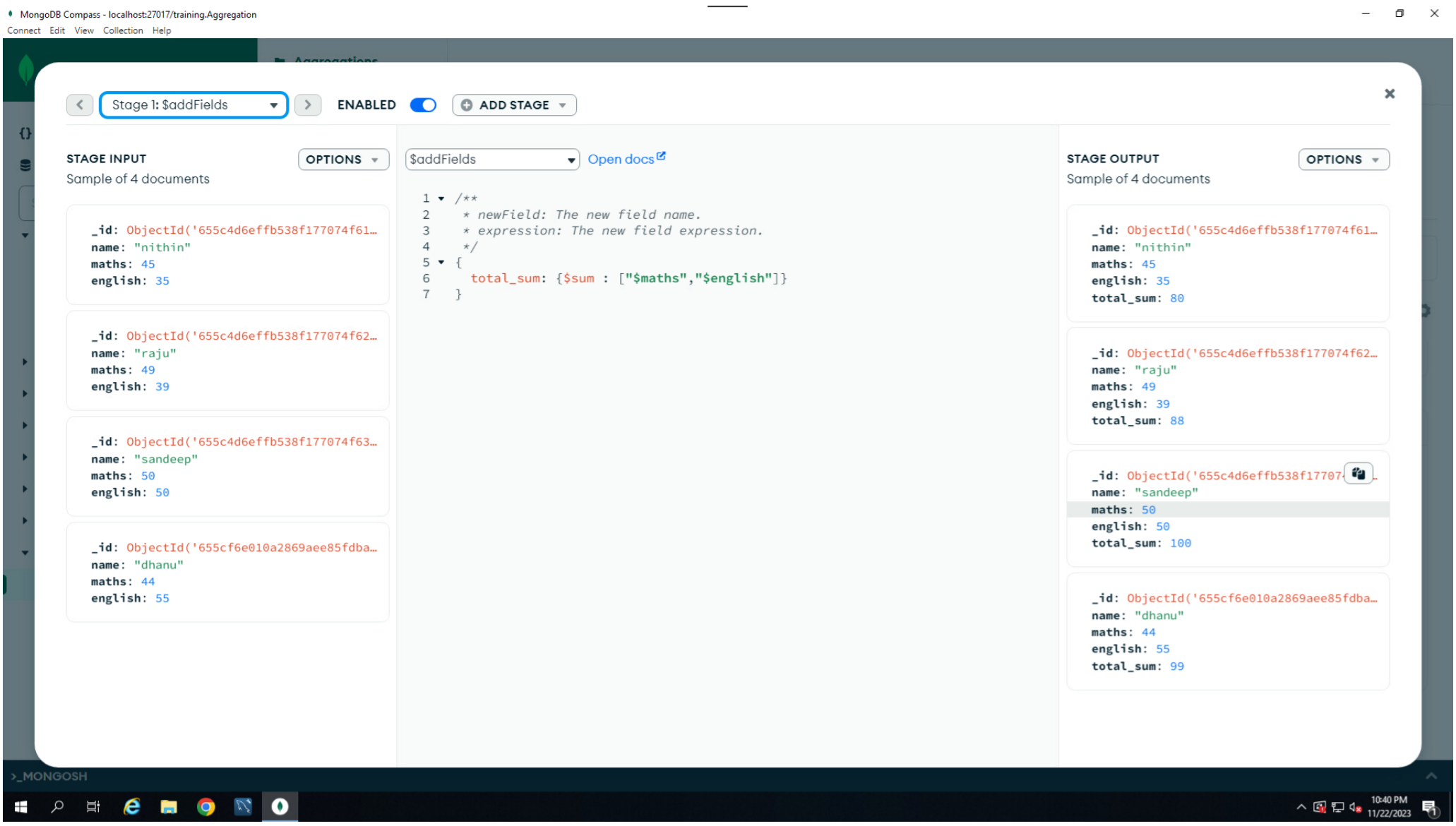


Fig:49

Syntax: $addFields {new_field :{$sum :[“field1”,”field2”]}}

Ex: $addFields {total_marks :{$sum :[“maths”,”english”]}}

$addFields $avg :

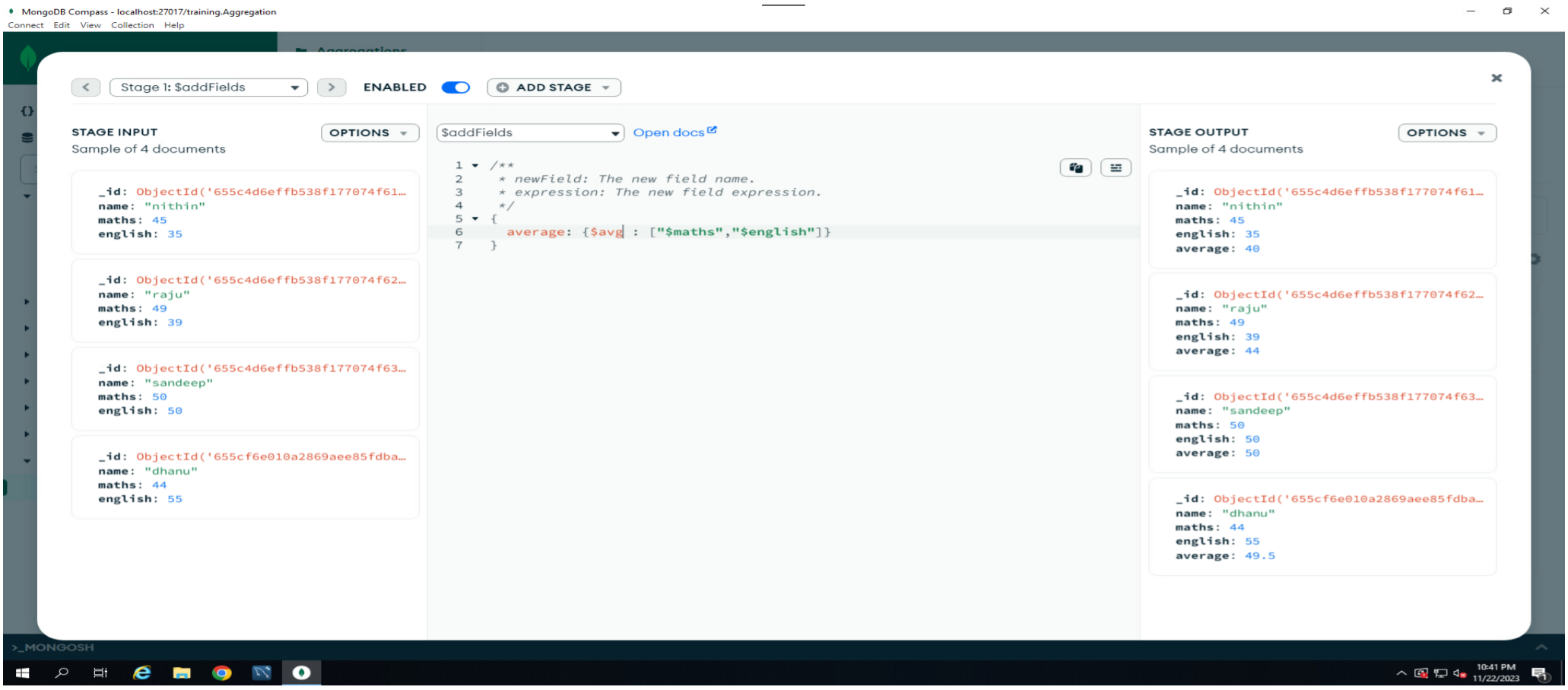


Fig:50

Syntax: $addFields {new_field :{$vg :[“field1”,”field2”]}

Ex: $addFields {average :{$avg :[“maths”,”english”]}}

$addFields $min

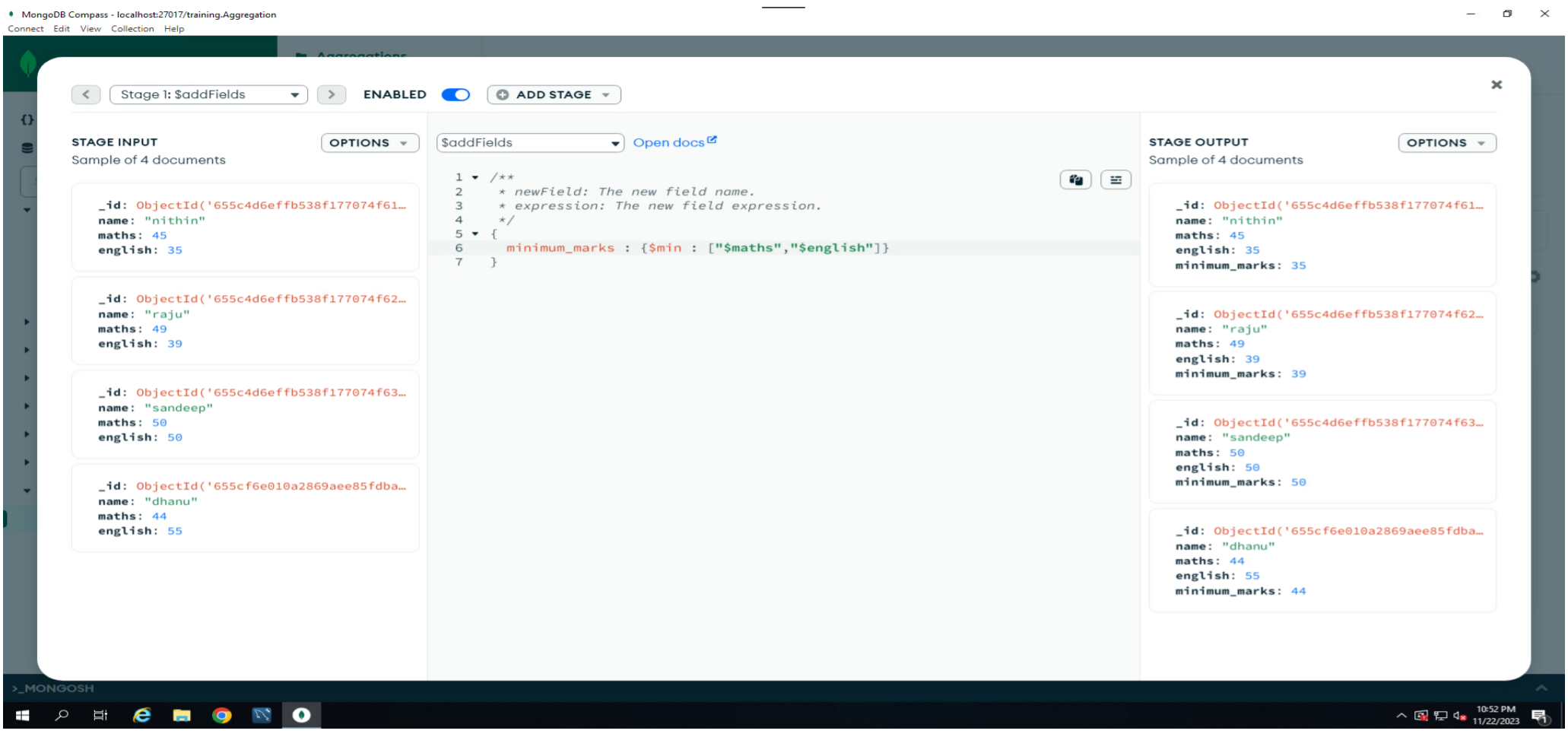


Fig:51

Syntax: $addFields {new_field :{$min :[“field1”,”field2”]}}

Ex: $addFields {minimum_marks :{$min :[“maths”,”english”]}}

$addField $max:

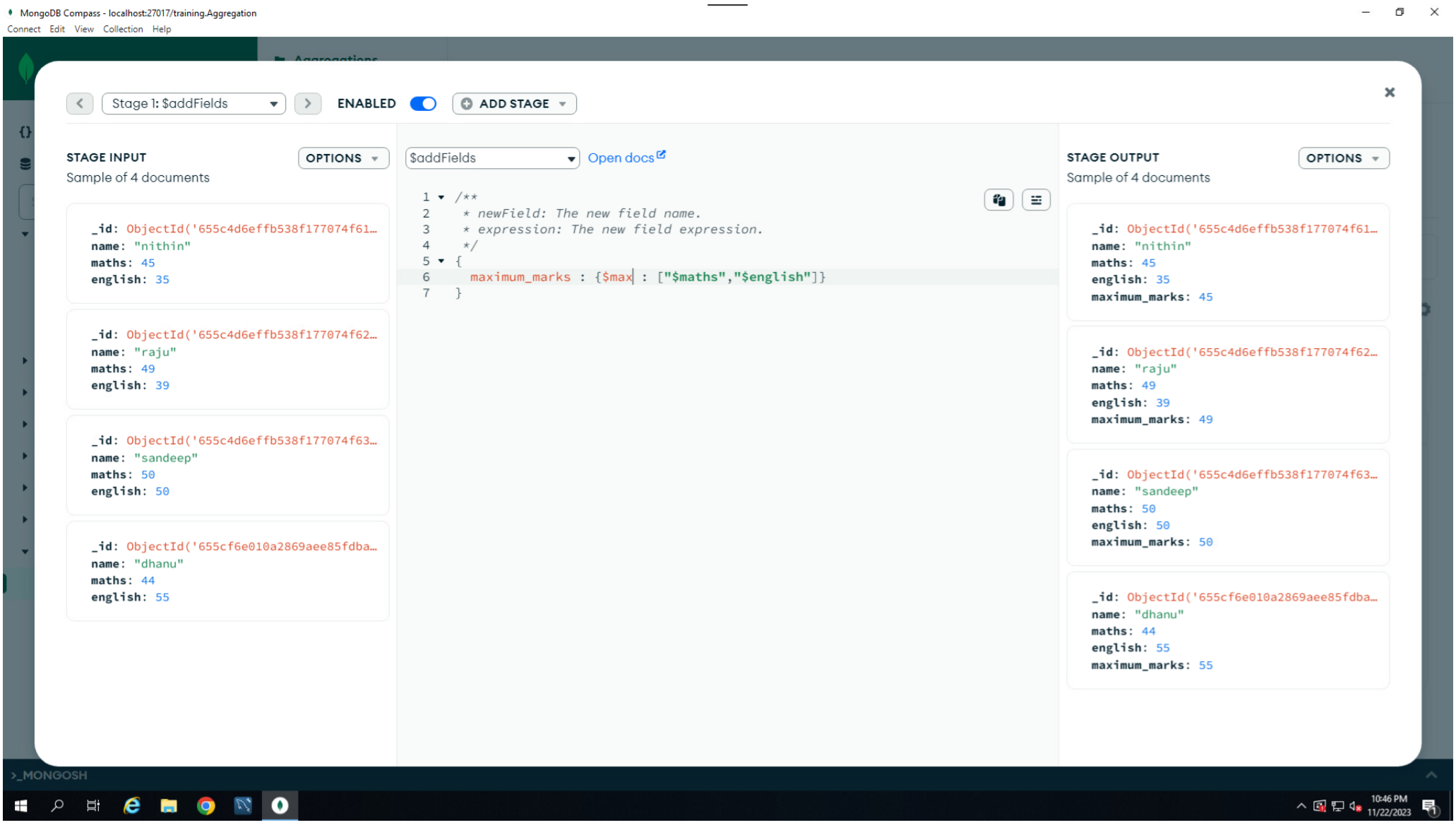


Fig:52

Syntax: $addFields {new_field :{$max :[“field1”,”field2”]}}

Ex: $addFields {maximum_marks :{$max :[“maths”,”english”]}}

Aggregation in mongosh:

$addFields: $sum

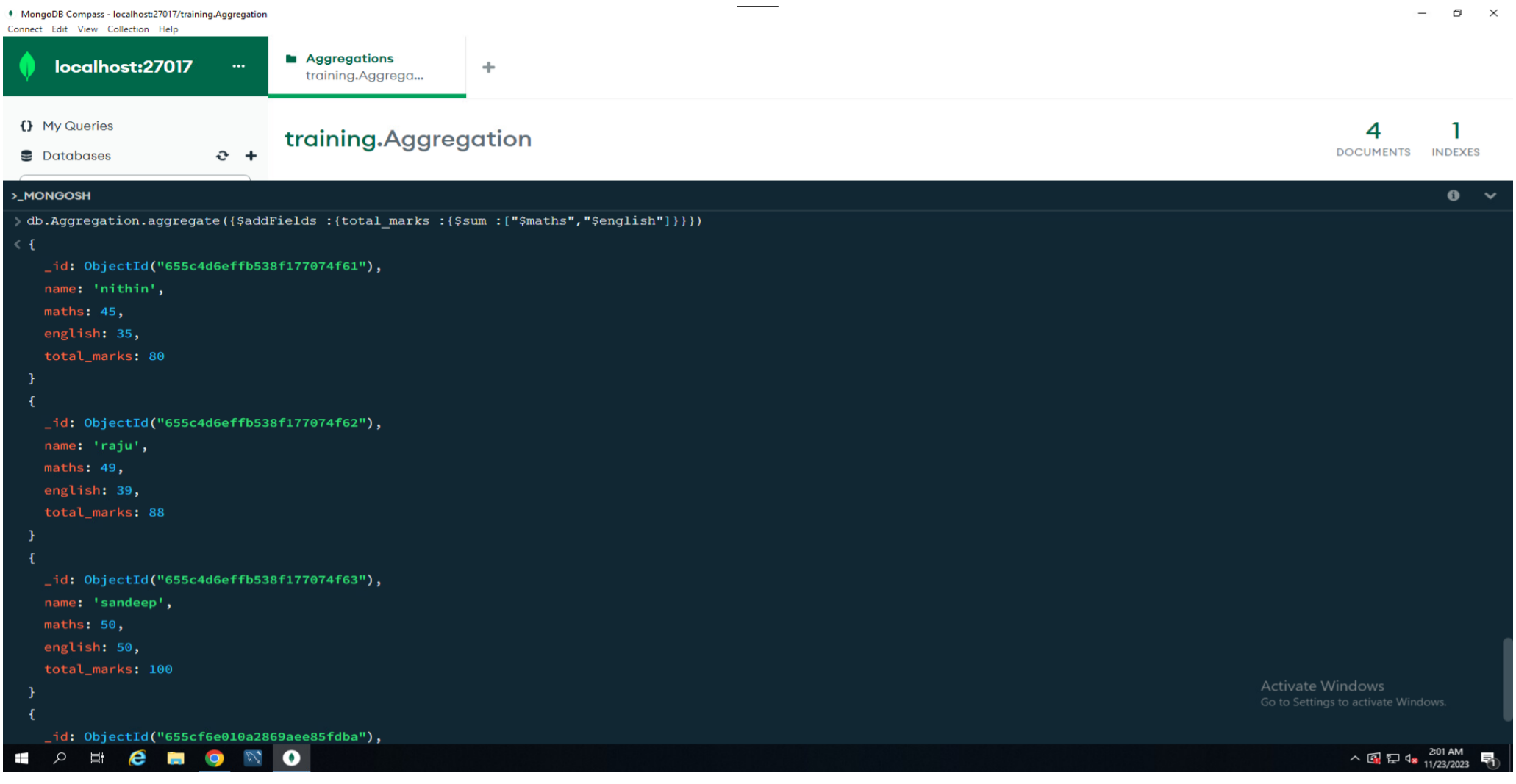


Fig:53

Syntax: db.Collection_name.aggrigate({$addFields :new_field :{$sum :[“$field1”,”$field2”]}}})

