REPORT

What is database?

A database is an organized collection of structured information or data, typically stored electronically in a computer system. Database holds one or more collections of documents. Each document is stored as a Json-like object (specifically bson) and is part of a collection. Unlike traditional relational databases, mongodb doesn't rely on a table-based structure. Instead, it uses flexible documents with dynamic schemas. This makes it easy to store and retrieve data efficiently, especially in hierarchical formats within the Json documents.

Structured Data:

The information is typically organized in a specific format, often using tables with rows and columns. This makes it easier to search, filter, and analyze the data.

Database Management System (DBMS):

This is the software that acts like the filing cabinet manager. It allows you to store, retrieve, update, and manage all the data within the database.

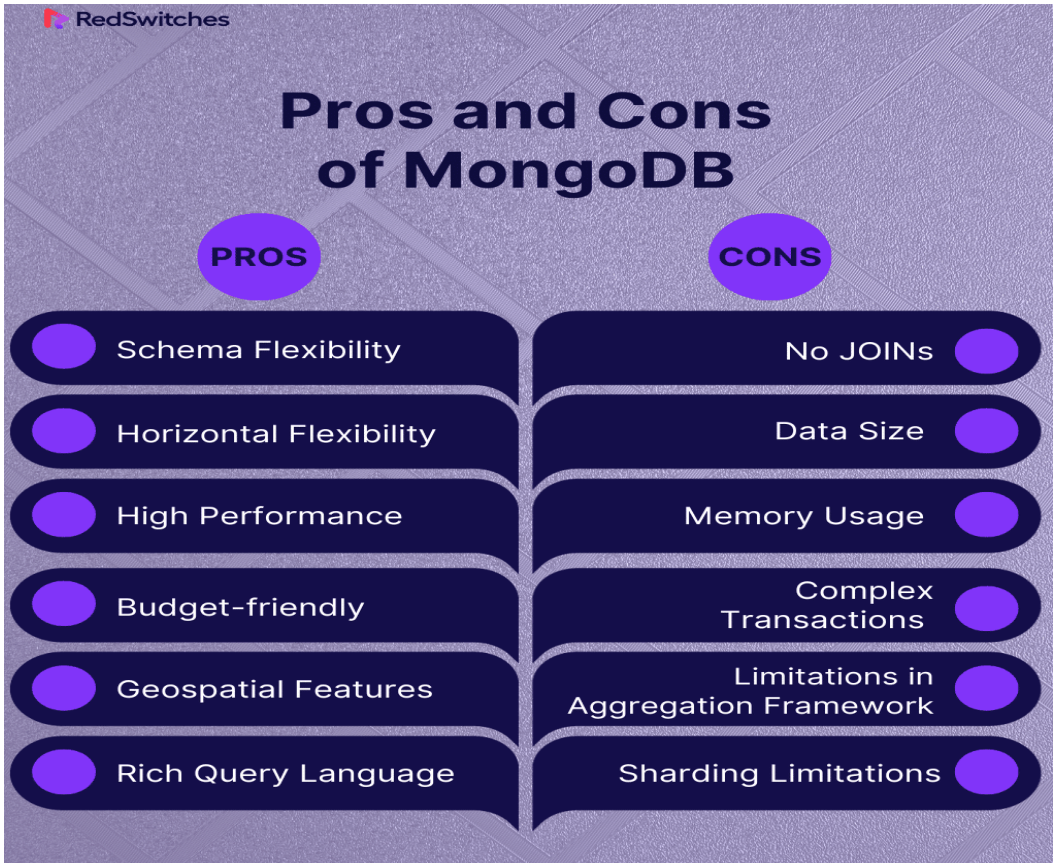
Data Types:

Databases can hold various kinds of information, including text, numbers, images, videos, and more.

Introduction of mongodb:

As per the above diagram, in MySQL the data are arranged in its own order and in mongodb they are distributed. The mongodb has no arrangement in it.

Advantages and limitations:

