

Mongo DB

INTRODUCTION:

Mongo DB is an open-source document-oriented database that is designed to store a large scale of data and also allows you to work with data efficiently. It is categorized under the NoSQL database because the storage and retrieval of data in the MongoDB are not in the form of tables.

The MongoDB database is developed and managed by MongoDB, Inc. under SSPL (server side public license) and initially released in February 2009. It also provides official driver support for all the popular languages like C, C++, C#, and .Net, Go, Java, Node.js, Python, etc.. Some of the companies that used MongoDB like Facebook, Nokia, eBay, Adobe, Google, etc. to store their large amount of data.

Why and where you should use mongo DB:

Since, MongoDB is a NoSQL database, so we need to understand when and why we need to use this type of database in the real-life applications. In normal circumstances, MongoDB is always preferred by the developers when our main concern is to deal with large volume of data with a high performance. If we want to insert thousands of records in a second, then the MongoDB is the best choice for that. MongoDB is good for some of the situations like:

- E-Commerce type of product-based applications.
- Blog and content management system.
- Need to maintain location wise Geospatial data.
- For maintaining data related to the social and networking types.

Comparison between MySQL and MongoDB:

MySQL	MongoDB
Database	Database
Table	Collection
Index	Index
Row	BSON Document
Column	BSON Field
Primary key	Primary key
Group By	Aggregation

WHAT IS DATABASE:

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 filling cabinet manager. It allows you to store, retrieve, update, and manage all the data within the database.

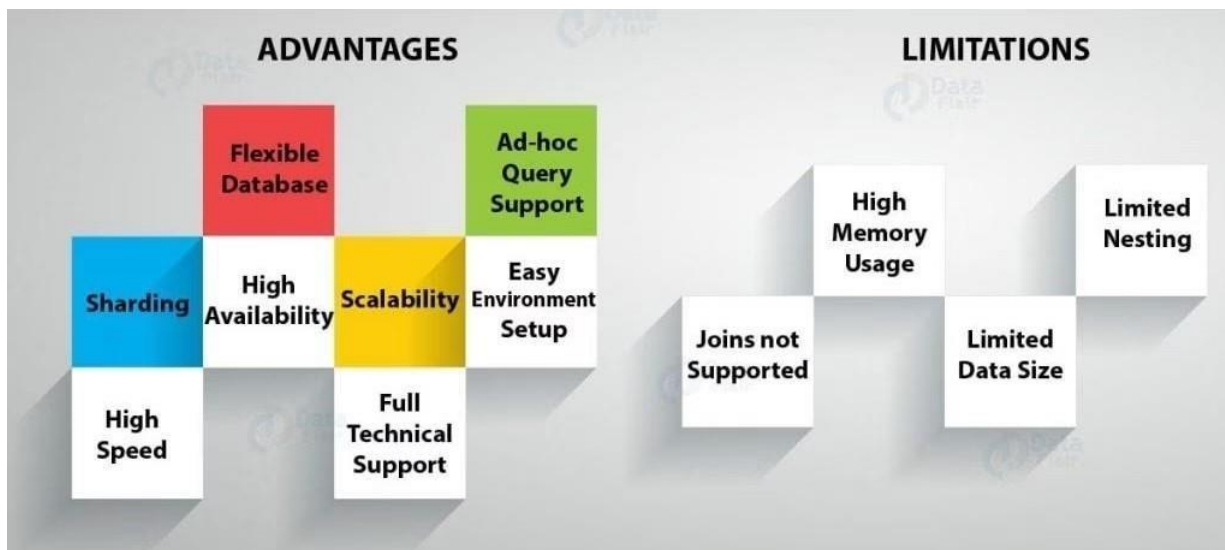
Data Types:

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



The above diagram is the best example for MongoDB and MySQL. In this if someone ask for a Paracetamol capsule then the shop owner search easily if they are arranged in correct order , if he don't now about the tablets he can easily find it seeing the arrangements and it is more faster it is possible incase of MySQL. But in MongoDB there is no arrangement they are messed up and it will consume more time and it make delay to search any data because they are in unorganized order.