Ex: db.Aggregation.aggrigate({$addFields:{ total_marks :{$sum :[“$maths”,”$english”]}}})

$addField $avg:

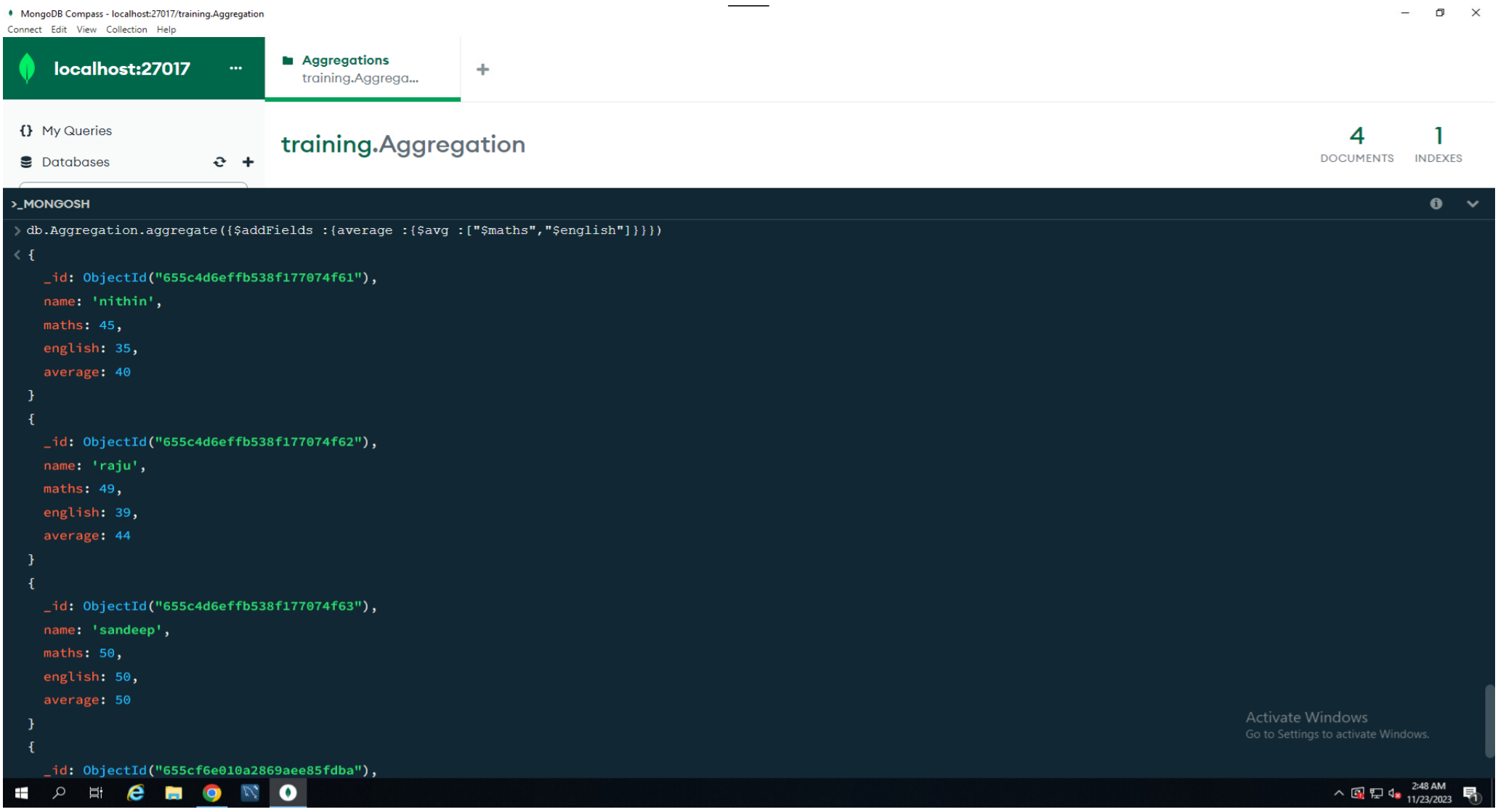


Fig:54

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

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

$addField $min:

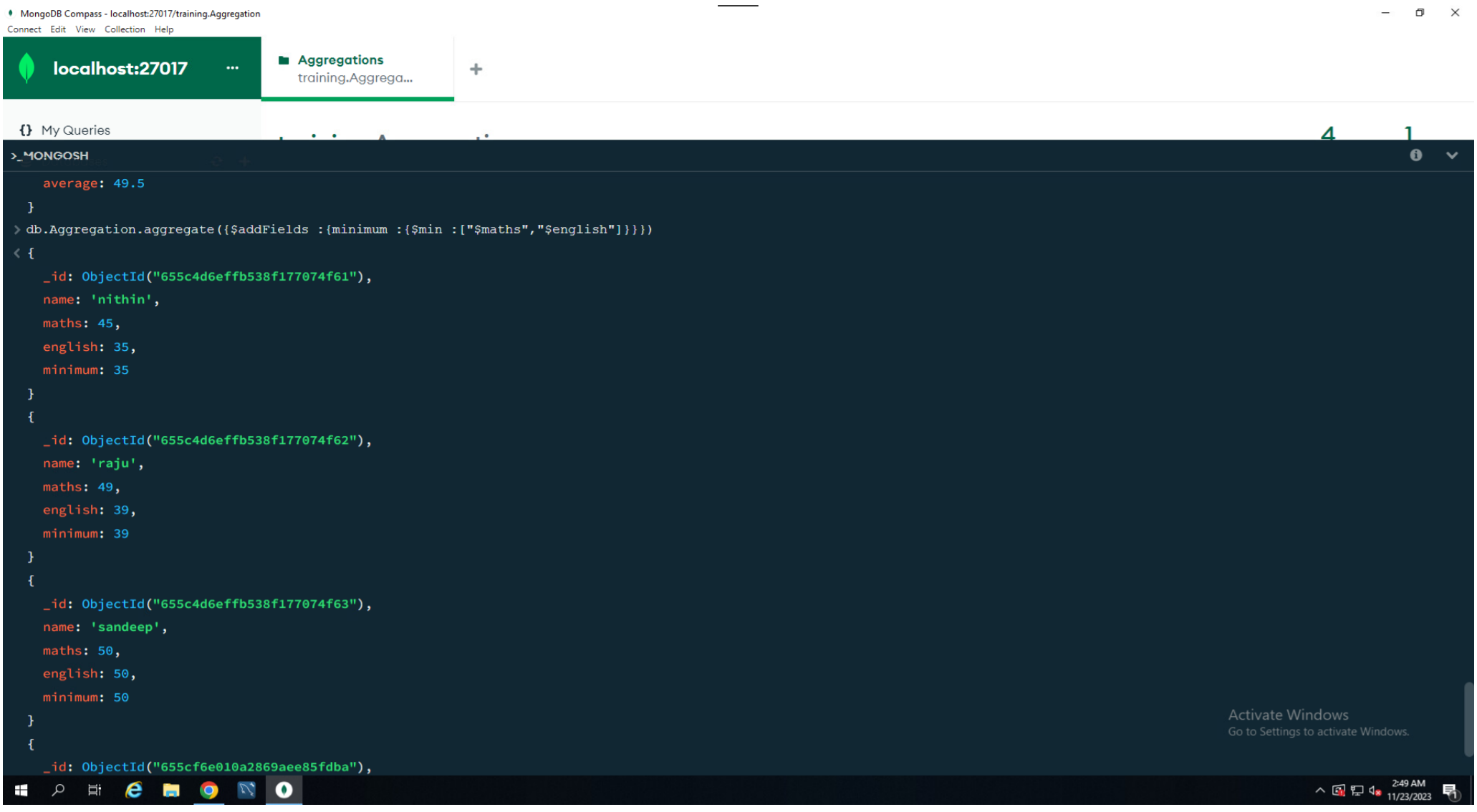


Fig:55

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

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

$addFields $max:

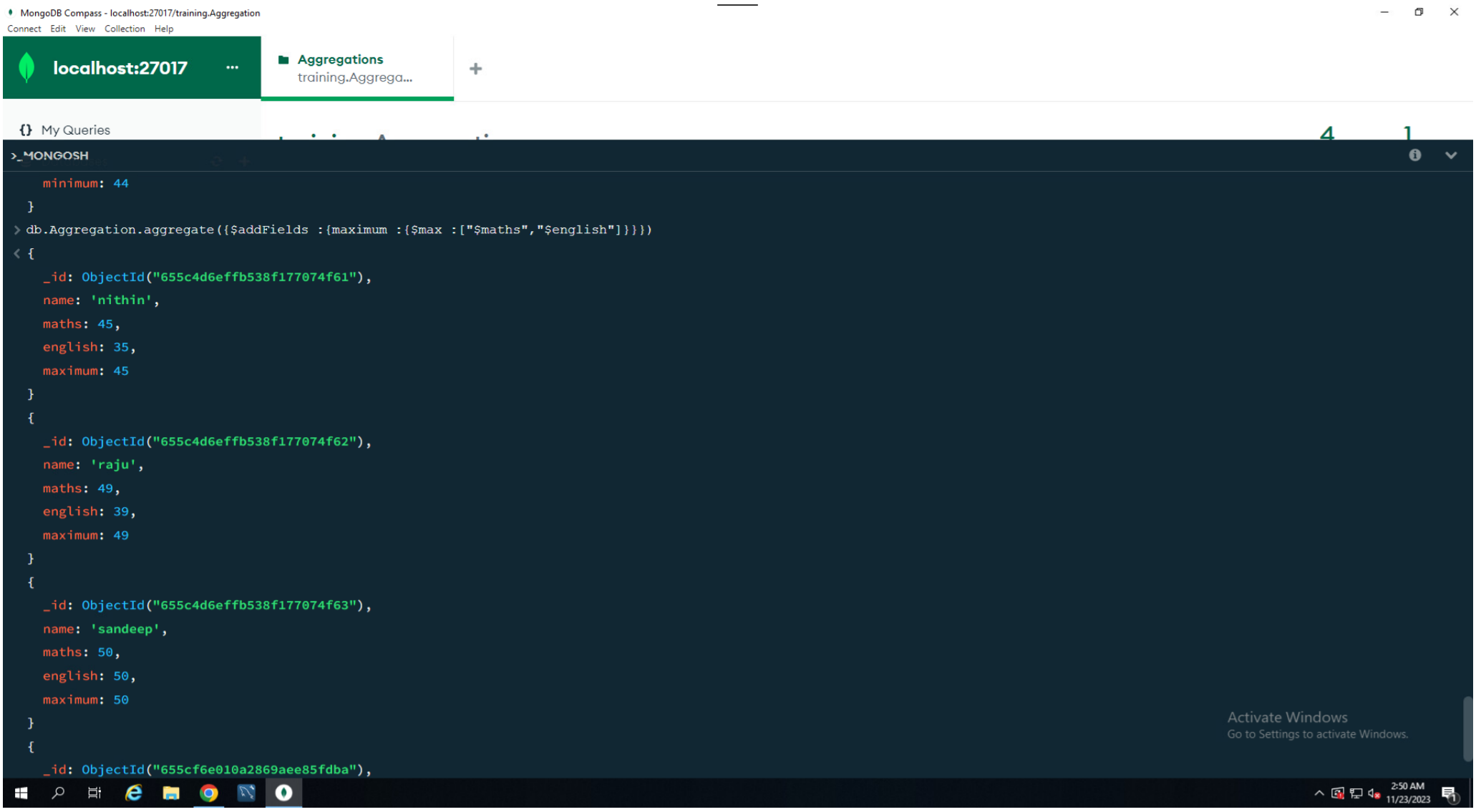


Fig:56

Syntax: db.Collection_name.aggrigate({$addFields :new_field :{$max :[“$field1”,”$field2”]}}})

Ex: db.Aggregation.aggrigate({$addFields:{ maximum_marks :{$max :[“$maths”,”$english”]}}})

$unwind in Mongosh:

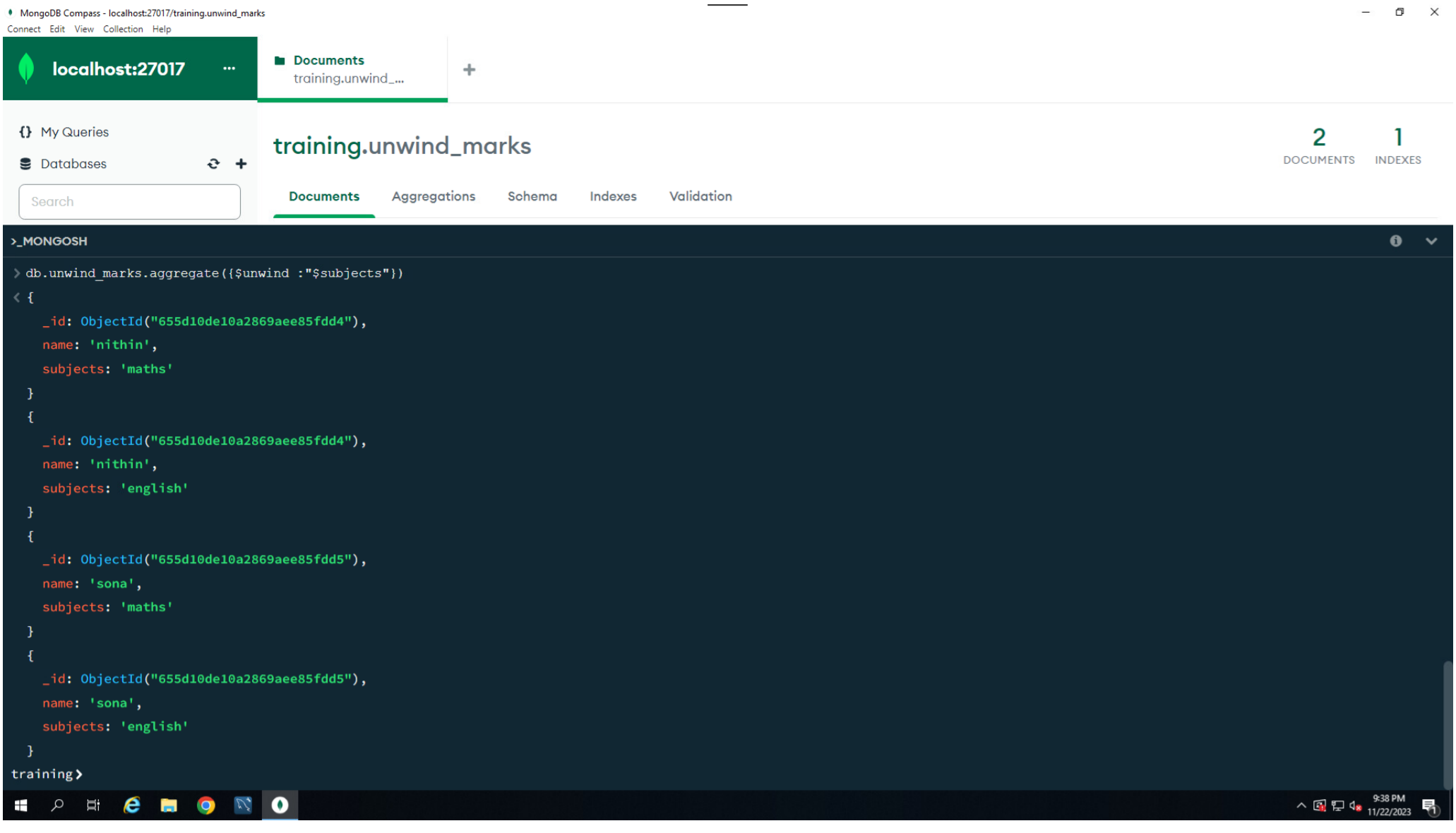


Fig:57

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

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

$Skip in Mongosh:

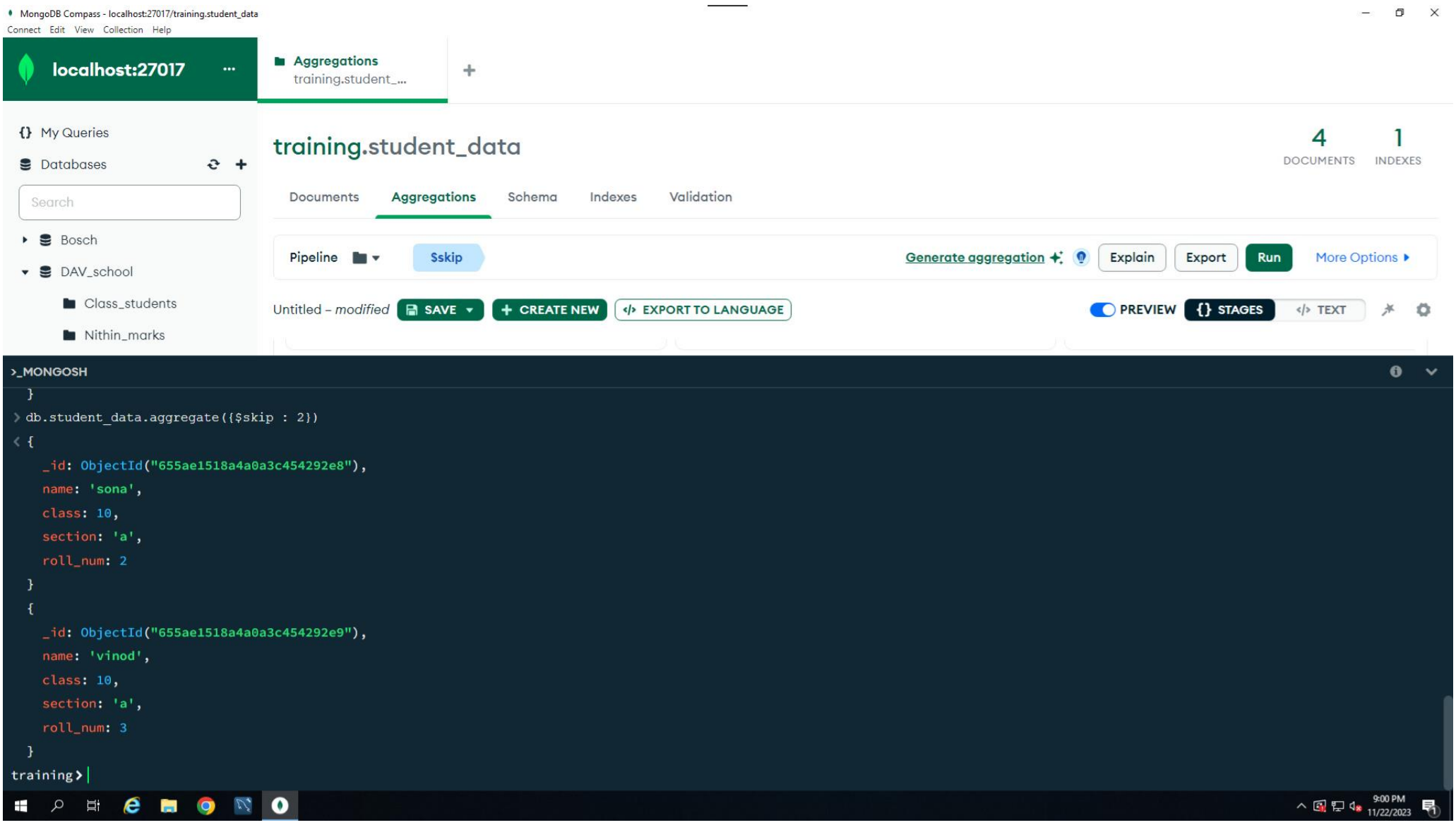


Fig:58

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

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

$limit in Mongosh:

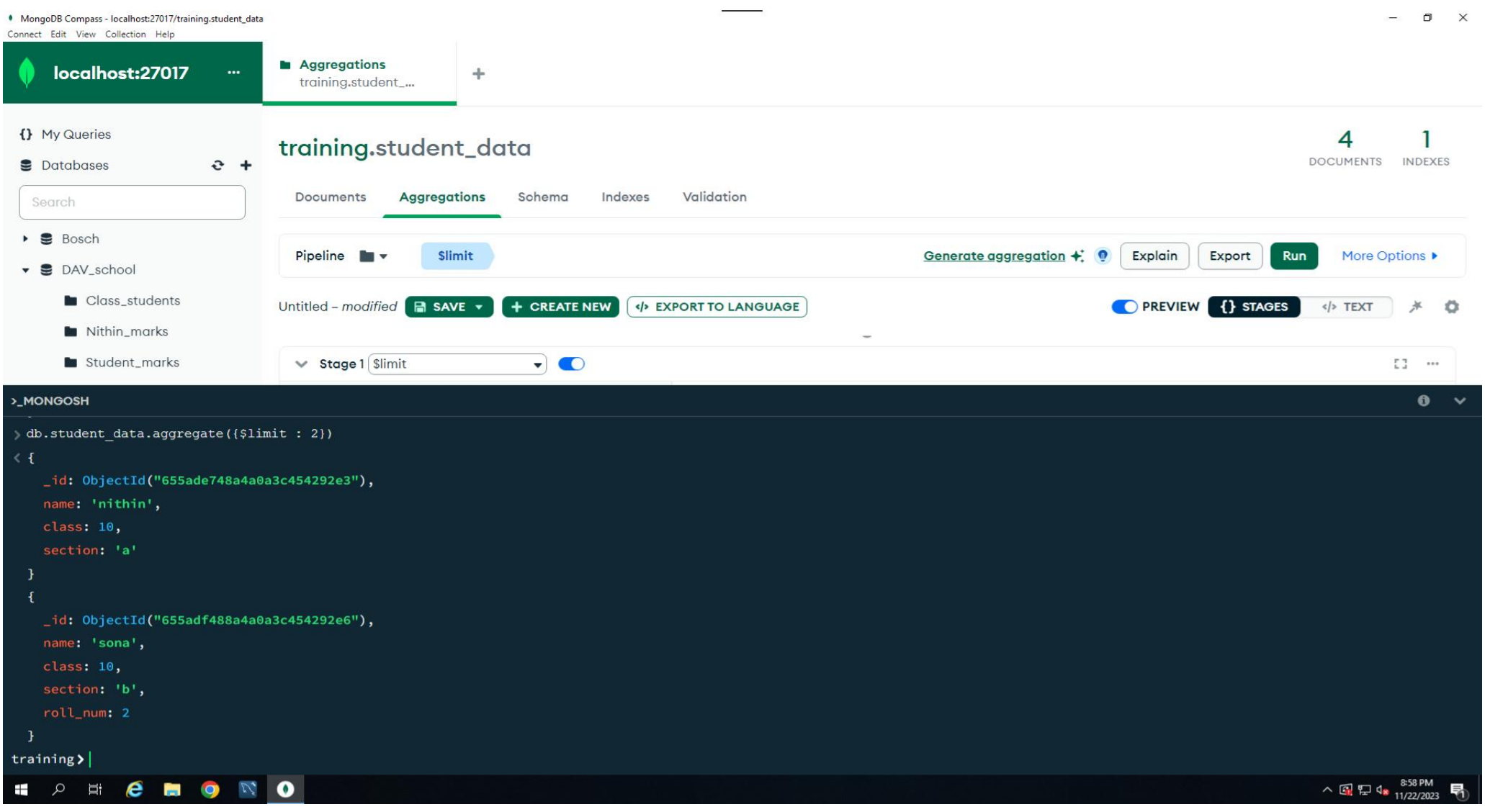


Fig:59

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

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

$group in Mongosh:

$group $sum :

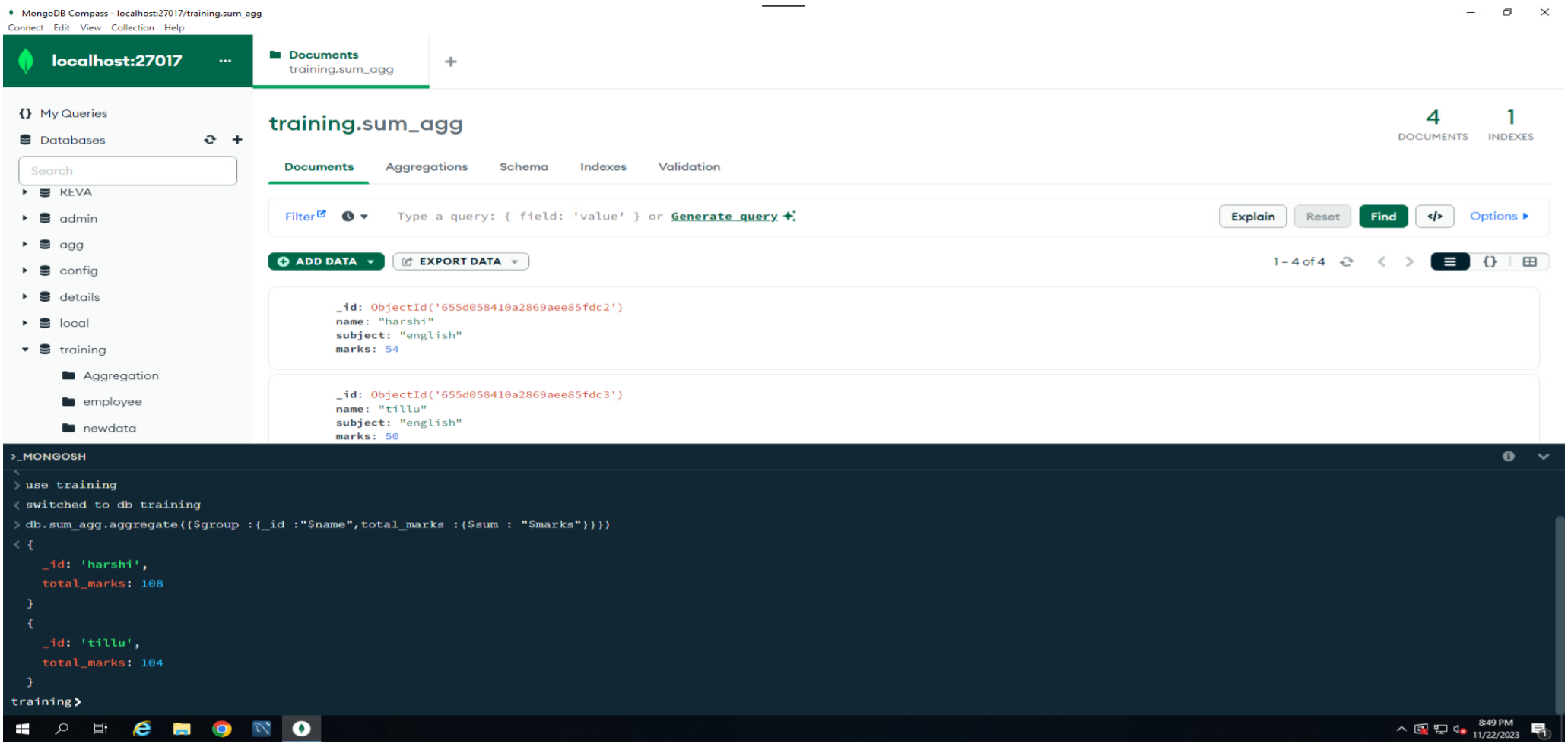


Fig:60

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 :

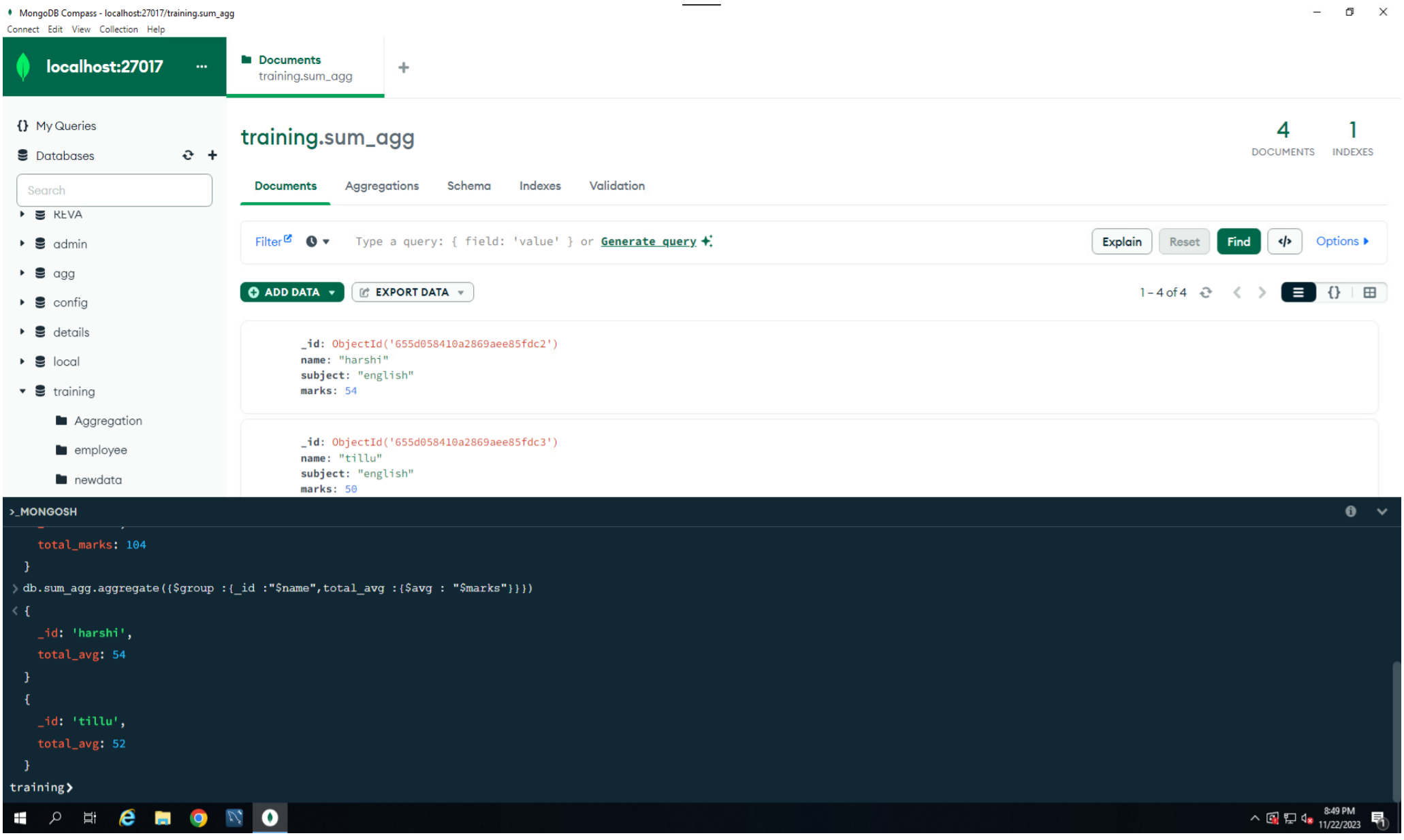


Fig:61

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:

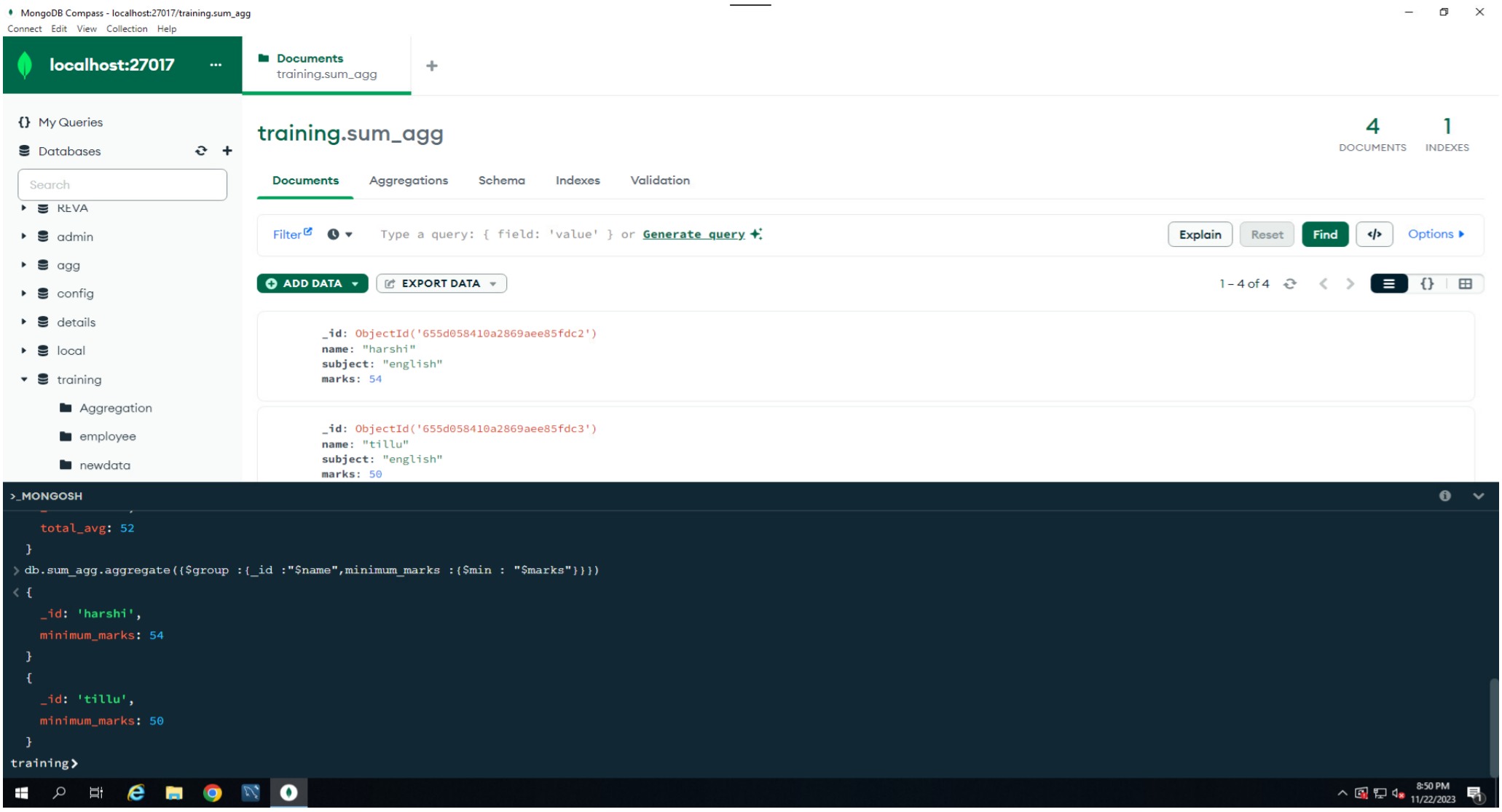


Fig:62

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:

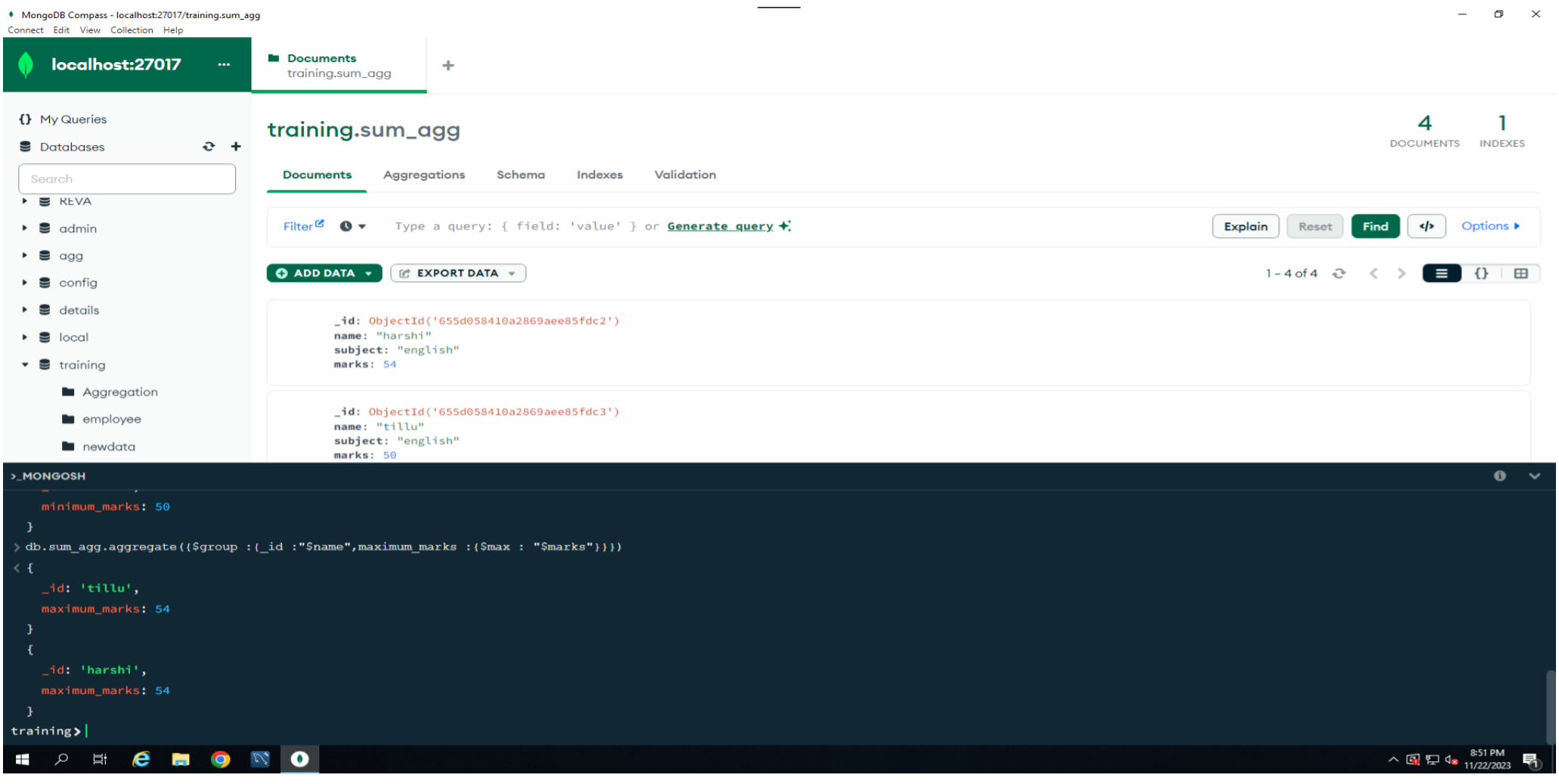


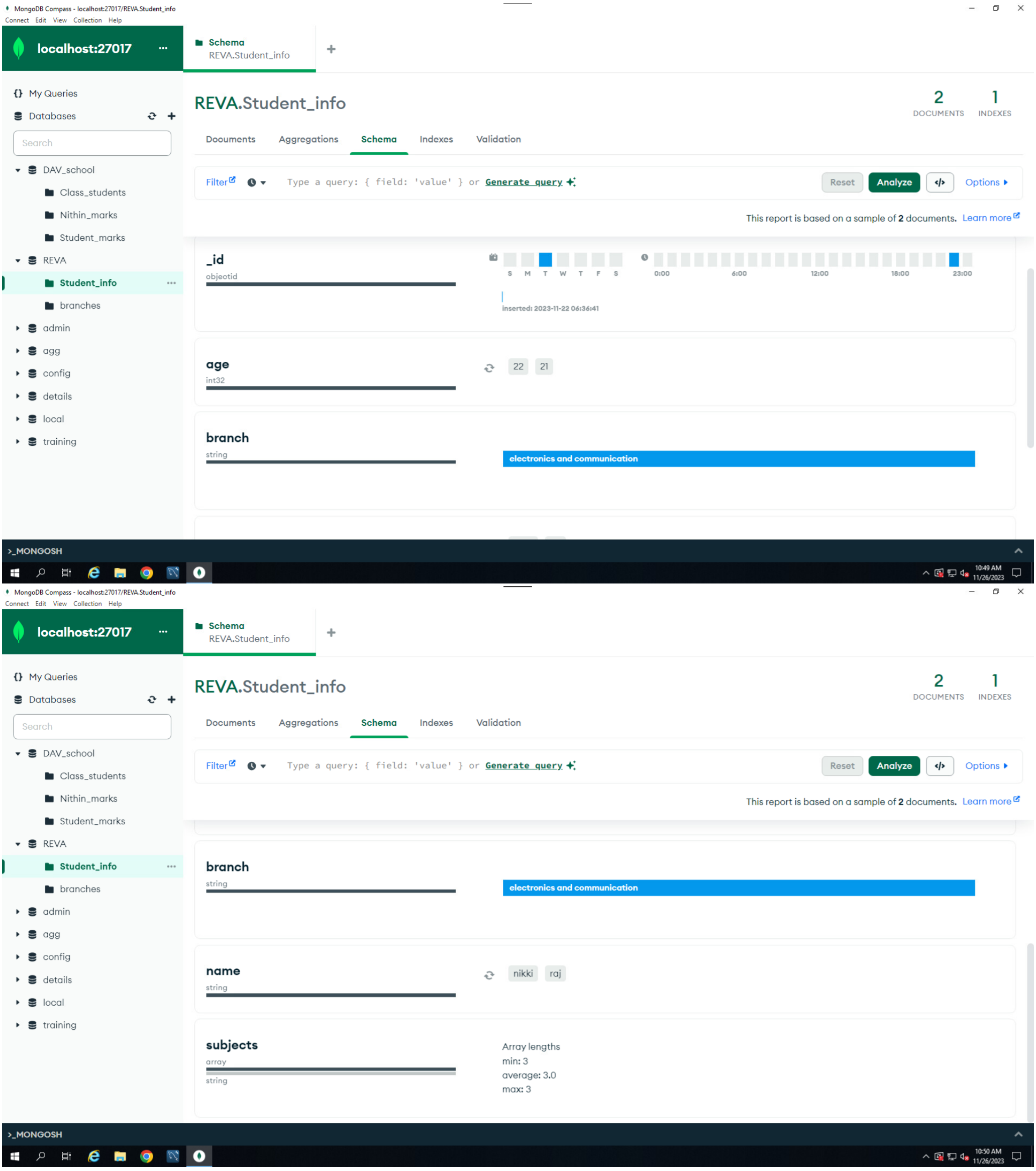
Fig:63

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 crutial tool for structuring, organizing and maintaining the consistency and integrity of data within a collection.



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 constrains, 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.

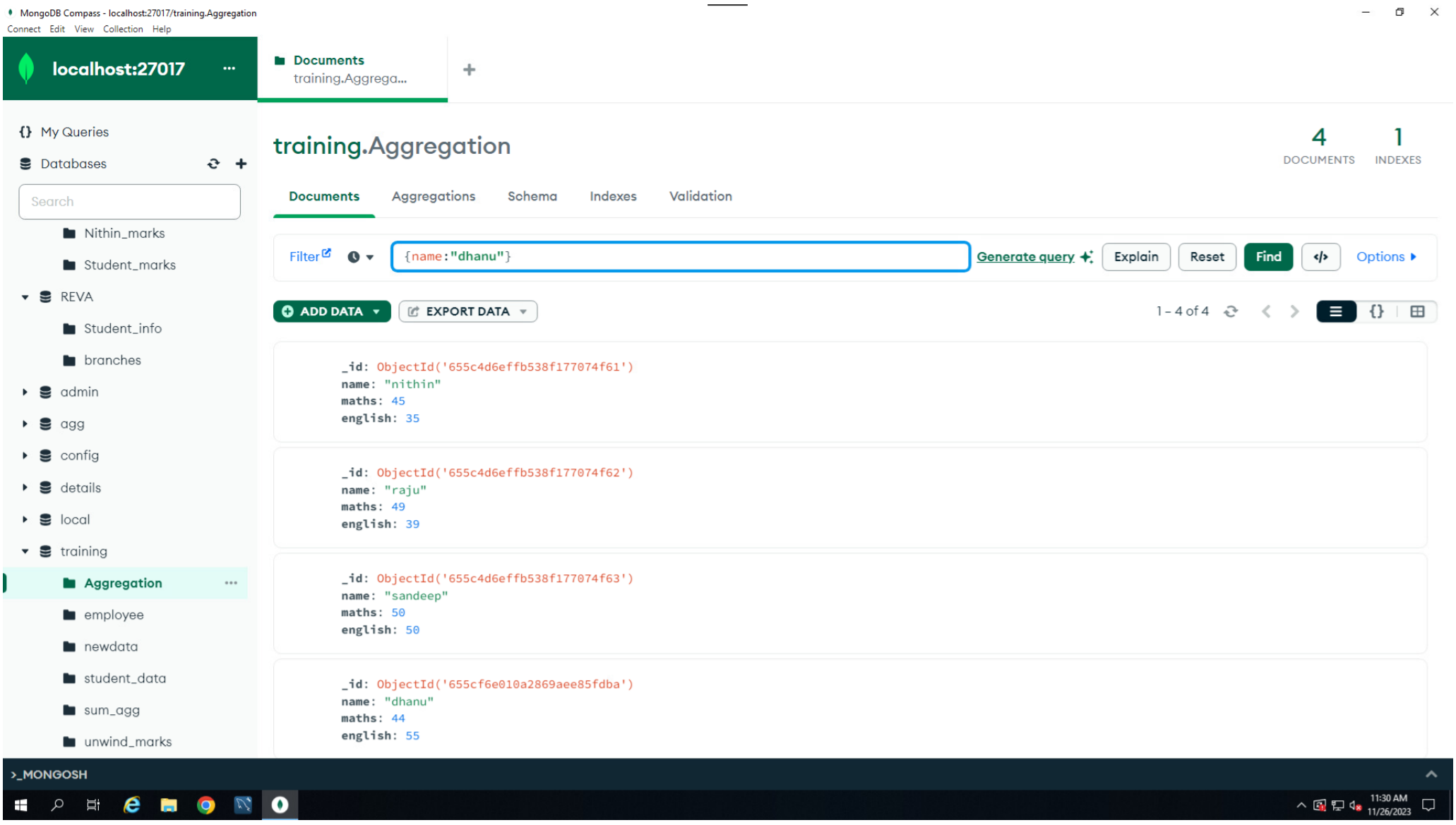


Fig:66

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

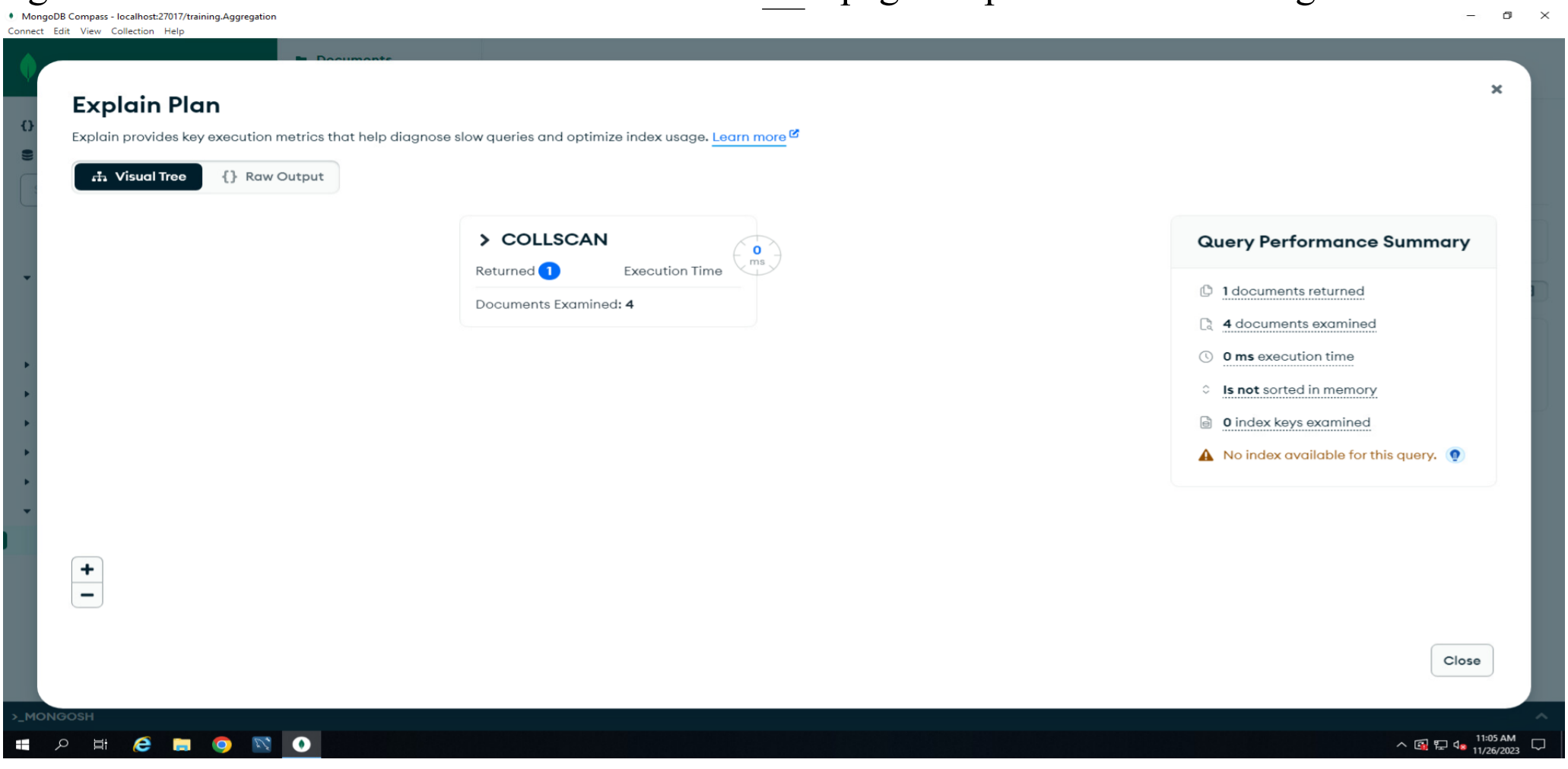


Fig:67

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

Creating an Index in MongoDB compass:

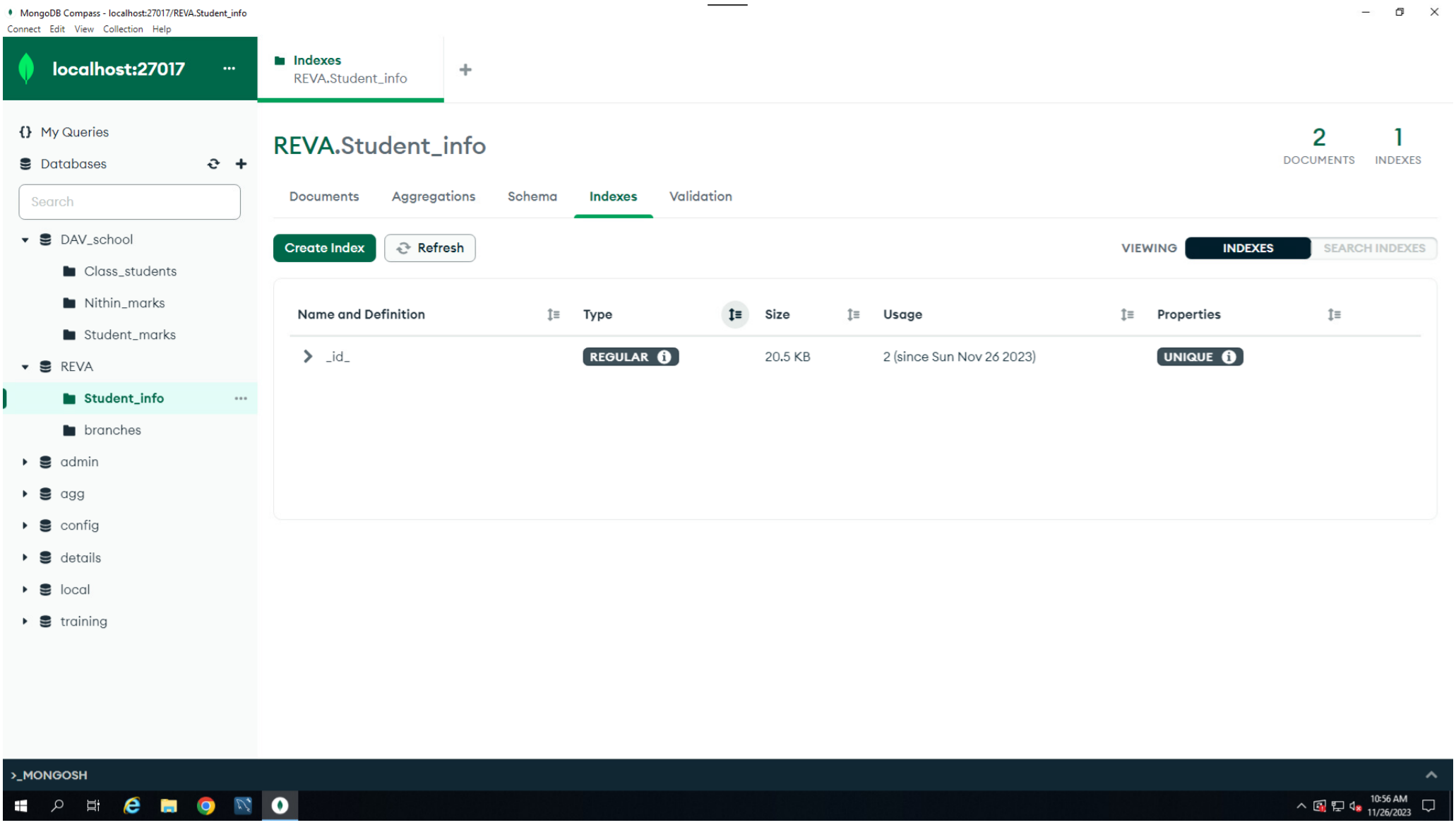


Fig:68

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

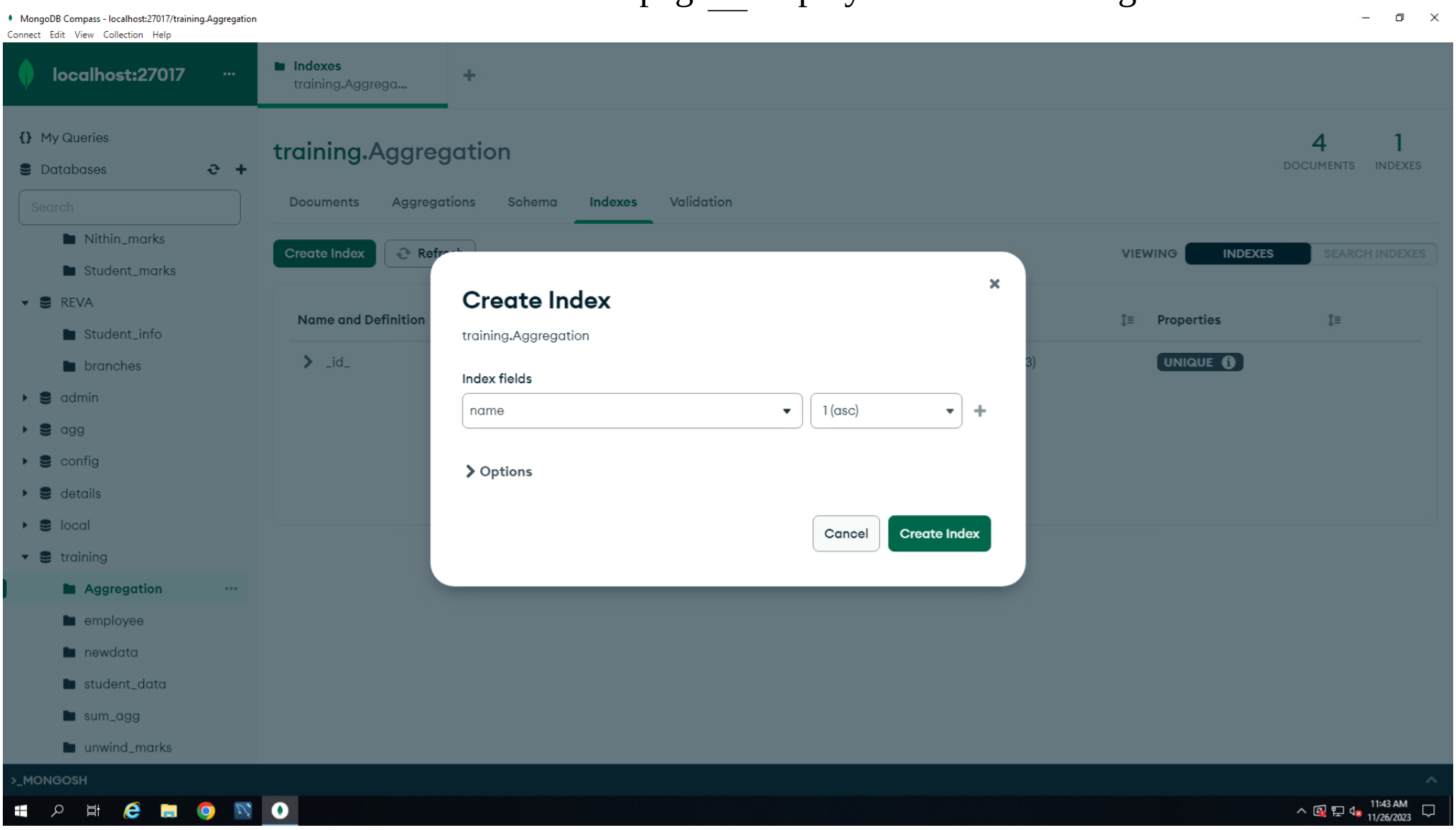


Fig:69

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.

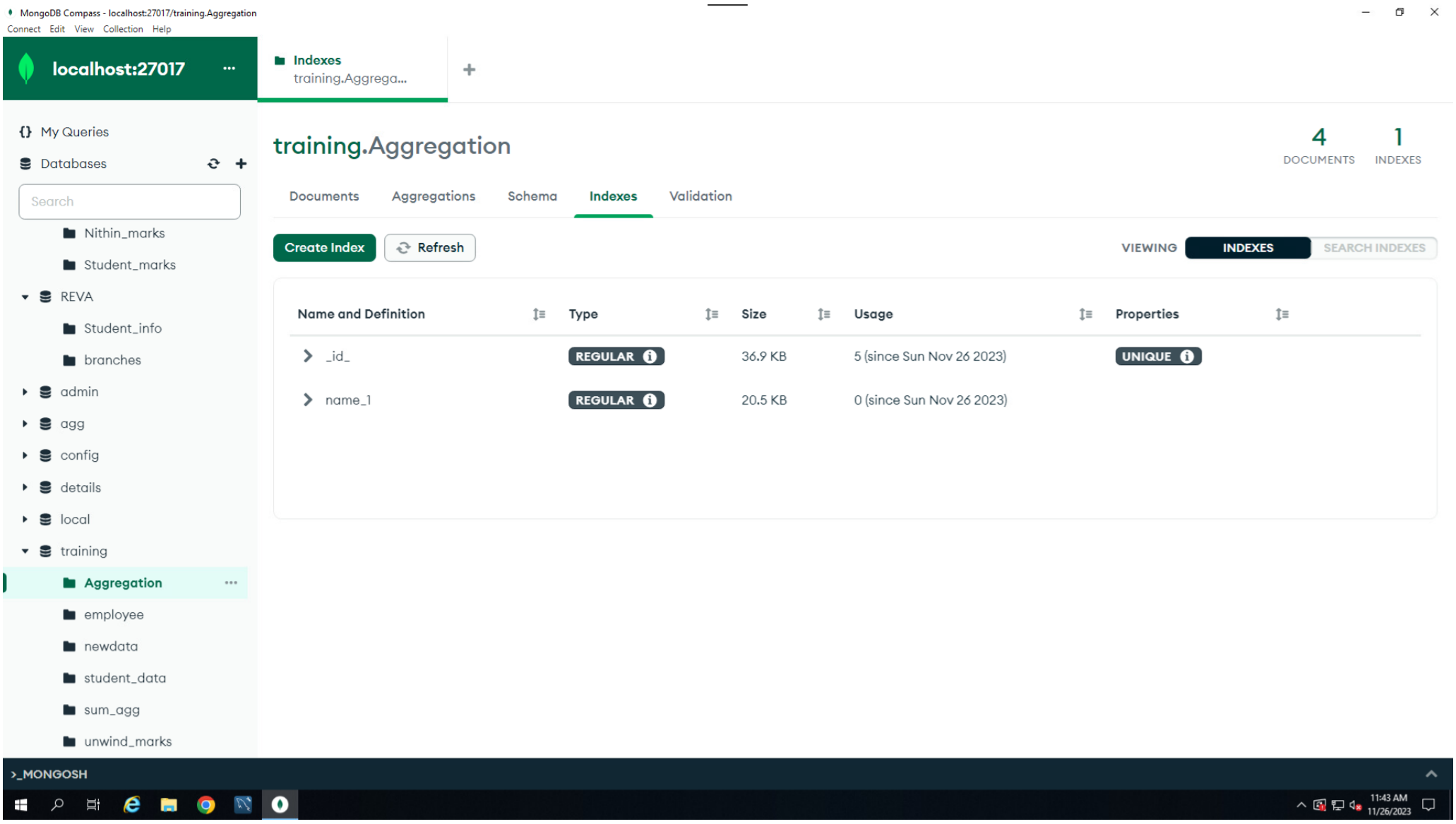


Fig:70

Name_1 index is created as shown in fig70.

Follow the process as shown in fig66.

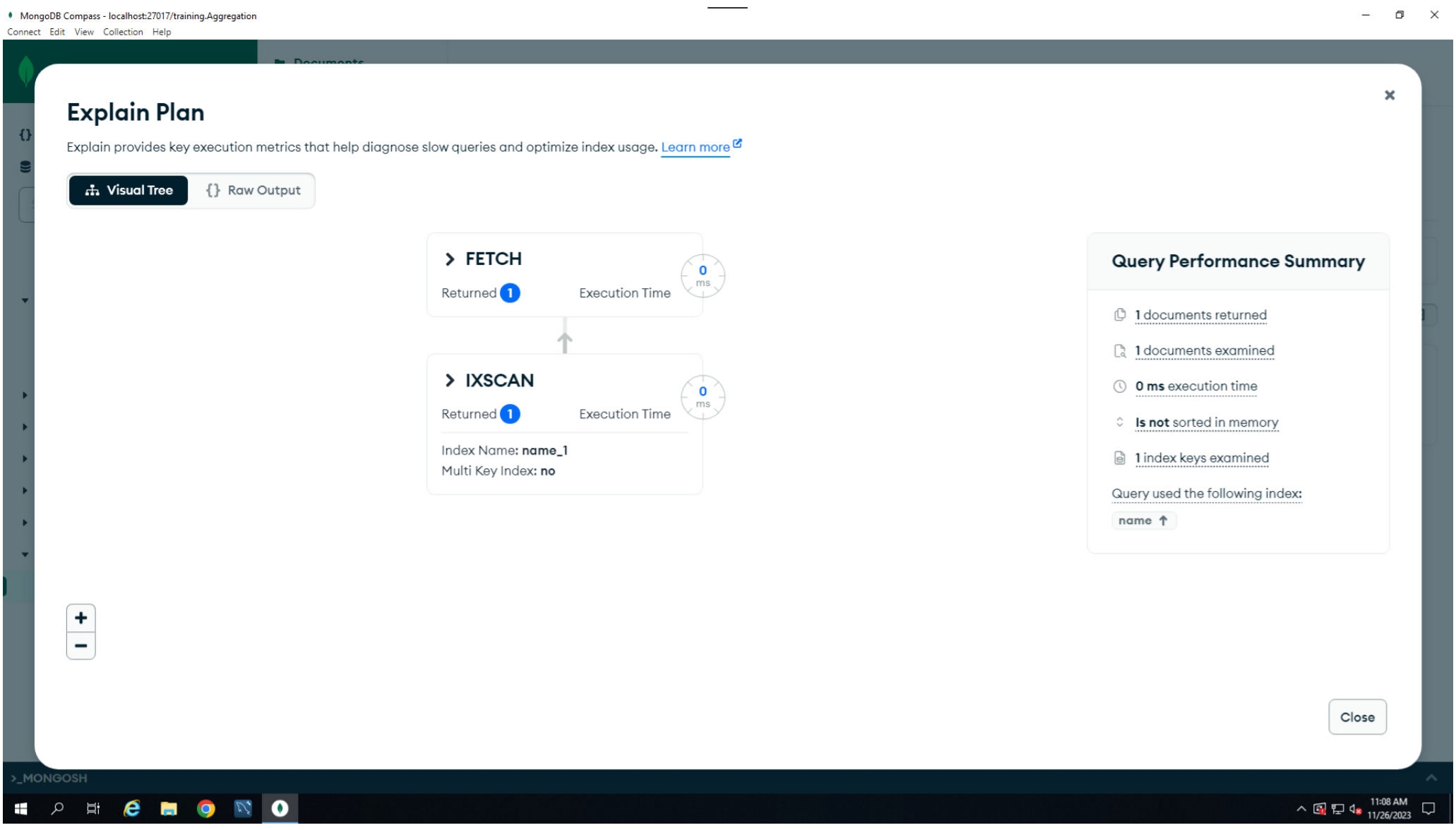


Fig:71

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

Dropping an Index in MongoDB compass:

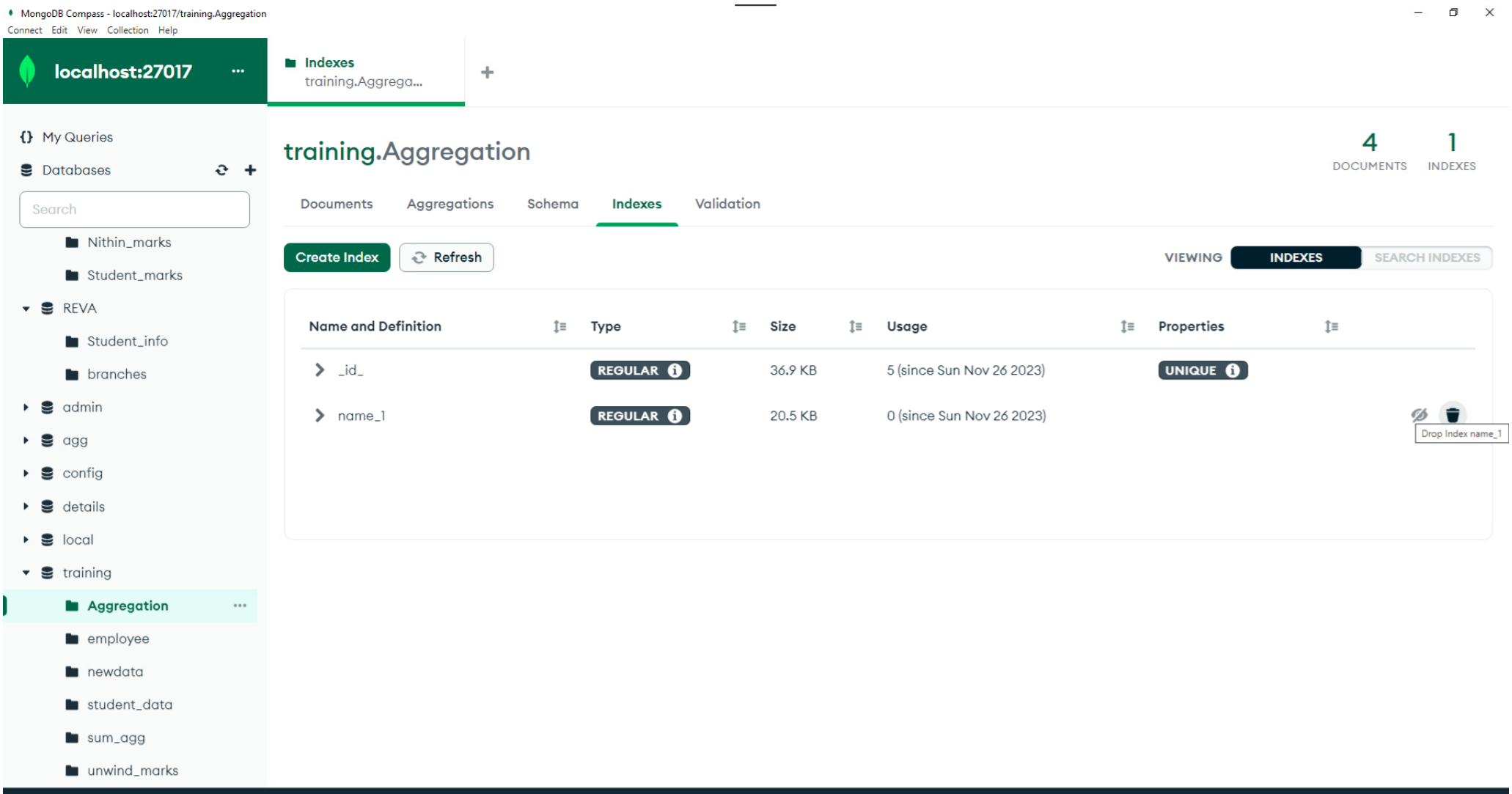


Fig:72

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

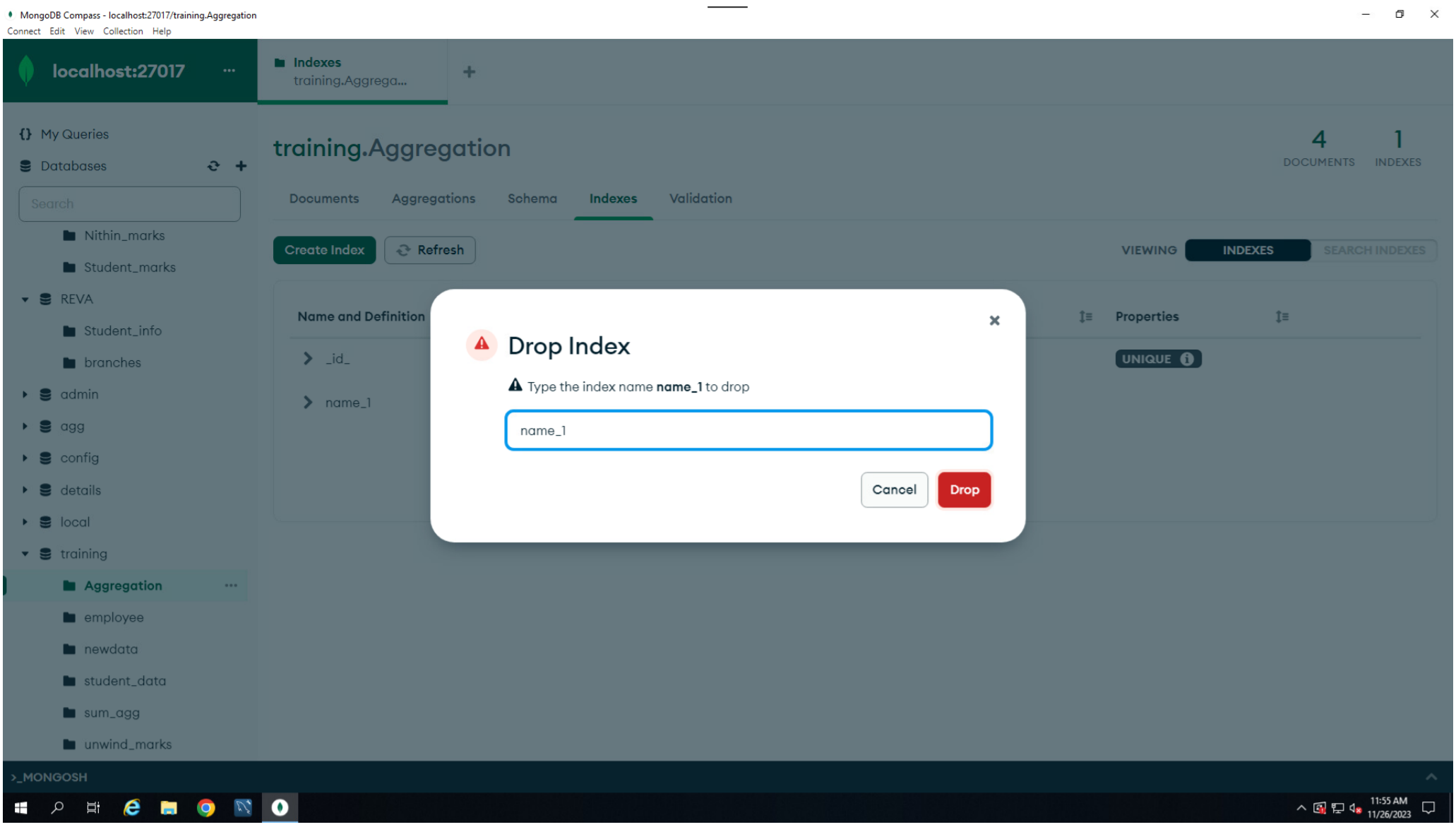


Fig:73

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

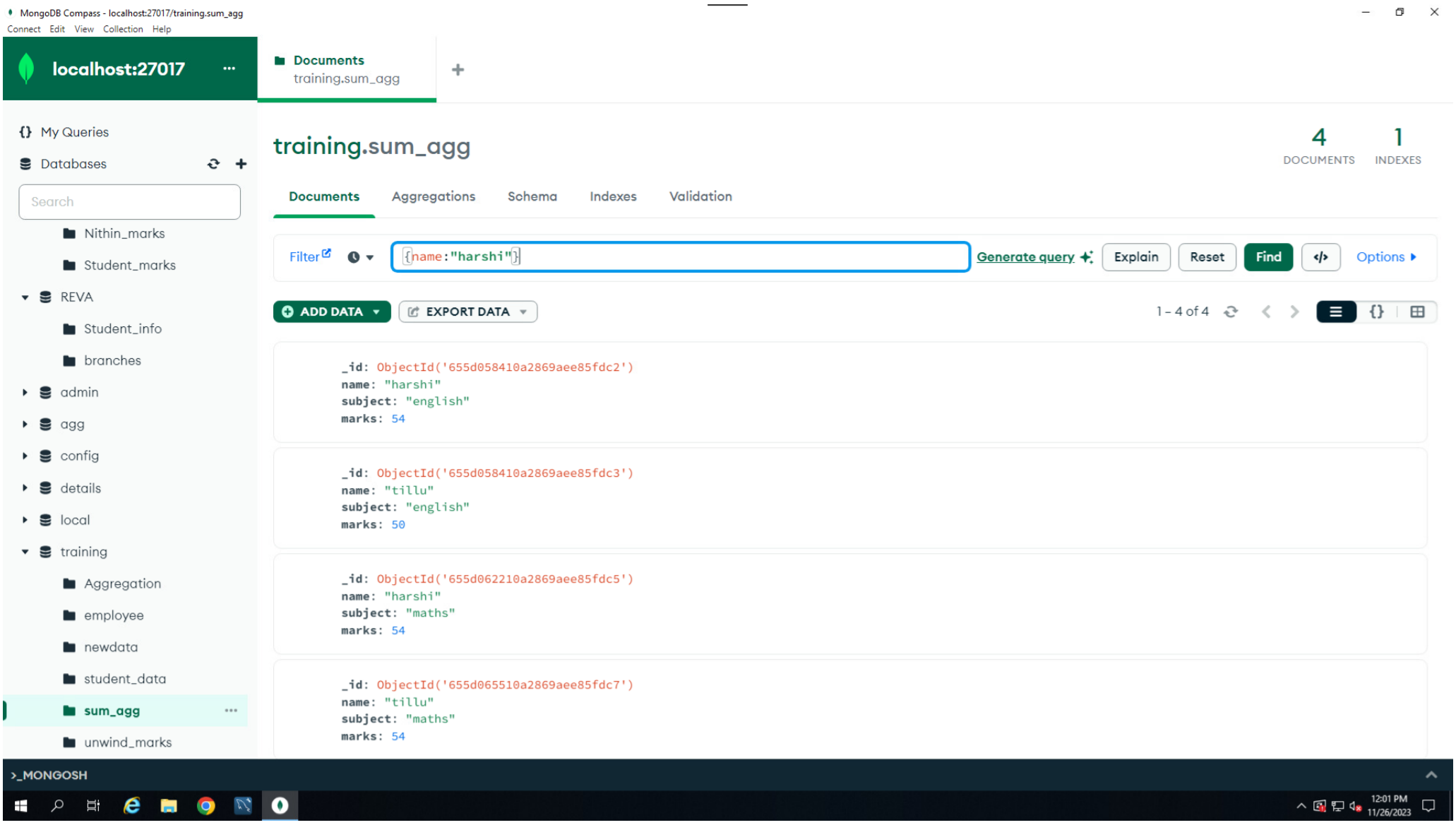


Fig:74

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

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

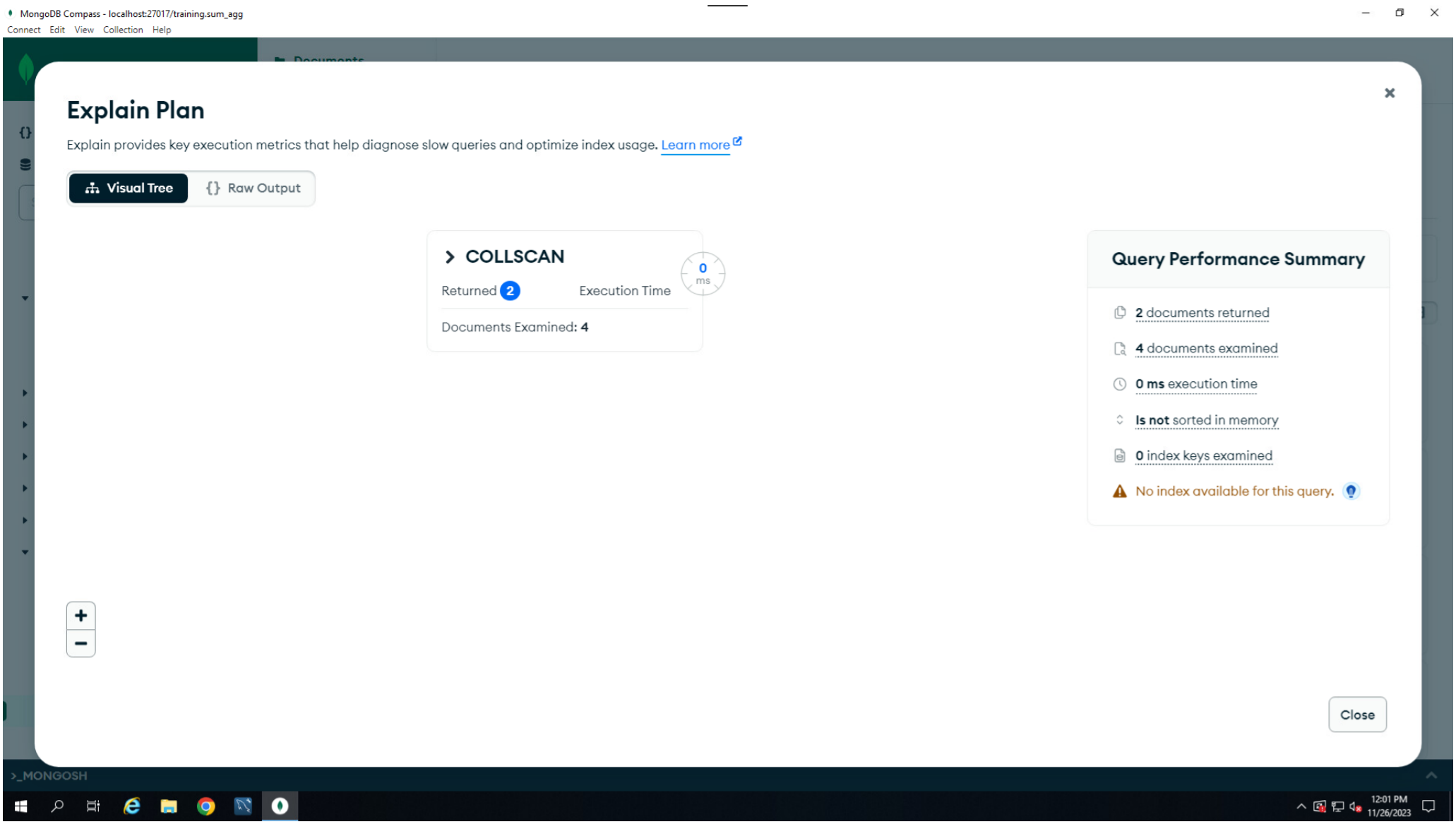


Fig:75

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

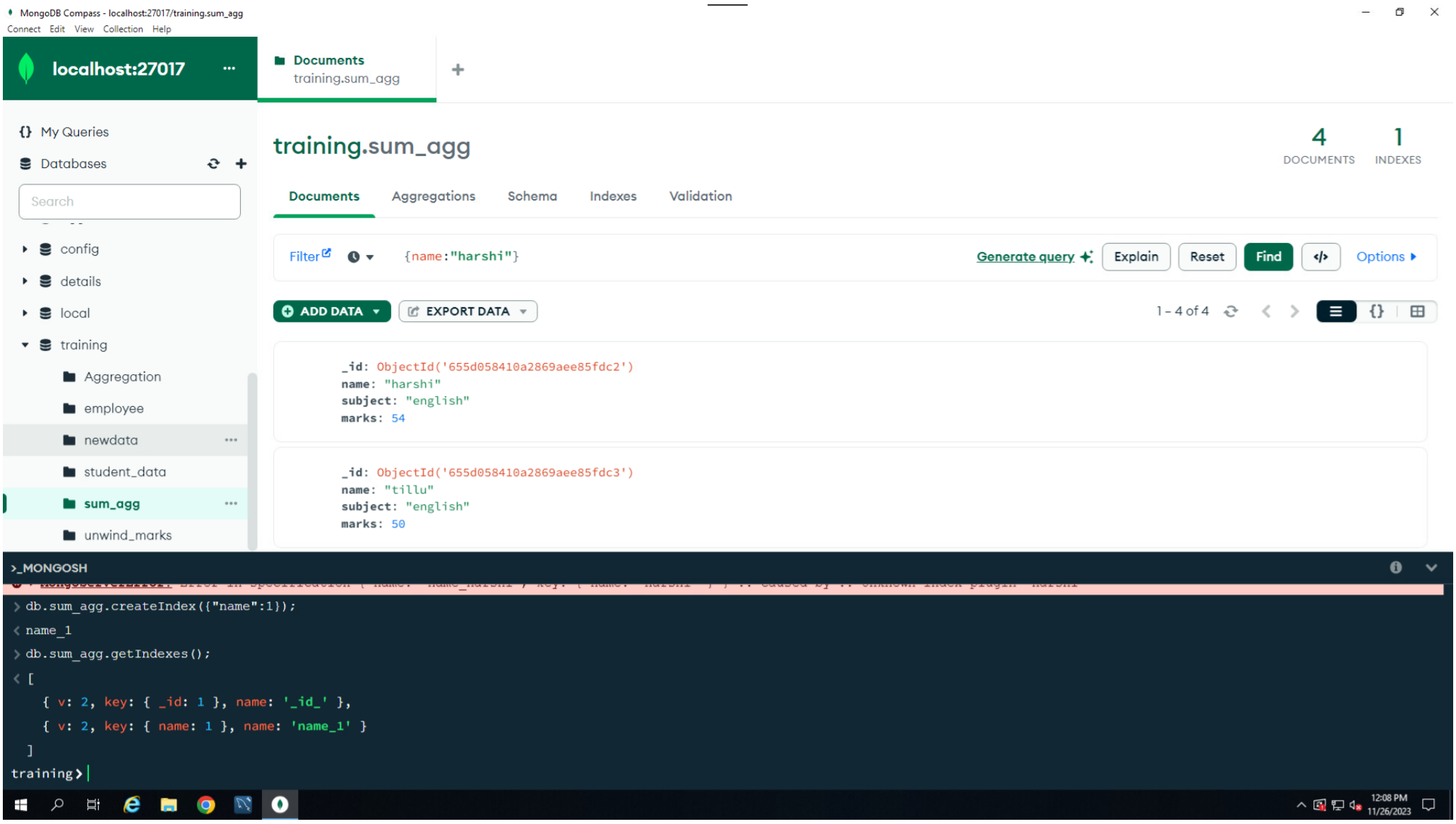


Fig:76

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.

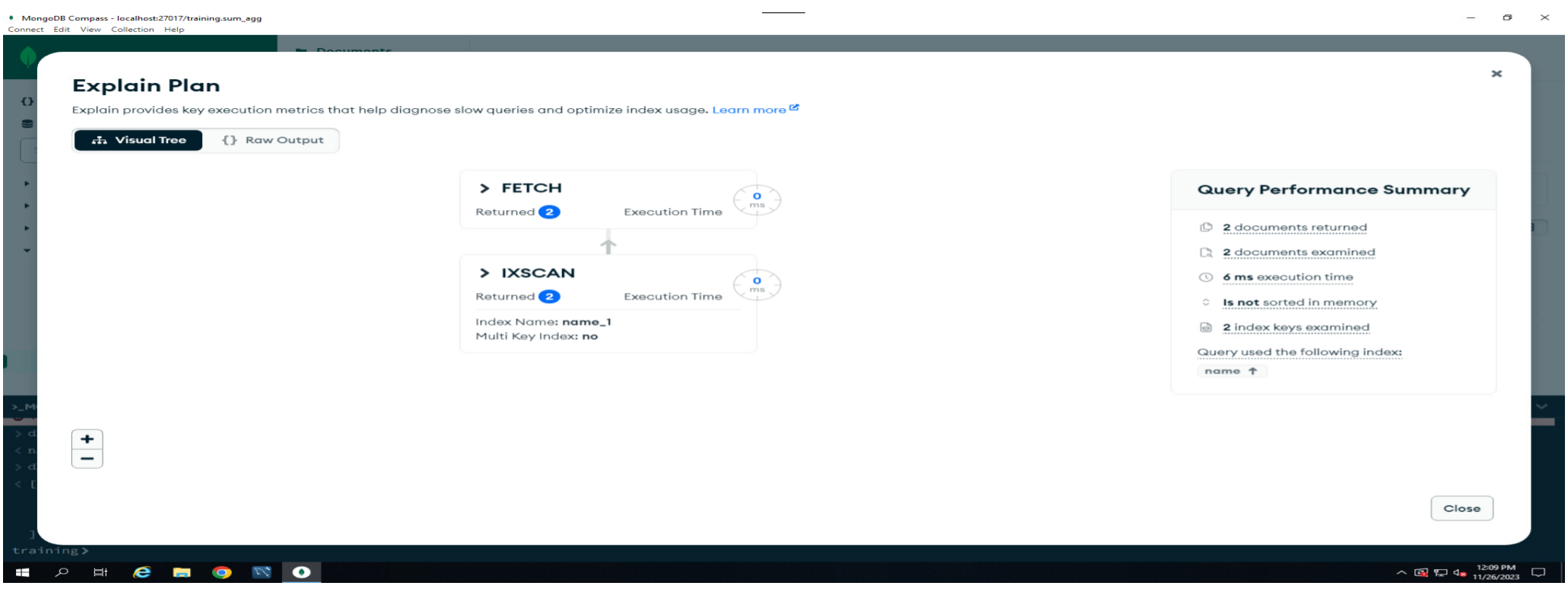


Fig:77

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

Creation of Index in mongosh: Compound Index.