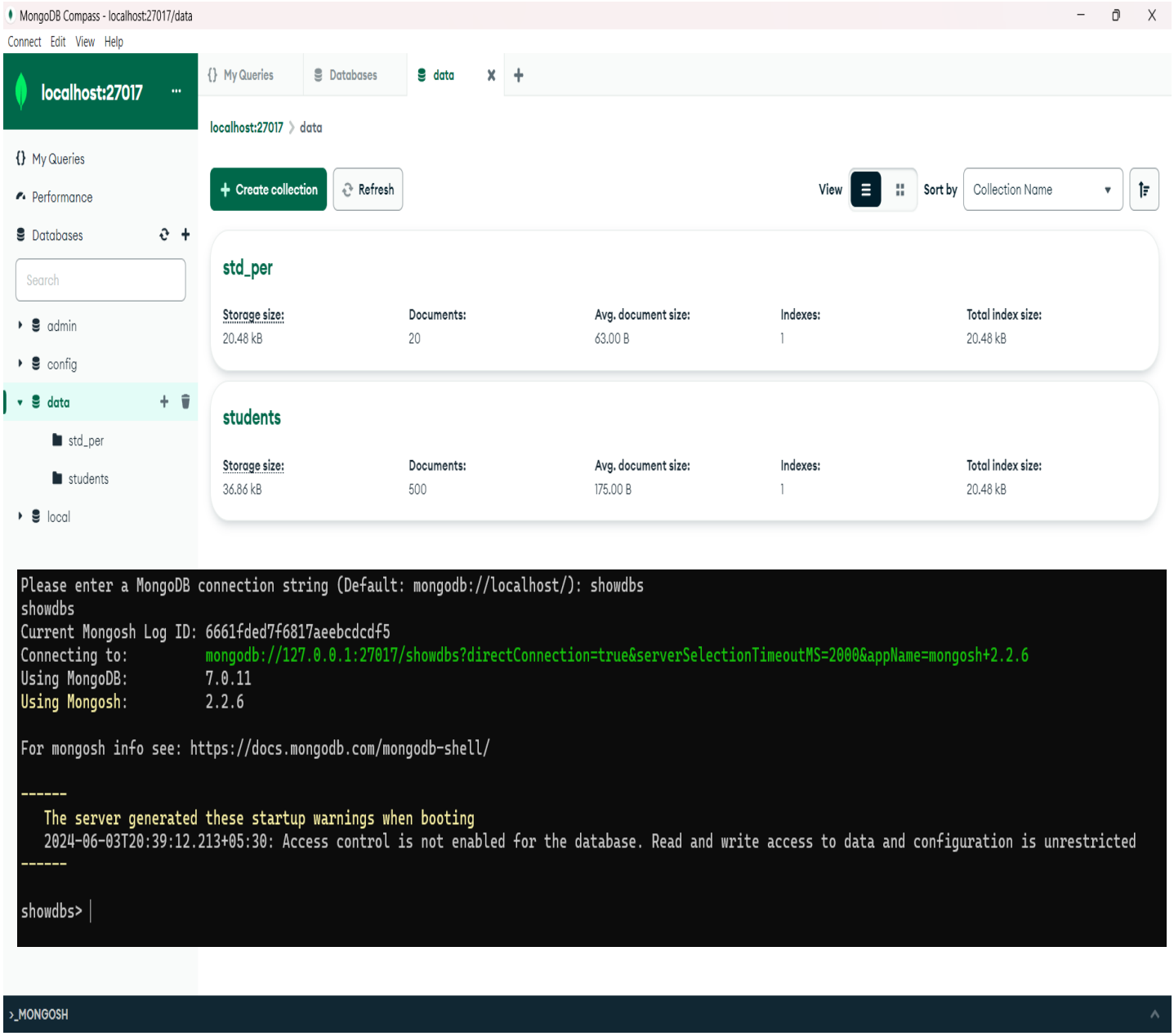


Few Commands to test after connections:

Command	Notes
Show dbs	All database are shown
Use db	Connect and use db
show collections	Show all tables
db.boo.insert({“car” :”bus”})	Insert a record to collection. create collection if not
db.boo.find()	Print all rows

Installation of mongodb:

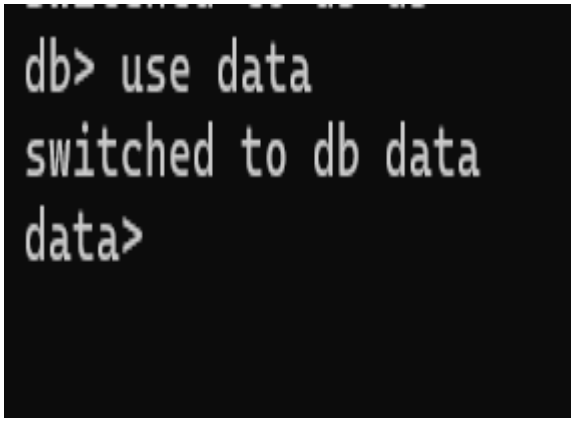
- Mongo shell download link
- Mongodb download link
- Mongodb compass download link
- Connect the shell with localhost:27017



showdbs:

Add, update and delete data:

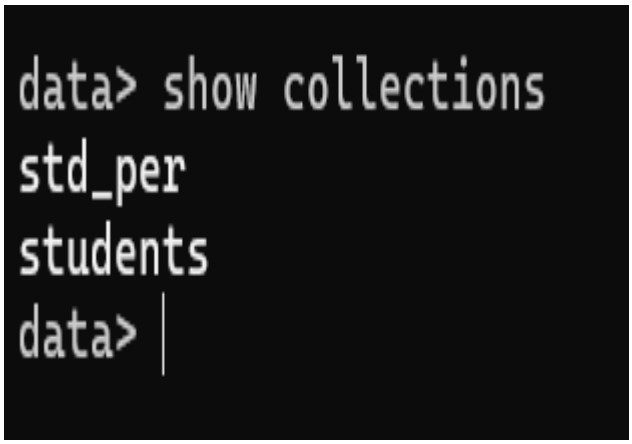
First step is we want to switch our database to the given collection by using command.