Advantages and Limitations of MongoDB:



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
db.boo.remove()	Remove the table

Installation of MongoDB :

- Mongo Shell download [link](#)
- All the work is expected to do it in mongo shell not in mongo compass.
- Install [Studio3T](#).connect to mongoddb://localhost:27017

➤ Add, Update and Delete Data:

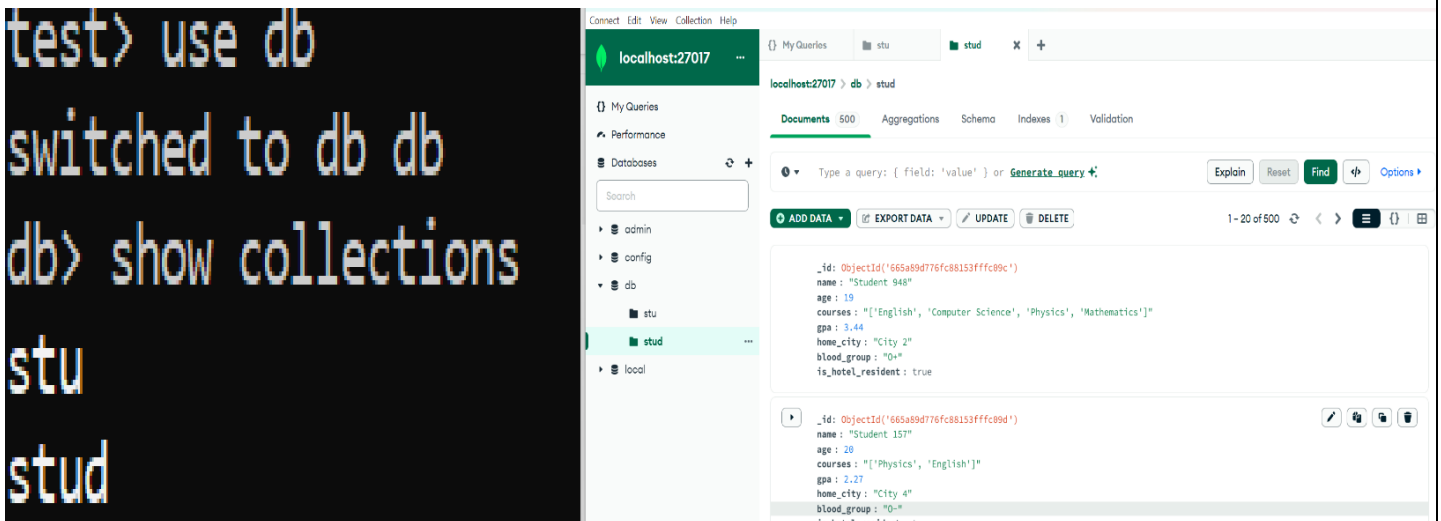
First step is we want to switch our database to the given collection by using command. “use db”.

```
test> use db
switched to db db
```

Now the database is switched to db.

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”



The image shows a terminal window and a MongoDB Compass interface. The terminal window displays the following commands and output:

```
test> use db
switched to db db
db> show collections
stu
stud
```

The MongoDB Compass interface shows the 'stud' collection selected. The left sidebar lists the databases: admin, config, db, stu, and local. The main panel shows the 'stud' collection with 500 documents. The first document is displayed:

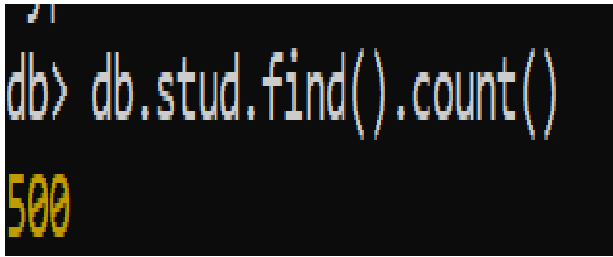
```
{
  "_id": ObjectId("665a89d776fc88153fffc09c"),
  "name": "Student 948",
  "age": 19,
  "courses": "[ 'English', 'Computer Science', 'Physics', 'Mathematics' ]",
  "gpa": 3.44,
  "home_city": "City 2",
  "blood_group": "O+",
  "is_hotel_resident": true
}
```

In the above example the collection name is stud or stu.

```
]
Type "it" for more
db> show dbs
admin      40.00 KiB
config     84.00 KiB
db         56.00 KiB
local      40.00 KiB
db> |
```

By using “show dbs” command all database are shown.