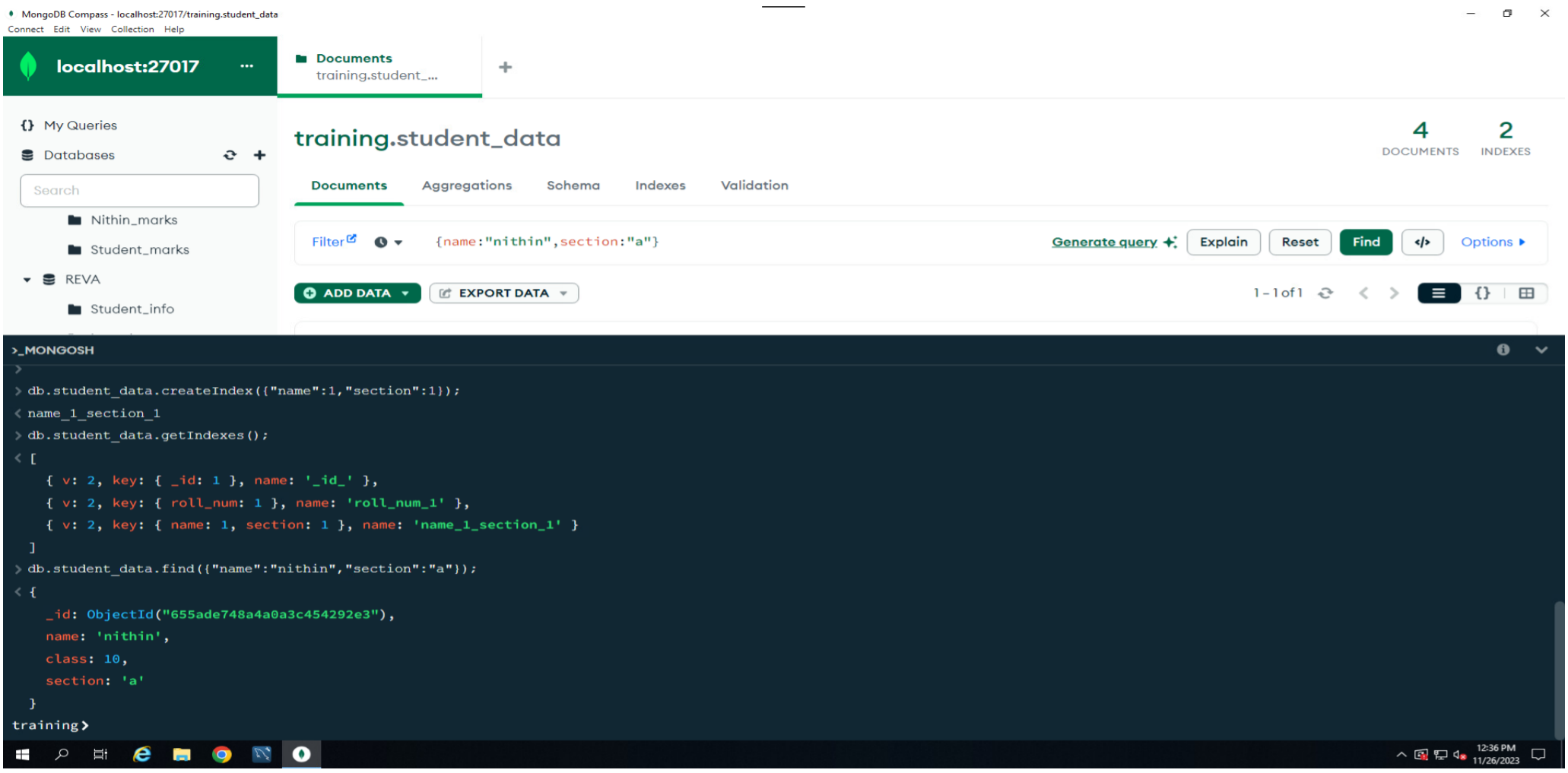


Fig:78

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});

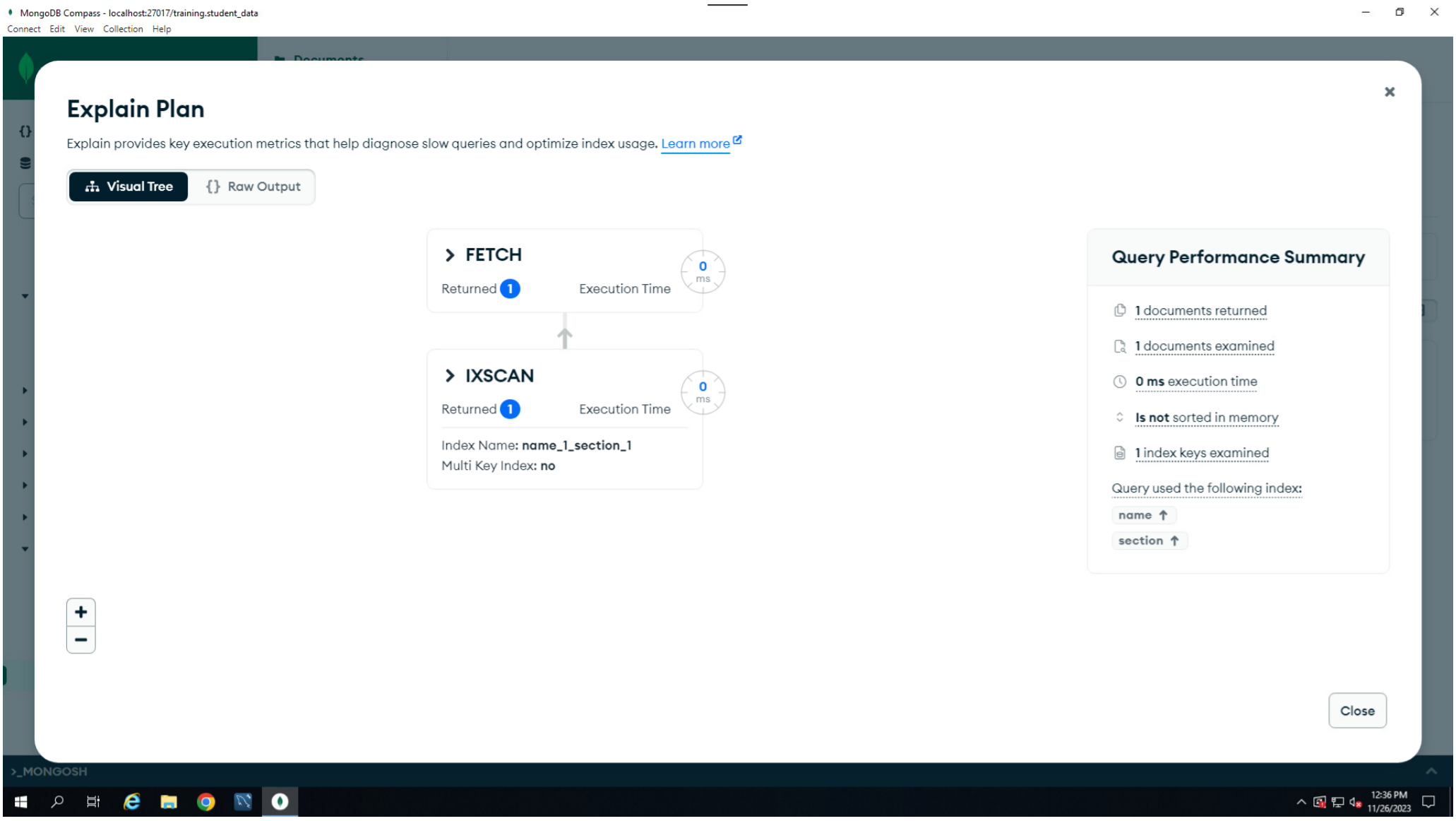


Fig:79

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.

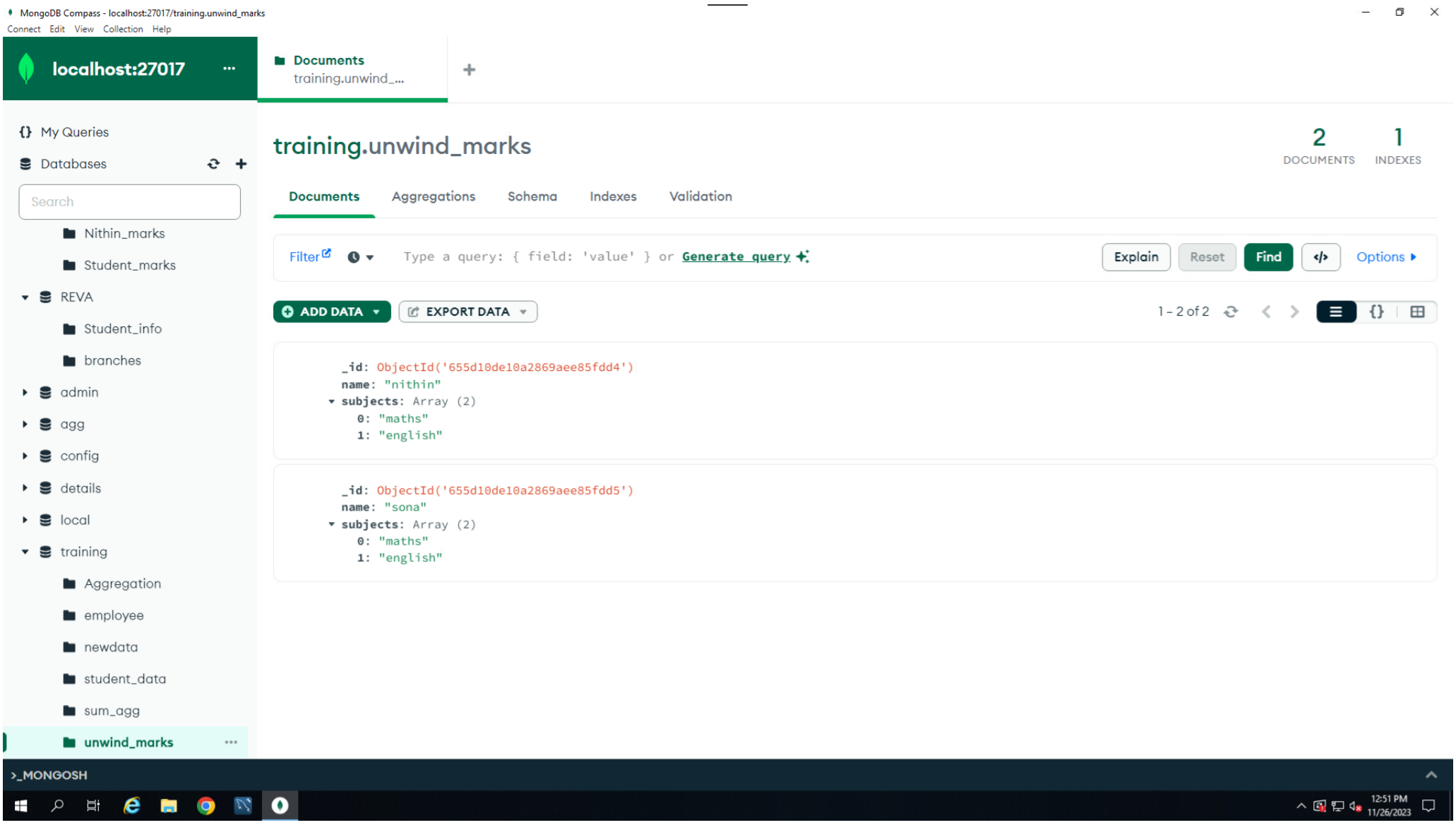


Fig:80

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});

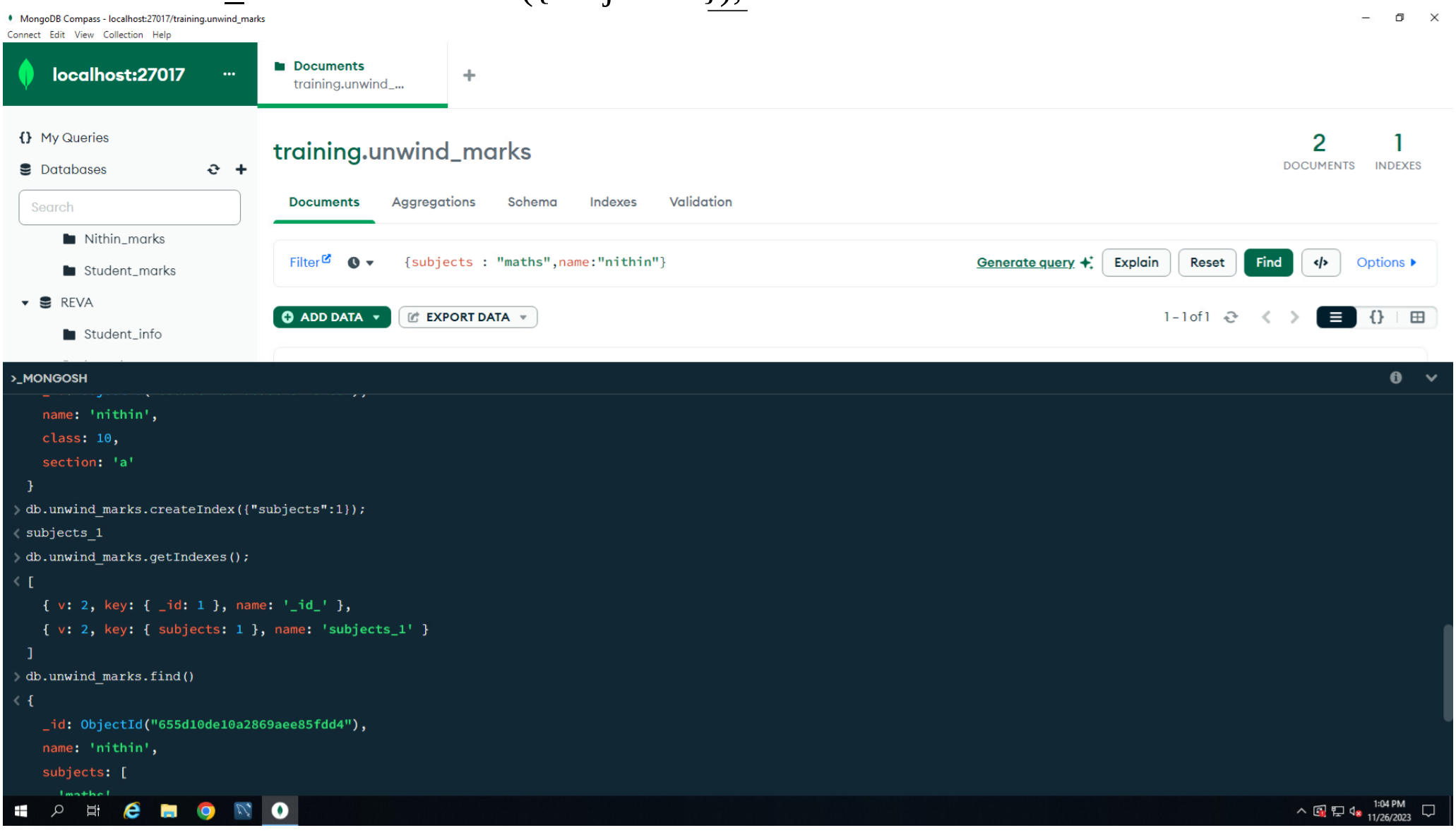


Fig:81

Drop an Index in Mongosh:

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

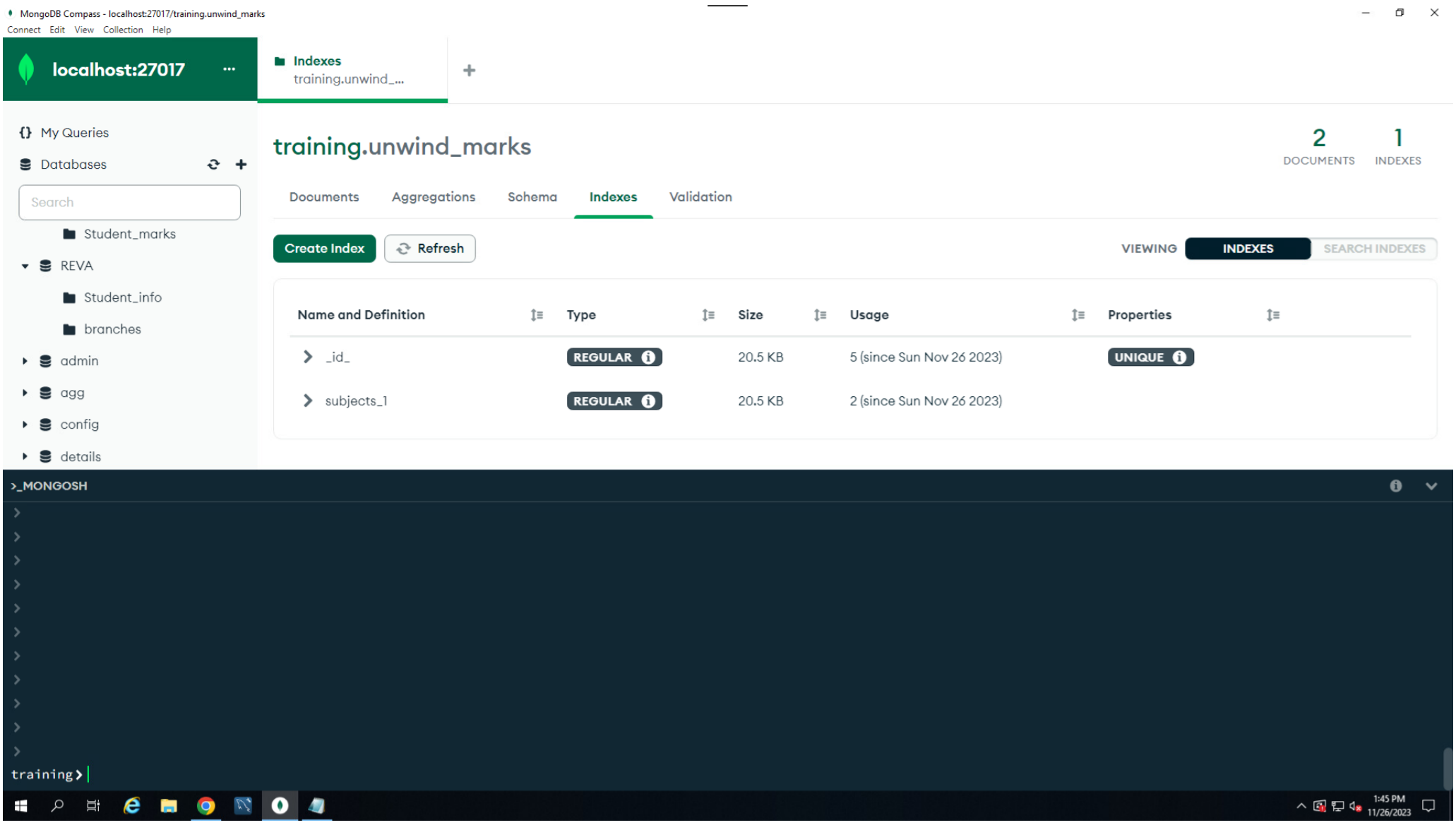


Fig:82

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

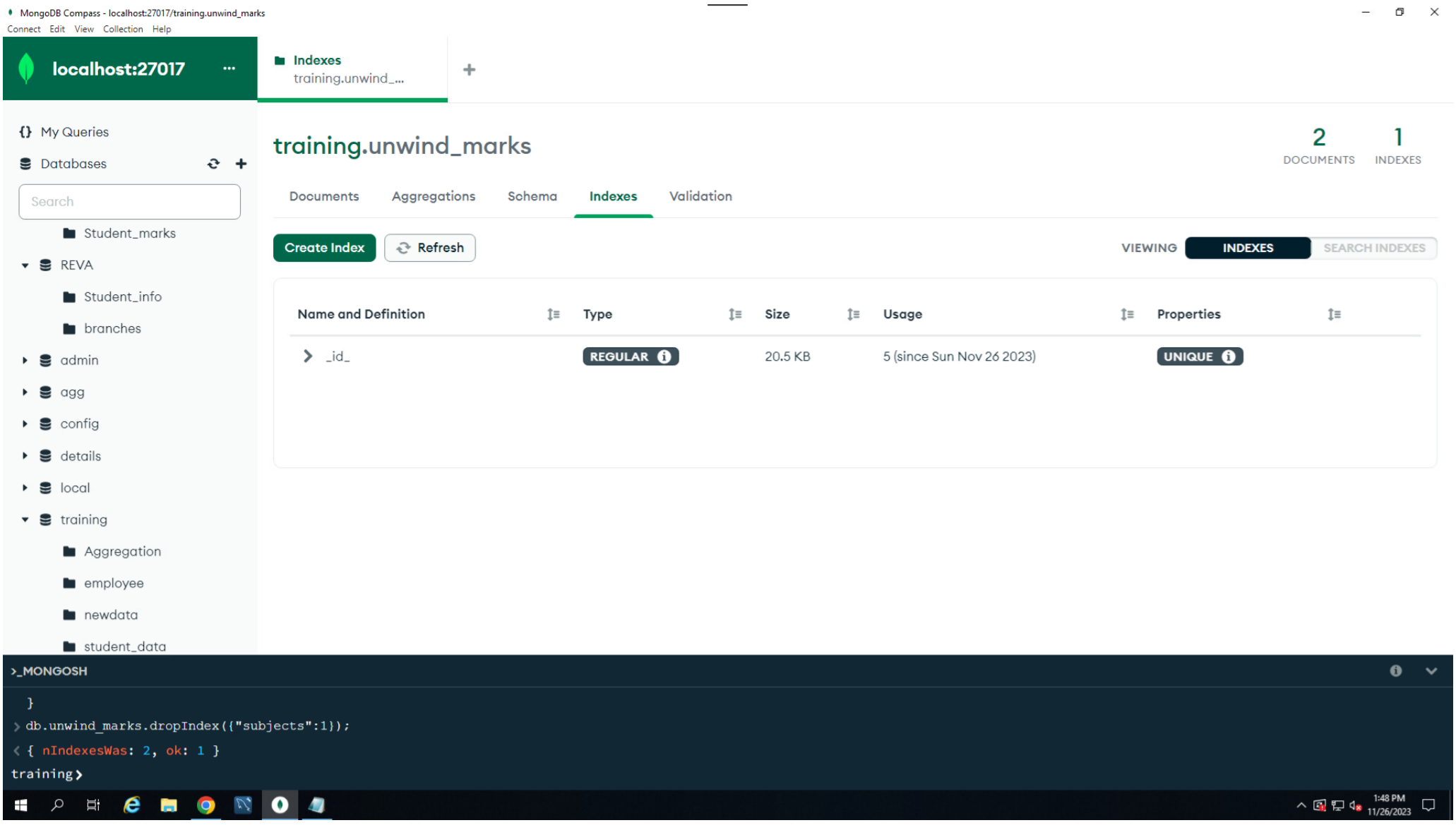


Fig:83

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.

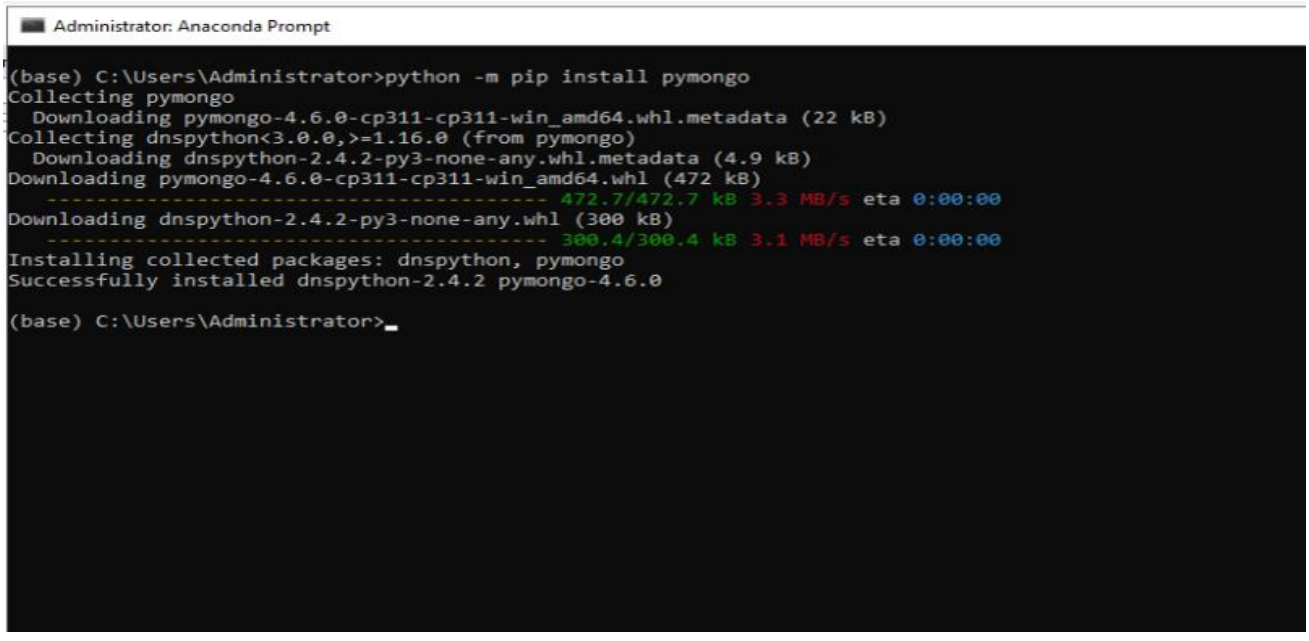
PyMongo, the standard **MongoDB driver library for Python**, is easy to use and offers an intuitive API for accessing databases, collections, and documents. Objects retrieved from MongoDB through PyMongo are compatible with dictionaries and lists, so we can easily manipulate, iterate, and print them.

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.



- 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.

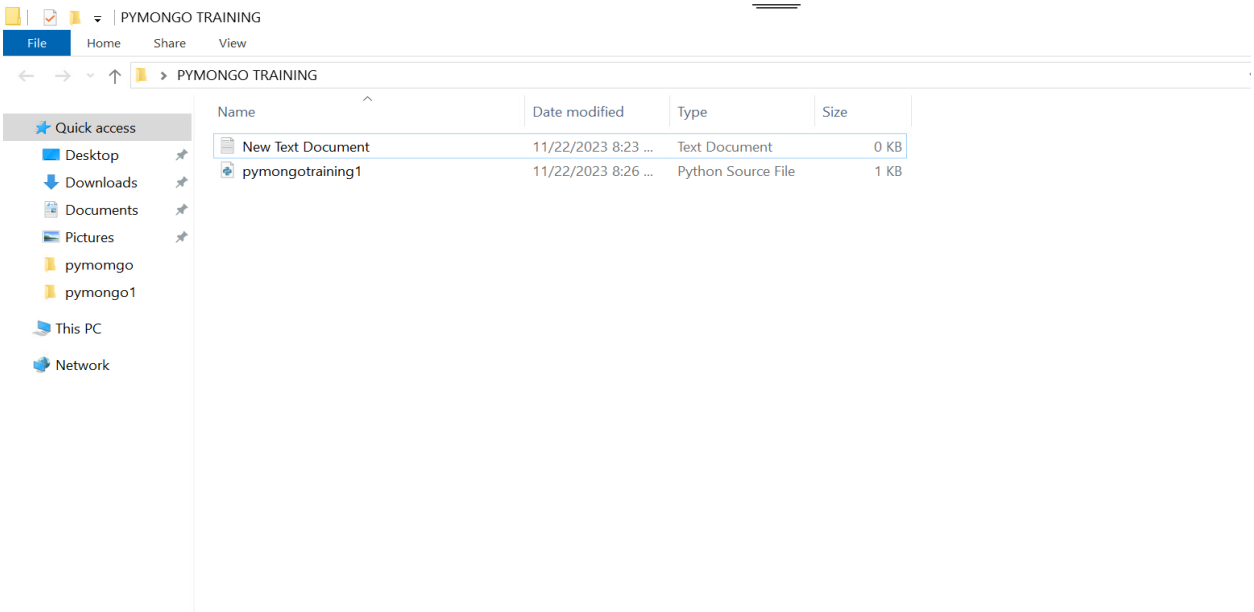
- The basic Syntax to write code in a created file.

  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:

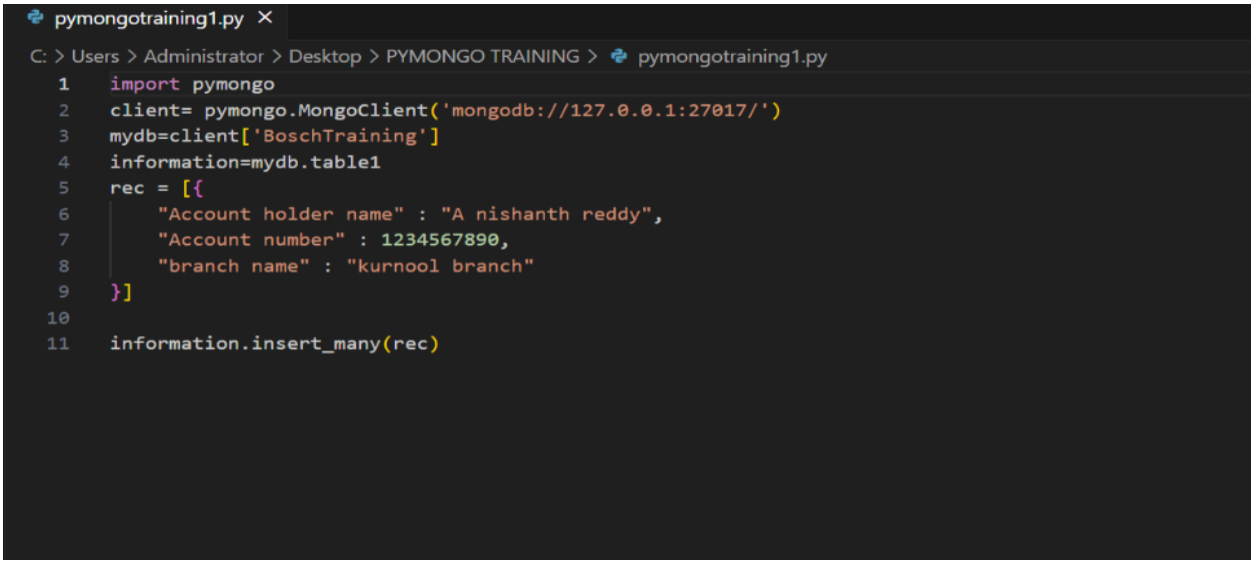
Here I have created a new folder in the name of PYMONGOTRAINING in that folder. I have created a new file like pymonogtraining1 text document and then I had written code to create database called "BOSCH TRAINING".

Create a pymongotraining1.py

Code:

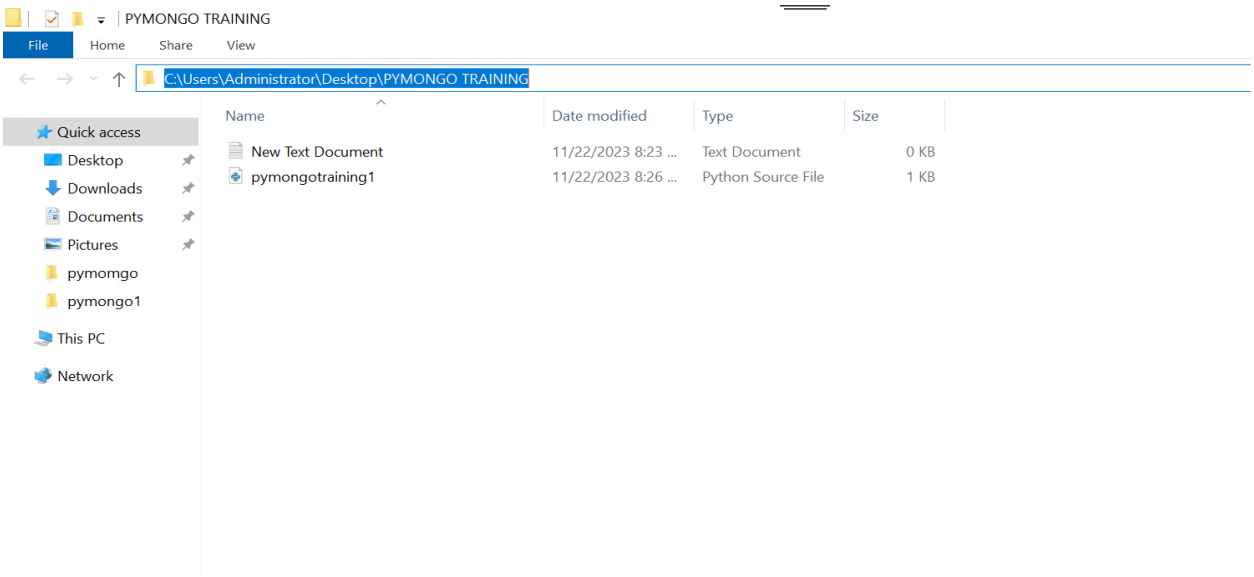
```
Import pymongo
client= pymongo.mongoclient('mongodb://127.0.0.1:27017/')
mydb=client['Bosch Training']
information=mydb.table1
rec = [{
    "Account holder name" : "A Nishanth Reddy",
    "Account number" : 1234567890,
    "branch name" : "Kurnool branch"
     }]
Information.insert_many(rec)
```

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.

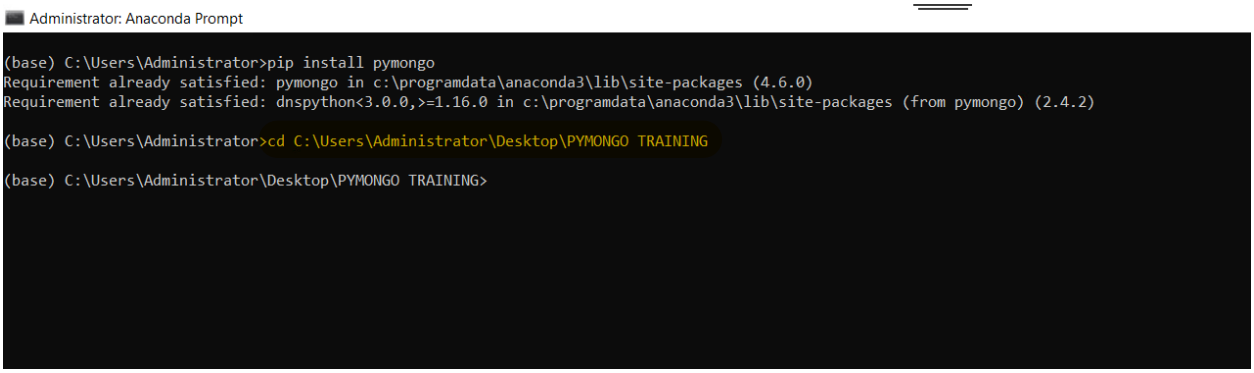


Step 2:

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.



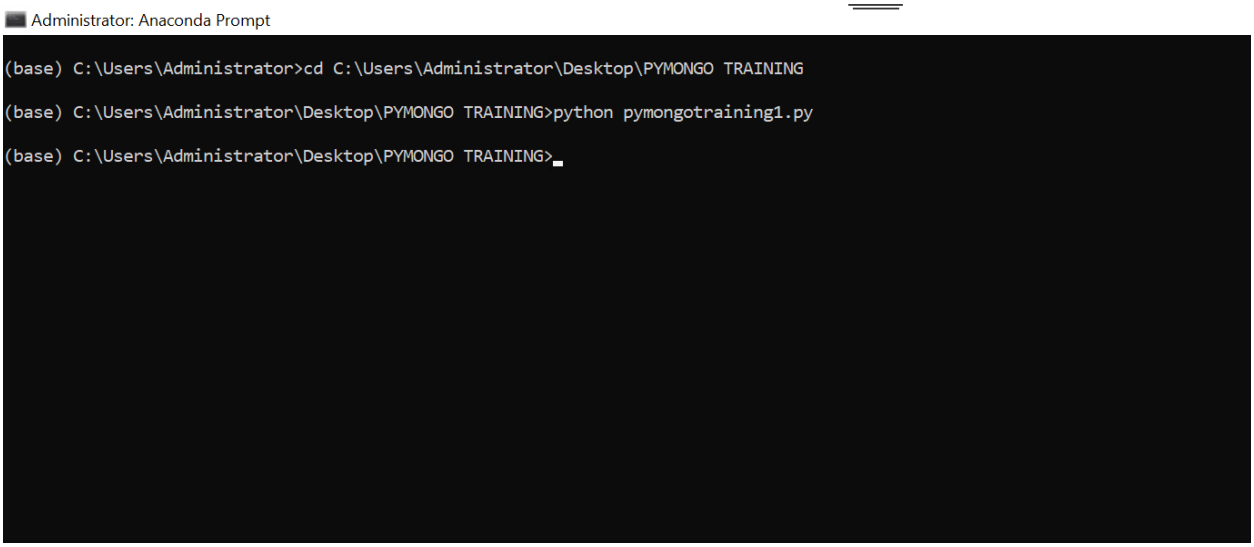
- Changed to Desktop directory form user.

Step 3:

Execute the file with the anaconda prompt.

- Syntax to execute a file in pymongo.

  Python Filename.py



Output:

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