Now the database is switched to db named ‘data’.

To find whether the data present in the given collection, here the collection name is about the information of students.

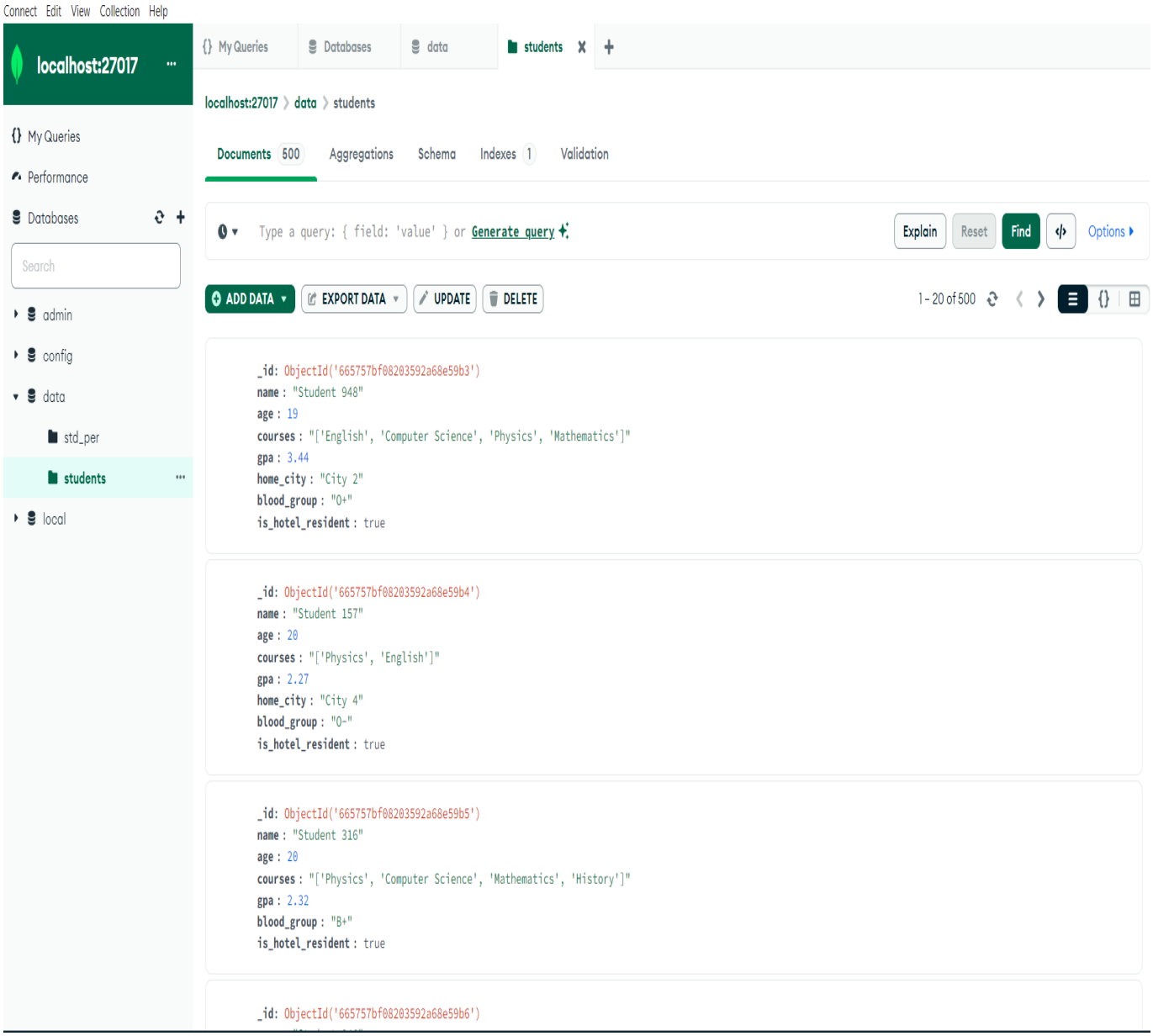
We can use the command “Show collections”.



In the above example the collection name is students.

To find the total number of collection of the database use the command.

“db.students.find().count()”



```
data> db.students.find().count()
500
data>
```

It shows the total collections of students in the database.

To find the collection of the database use the command.

“db.students.find()”

```
data> db.students.find()
[
  {
    _id: ObjectId('665757bf08203592a68e59b3'),
    name: 'Student 948',
    age: 19,
    courses: "['English', 'Computer Science', 'Physics', 'Mathematics']",
    gpa: 3.44,
    home_city: 'City 2',
    blood_group: 'O+',
    is_hotel_resident: true
  },
  {
    _id: ObjectId('665757bf08203592a68e59b4'),
    name: 'Student 157',
    age: 20,
    courses: "['Physics', 'English']",
    gpa: 2.27,
    home_city: 'City 4',
    blood_group: 'O-',
    is_hotel_resident: true
  },
  {
    _id: ObjectId('665757bf08203592a68e59b5'),
    name: 'Student 316',
    age: 20,
    courses: "['Physics', 'Computer Science', 'Mathematics', 'History']",
    gpa: 2.32,
    blood_group: 'B+',
    is_hotel_resident: true
  },
  {
    _id: ObjectId('665757bf08203592a68e59b6'),
    name: 'Student 346',
    age: 25,
    courses: "['Mathematics', 'History', 'English']",
    gpa: 3.31,
    home_city: 'City 8',
    blood_group: 'O-',
    is_hotel_resident: true
```

Collections:

A collection is a group of documents. If a document is the MongoDB analog of a row in a relational database, then a collection can be thought of as the analog to a table.

Database:

MongoDB groups collections into databases. A single instance of MongoDB can host several databases, each grouping together zero or more collections.

Data Types:

MongoDB server stores data using BSON format which

supports some additional Data types that are not available using the JSON format.

WHERE AND OR &CRUD:

WHERE:

Given a collection you want to filter a subset based on a condition. It is the place WHERE issued.

To find all students with GPA greater than 3, we use command-“db.students.find({gpa:{$gt:3}});”

```
data> db.students.find({gpa:{$gt:3}});
[
  {
    _id: ObjectId('665757bf08203592a68e59b3'),
    name: 'Student 948',
    age: 19,
    courses: "['English', 'Computer Science', 'Physics', 'Mathematics']",
    gpa: 3.44,
    home_city: 'City 2',
    blood_group: 'O+',
    is_hotel_resident: true
  },
  {
    _id: ObjectId('665757bf08203592a68e59b6'),
    name: 'Student 346',
    age: 25,
    courses: "['Mathematics', 'History', 'English']",
    gpa: 3.31,
    home_city: 'City 8',
    blood_group: 'O-',
    is_hotel_resident: true
  },
  {
    _id: ObjectId('665757bf08203592a68e59b7'),
    name: 'Student 930',
    age: 25,
    courses: "['English', 'Computer Science', 'Mathematics', 'History']",
    gpa: 3.63,
    home_city: 'City 3',
    blood_group: 'A-',
    is_hotel_resident: true
  },
  {
    _id: ObjectId('665757bf08203592a68e59b8'),
    name: 'Student 305',
    age: 24,
    courses: "['History', 'Physics', 'Computer Science', 'Mathematics']",
    gpa: 3.4,
    home_city: 'City 6',
    blood_group: 'O+',
    is_hotel_resident: true
  },
  {
```

In the above example we use the condition gpa greater than 3.

Here ‘$gt’ means greater than.

AND:

Given a collection you want to filter a subset based on multiple conditions. For
this type of situation we use AND.
To find all students who live in “City 4” AND have a based group of “B+”
Here we use the command:

```
db.students.find({
$and: [
{home_city : “City 4”},
{blood_group: “B+”}
]
});
```

```
data> db.students.find({$and:[{home_city:"City 4"},{blood_group:"B+"}]});
[
  {
    _id: ObjectId('665757bf08203592a68e5a4d'),
    name: 'Student 985',
    age: 21,
    courses: "['English', 'Computer Science', 'History', 'Physics']",
    gpa: 2.76,
    home_city: 'City 4',
    blood_group: 'B+',
    is_hotel_resident: true
  },
  {
    _id: ObjectId('665757bf08203592a68e5a56'),
    name: 'Student 267',
    age: 20,
    courses: "['Computer Science', 'Physics', 'Mathematics', 'English']",
    gpa: 2.5,
    home_city: 'City 4',
    blood_group: 'B+',
    is_hotel_resident: true
  },
  {
    _id: ObjectId('665757bf08203592a68e5a59'),
    name: 'Student 331',
    age: 25,
    courses: "['Computer Science', 'Physics']",
    gpa: 2.04,
    home_city: 'City 4',
    blood_group: 'B+',
    is_hotel_resident: false
  }
]
```

Above example is filtered based on some conditions like: ‘home_city: City4’ and ‘blood_group : B+’. It gives the output as the home_city of city4 and the blood_group that are having B+ only. This is how the AND operation works.

OR:

In the given collection that is student we want to filter a subset based on multiple conditions but any one is sufficient.

```
data> db.students.find({$or:[{home_city:"City 4"},{blood_group:"B+"}]});
[
  {
    _id: ObjectId('665757bf08203592a68e59b4'),
    name: 'Student 157',
    age: 20,
    courses: "['Physics', 'English']",
    gpa: 2.27,
    home_city: 'City 4',
    blood_group: 'O-',
    is_hotel_resident: true
  },
  {
    _id: ObjectId('665757bf08203592a68e59b5'),
    name: 'Student 316',
    age: 20,
    courses: "['Physics', 'Computer Science', 'Mathematics', 'History']",
    gpa: 2.32,
    blood_group: 'B+',
    is_hotel_resident: true
  },
  {
    _id: ObjectId('665757bf08203592a68e59bd'),
    name: 'Student 256',
    age: 19,
    courses: "['Computer Science', 'Mathematics', 'History', 'English']",
    gpa: 2.94,
    home_city: 'City 1',
    blood_group: 'B+',
    is_hotel_resident: true
  },
  {
    _id: ObjectId('665757bf08203592a68e59cb'),
    name: 'Student 504',
    age: 21,
    courses: "['Physics', 'Computer Science', 'English', 'Mathematics']",
    gpa: 2.42,
    home_city: 'City 2',
    blood_group: 'B+',
    is_hotel_resident: true
  },
  {
    _id: ObjectId('665757bf08203592a68e59cc'),
    name: 'Student 652',
```

In the above example, the stud database is filtered based on either ‘blood_group : A+’ or ‘gpa greater than 3.5’. ($gt: greater than).

Here $lt represent less than.

CURD:

C-Create/Insert

R-Remove

U-Update

D-Delete

This is applicable for a collection or a document.

- **Insert:**
  Here we can insert a new data into the exist collection.
  To insert the data into collection we use the following command:
  Const studentData={
  “name”: “Ash”,
  “age”:15,,
  “courses”:[“CS”,“MATHS”,”ENGLISH”],
  “gpa”:3.2,
  “home_city”: “City 3”,
  “blood_group”: “O+”,
  “is_hotel_resident”:true
  }

```
data> const studentData={ "name": "Ash","age": 15,"courses": ["CS", "MATHS", "ENGLISH"], "gpa": 3.2, "home_city": "City 3", "blood_group": "O+",
 "is_hotel_resident": true }

data> db.students.insertOne(studentData);
{
  acknowledged: true,
  insertedId: ObjectId('66868650431a883301cdcdf6')
}
```

In the above example we are inserting the student details name ‘Ash’ and other information to the collection of database called studentData. Here the insertion can be done for one time. But we can insert an information for many number of times. And the updated is stored in the given collection.

- **Update:**
  Here we can update any data that are present in the collections. To update we use ‘$set’ command.

```
data> db.students.updateOne({name:"Ash"},{$set:{gpa:3.67}})
{
  acknowledged: true,
  insertedId: null,
  matchedCount: 1,
  modifiedCount: 1,
  upsertedCount: 0
}
```

As shown in above figure it update the exist database.

And the new collection is added to the previous collections.

- **Delete:**

The delete operation is used to delete the data present in the given collection.

```
data> db.students.deleteOne({name:"Ash"}
... )
{ acknowledged: true, deletedCount: 1 }
data>
```

Here we use the commend deleteOne. Then it delete the the specified group of data ,here ‘Ash’ is deleted.

Projection, Limit & Selectors:

Here we can understand about the Projection and the limit and Selectors.

▪ Projection:

MongoDB projection is a tool that allows user to select only the fields they need form a document in a query result. It's built on top the find() method, so we can use any projection query without significantly modifying the existing functions.

Benefits of Projection:

➢ Reduced data transferred between the database and your application.

➢ Simplifies your code by focusing on the specific information you need.

➢ Improve query performance by retrieving only necessary data.

It allows us to select only the necessary data rather than selecting whole data from the document.

Some examples for projection are shown below:

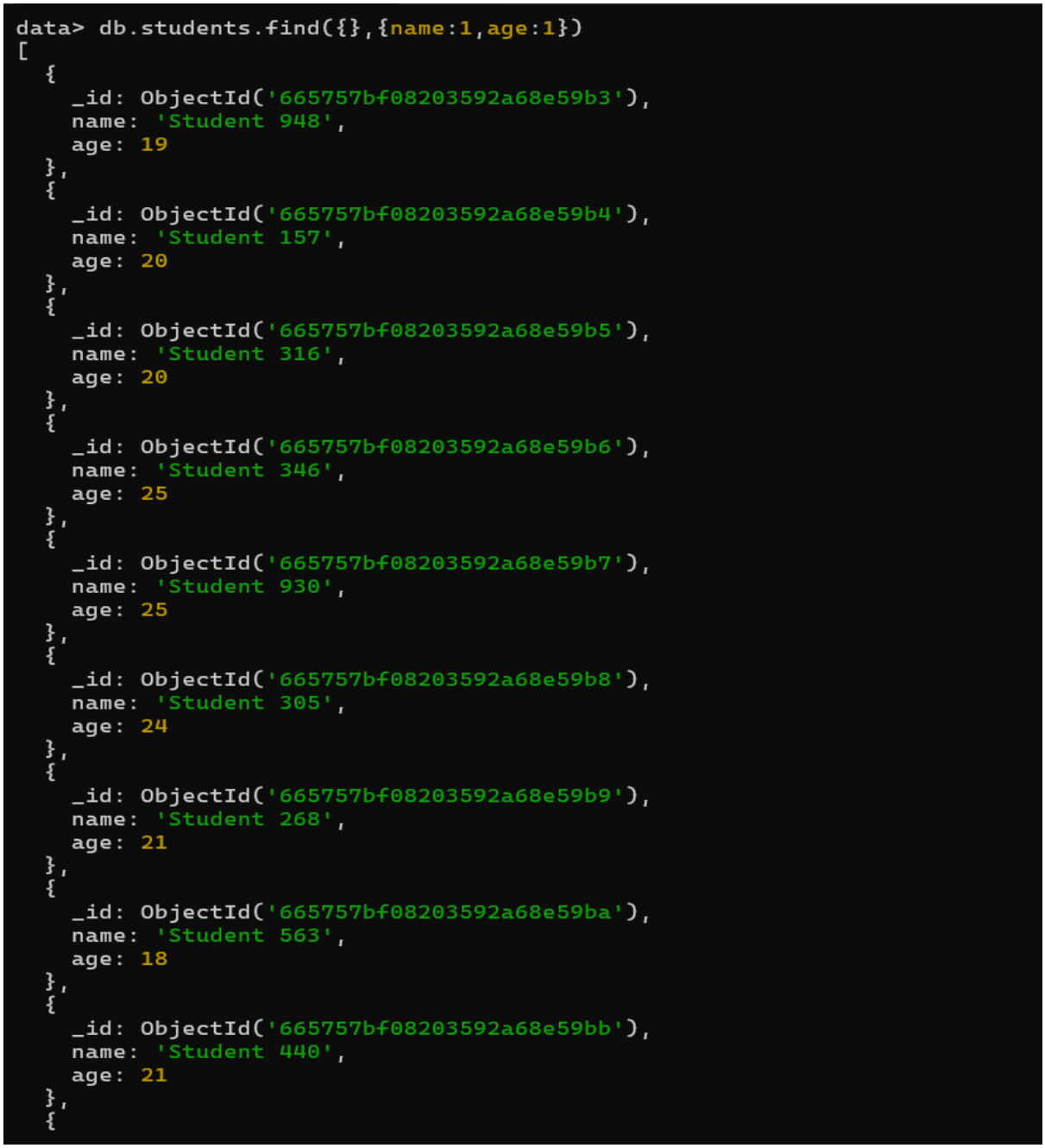
- Get Selected Attributes:

In the given collection if we want to FILTER a subset of attribute. That is the region where the projection is used.

In order to get only the name and age of the students we use the following commands.

"db.students.find({},{name:1,age:1});"

Then the output is shown below.



In the above example, Attribute shows only the exact name of the student and their age without any unnecessary informations. This how the projection works.

- Ignore Attributes:
  This attributes is used to print the exact data by excluding the object_id. Here _id is used to find the exact student rather than searching here and there in the database. It just saw the _id and find the specific group.

```
data> db.students.find({},{_id:0})
[
  {
    name: 'Student 948',
    age: 19,
    courses: "['English', 'Computer Science', 'Physics', 'Mathematics']",
    gpa: 3.44,
    home_city: 'City 2',
    blood_group: 'O+',
    is_hotel_resident: true
  },
  {
    name: 'Student 157',
    age: 20,
    courses: "['Physics', 'English']",
    gpa: 2.27,
    home_city: 'City 4',
    blood_group: 'O-',
    is_hotel_resident: true
  },
  {
    name: 'Student 316',
    age: 20,
    courses: "['Physics', 'Computer Science', 'Mathematics', 'History']",
    gpa: 2.32,
    blood_group: 'B+',
    is_hotel_resident: true
  },
  {
    name: 'Student 346',
    age: 25,
    courses: "['Mathematics', 'History', 'English']",
    gpa: 3.31,
    home_city: 'City 8',
    blood_group: 'O-',
    is_hotel_resident: true
  },
  {
    name: 'Student 930',
    age: 25,
    courses: "['English', 'Computer Science', 'Mathematics', 'History']",
    gpa: 3.63,
    home_city: 'City 3',
    blood_group: 'A-',
    is_hotel_resident: true
  },
  {
    name: 'Student 305',
```

As mentioned above it removes all the _id of the collection and just give the remaining instructions as there in the given collections.

LIMIT AND SELECTORS:

- **Limit:**

The limit operator is used with the find method. It's chained after the filter criteria or anysorting operations. It provides only the limited number of characters.

Some of the examples are shown below.

Here limit is used to get a minimum number of collection which are specified by the user.

```
data> db.students.find({},{_id:0}).limit(2);
[
  {
    name: 'Student 948',
    age: 19,
    courses: "['English', 'Computer Science', 'Physics', 'Mathematics']",
    gpa: 3.44,
    home_city: 'City 2',
    blood_group: 'O+',
    is_hotel_resident: true
  },
  {
    name: 'Student 157',
    age: 20,
    courses: "['Physics', 'English']",
    gpa: 2.27,
    home_city: 'City 4',
    blood_group: 'O-',
    is_hotel_resident: true
  }
]
```

The above example gives the complete information about limit, Here we take _id as zero because to prevent the _id numbers and we restricted or we use the limit to print only the minimum number of characters in the above example we mentioned only 2 so it prints the collections upto three collections.

- **Selectors:**

  - Comparison gt and lt
  - AND operator
  - OR operator

- Comparison gt and lt:

This operation is used to find the database that are greater than or lesser than.

NOTE:

gt: greater than

lt: lesser than.

Here is one example to find all the students whose age is less than 25.

```
data> db.students.find({age:{$lt:25}})
[
  {
    _id: ObjectId('665757bf08203592a68e59b3'),
    name: 'Student 948',
    age: 19,
    courses: "['English', 'Computer Science', 'Physics', 'Mathematics']",
    gpa: 3.44,
    home_city: 'City 2',
    blood_group: 'O+',
    is_hotel_resident: true
  },
  {
    _id: ObjectId('665757bf08203592a68e59b4'),
    name: 'Student 157',
    age: 20,
    courses: "['Physics', 'English']",
    gpa: 2.27,
    home_city: 'City 4',
    blood_group: 'O-',
    is_hotel_resident: true
  },
  {
    _id: ObjectId('665757bf08203592a68e59b5'),
    name: 'Student 316',
    age: 20,
    courses: "['Physics', 'Computer Science', 'Mathematics', 'History']",
    gpa: 2.32,
    blood_group: 'B+',
    is_hotel_resident: true
  },
  {
    _id: ObjectId('665757bf08203592a68e59b8'),
    name: 'Student 305',
    age: 24,
    courses: "['History', 'Physics', 'Computer Science', 'Mathematics']",
    gpa: 3.4,
    home_city: 'City 6',
    blood_group: 'O+',
    is_hotel_resident: true
  },
  {
    _id: ObjectId('665757bf08203592a68e59b9'),
    name: 'Student 268',
    age: 21,
    courses: "['Mathematics', 'History', 'Physics']",
    gpa: 3.98,
    blood_group: 'A+',
```

It gives an exact output of the students whose age is lesser then 25.

- AND operator:

AND operation is used to find the specific details about the collection. Here is some examples on AND operation.

To find students from “city 3” with blood group “O+”.

```
Type "it" for more
data> db.students.find({
... $and:[
... {home_city:"City 3"},
... {blood_group:"O+"}
... ]
... });
[
  {
    _id: ObjectId('665757bf08203592a68e59f2'),
    name: 'Student 722',
    age: 23,
    courses: "['Physics', 'History', 'Computer Science', 'Mathematics']",
    gpa: 3.75,
    home_city: 'City 3',
    blood_group: 'O+',
    is_hotel_resident: false
  },
  {
    _id: ObjectId('665757bf08203592a68e5b96'),
    name: 'Student 368',
    age: 19,
    courses: "['English', 'Computer Science', 'Mathematics']",
    gpa: 2.36,
    home_city: 'City 3',
    blood_group: 'O+',
    is_hotel_resident: true
  }
]
```

In the above given example shows the collections of the students who are from the city 3 and the blood Group are of type O+.

This can be easily done by using the AND operation.

- OR operation:

The OR operation can be explained by using the following example, here we take a example to find the Student who are hostel residents OR have a GPA less than 3.9.

It means it takes either the students how are hostel residents or the students whose gpa is less than 3.9.

```
data> db.students.find({ $or: [ { is_hotel_resident: true }, { gpa: { $lt: 3.9 } }] });
[
  {
    _id: ObjectId('665757bf08203592a68e59b3'),
    name: 'Student 948',
    age: 19,
    courses: "['English', 'Computer Science', 'Physics', 'Mathematics']",
    gpa: 3.44,
    home_city: 'City 2',
    blood_group: 'O+',
    is_hotel_resident: true
  },
  {
    _id: ObjectId('665757bf08203592a68e59b4'),
    name: 'Student 157',
    age: 20,
    courses: "['Physics', 'English']",
    gpa: 2.27,
    home_city: 'City 4',
    blood_group: 'O-',
    is_hotel_resident: true
  },
  {
    _id: ObjectId('665757bf08203592a68e59b5'),
    name: 'Student 316',
    age: 20,
    courses: "['Physics', 'Computer Science', 'Mathematics', 'History']",
    gpa: 2.32,
    blood_group: 'B+',
    is_hotel_resident: true
  },
  {
    _id: ObjectId('665757bf08203592a68e59b6'),
    name: 'Student 346',
    age: 25,
    courses: "['Mathematics', 'History', 'English']",
    gpa: 3.31,
    home_city: 'City 8',
    blood_group: 'O-',
    is_hotel_resident: true
  },
  {
    _id: ObjectId('665757bf08203592a68e59b7'),
    name: 'Student 930',
    age: 25,
    courses: "['English', 'Computer Science', 'Mathematics', 'History']",
    gpa: 3.63,
    home_city: 'City 3',
```

The above example gives the correct conclusion about the OR operation. Here we take the example that wants to give the output as the students who are hostel residents OR the students whose gpa is less than 3.9.

Bitwise operator:

- In our example its a 32 bit each bit representing different things
- Bitwise value 7 means all access 7 -> 111

Bit 3	Bit 2	Bit 1
Cafe	campus	Lobby

Bitwise type:

Bitwise

Name	Description
`$bitsAllClear`	Matches numeric or binary values in which a set of bit positions *all* have a value of `0`.
`$bitsAllSet`	Matches numeric or binary values in which a set of bit positions *all* have a value of `1`.
`$bitsAnyClear`	Matches numeric or binary values in which *any* bit from a set of bit positions has a value of `0`.
`$bitsAnySet`	Matches numeric or binary values in which *any* bit from a set of bit positions has a value of `1`.

Query:

In MongoDB, a query is a way to search for and retrieve documents from a collection that match specified criteria. Queries are typically performed using the find() method, which allows you to define filters and conditions to narrow down the results.

Here is a basic example of a MongoDB query:
javascript
db.collections.find({ field: value })

This query searches for documents in the specified collection where field equals value.
CONST
LOBBY_PERM
ISSION=1;
CONST
CAMPUS_PER
MISSION=2;

Two constants are defined: LOOBY_PERMISSION with a value 1 and CAMPUS_PERMISSION WITH A VALUE 2.

To find students with both lobby and campus permission we use
db.std_per.find({
permission
:{$bitsAllSets:[LOBBY_PERMISSION,CAMPUS_PERMISSION]}
});

```
data> const LOOBY_PERMISSION=1;

data> const CAMPUS_PERMISSION=2;

data> db.std_per.find({permissions:{$bitsAllSet:[LOOBY_PERMISSION,CAMPUS_PERMISSION]}})
[
  {
    _id: ObjectId('66575dea673efefc7bba033c'),
    name: 'George',
    age: 21,
    permissions: 6
  },
  {
    _id: ObjectId('66575dea673efefc7bba033d'),
    name: 'Henry',
    age: 27,
    permissions: 7
  },
  {
    _id: ObjectId('66575dea673efefc7bba033e'),
    name: 'Isla',
    age: 18,
    permissions: 6
  }
]
```

To find all the students in the collections we use

db.std_per.find({

permission

:{$bitsAllSets:[LOBBY_PERMISSION,CAMPUS_PERMISSION]}

}).count()

```
data> db.std_per.find({permissions:{$bitsAllSet:[LOOBY_PERMISSION,CAMPUS_PERMISSION]}}).count()
3
data> db.std_per.find({permissions:{$bitsAllSet:[LOOBY_PERMISSION]}}).count()
9
```

$bitsAllSets:

In MongoDB, $bitsAllSet is an operator that matches documents where all of the bit positions given by the query are set (i.e. 1) in a specified field. The field value must be either numeric or a BinData instance for the operator to work.

Geospatial Query:

A geospatial query involves retrieving information from a database based on geographic locations and spatial relationships. These queries are used in Geographic Information Systems (GIS) to analyze and visualize spatial data.

_id:2
name: “Restaurant B”
location: Object
type: “Point”
coordinates: Array (2)

```
db.locations.
find({
locations:{
$geoWithin:{
$centerSphere:[[-74.005,40.712],0.00621376]
}
}
});
```