To find the total number of collection of the database use the command. “db.stud.find().count()”



```
db> db.stud.find().count()
500
```

It shows the total collections of students in the database.

To find the collection of the database use the command. “db.stud.find()”



```
db> db.stud.find()
[
  {
    _id: ObjectId('665a89d776fc88153fffc09c'),
    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('665a89d776fc88153fffc09d'),
    name: 'Student 157',
    age: 20,
    courses: "['Physics', 'English']",
    gpa: 2.27,
    home_city: 'City 4',
    blood_group: 'O-',
    is_hotel_resident: true
  },
  {
    _id: ObjectId('665a89d776fc88153fffc09e'),
    name: 'Student 316',
    age: 20,
    courses: "['Physics', 'Computer Science', 'Mathematics', 'History']",
    gpa: 2.32,
    blood_group: 'B+',
    is_hotel_resident: true
  },
  {
    _id: ObjectId('665a89d776fc88153fffc09f'),
    name: 'Student 346',
    age: 25,
    courses: "['Mathematics', 'History', 'English']",
    gpa: 3.31,
    home_city: 'City 8',
    blood_group: 'O-'
  }
]
```

The above example shows the list of data base present in the given collection. Here we assign a database as student so it displays all the details about the student in the specified manner.

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. That is the place WHERE is used.

To find all students with GPA greater than 3.5, we use

command-“db.stud.find({gpa:{\$gt:3.5}});”

```
db> db.stud.find({gpa:{$gt:3.5}});
[
  {
    _id: ObjectId('665a89d776fc88153fffc0a0'),
    name: 'Student 930',
    age: 25,
    courses: "['English', 'Computer Science', 'Mathematics', 'History']",
    gpa: 3.63,
    home_city: 'City 3',
    blood_group: 'A-',
    is_hotel_resident: true
  },
  {
    _id: ObjectId('665a89d776fc88153fffc0a2'),
    name: 'Student 268',
    age: 21,
    courses: "['Mathematics', 'History', 'Physics']",
    gpa: 3.98,
  },
]
db> db.stud.find({gpa:{$gt:3.5}});
[
  {
    _id: ObjectId('665a89d776fc88153fffc0a0'),
    name: 'Student 930',
    age: 25,
  },
  {
    _id: ObjectId('665a89d776fc88153fffc0a2'),
    name: 'Student 268',
    age: 21,
  },
]
```

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 5” AND have a blood group of “A+”

Here we use the command:

```
db.stud.find({  
  
$and: [  
  
{home_city : "City 5"},  
  
{blood_group: "A+"}  
  
]  
  
});
```

```
db> db.stud.find({  
... $and:[  
... {home_city:"City 5"},  
... {blood_group:"A+"}  
... ]  
... });  
[  
  {  
    _id: ObjectId('665a89d776fc88153fffc0d3'),  
    name: 'Student 142',  
    age: 24,  
    courses: "['History', 'English', 'Physics', 'Computer Science']",  
    gpa: 3.41,  
    home_city: 'City 5',  
    blood_group: 'A+',  
    is_hotel_resident: false  
  },  
  {  
    _id: ObjectId('665a89d776fc88153fffc1f3'),  
    name: 'Student 947',  
    age: 20,  
    courses: "['Physics', 'History', 'English', 'Computer Science']",  
  },  
  {  
    home_city: 'City 5',  
    blood_group: 'A+',  
    is_hotel_resident: true  
  },  
  {  
    _id: ObjectId('665a89d776fc88153fffc265'),  
    name: 'Student 567',  
    age: 22,  
    courses: "['Computer Science', 'History', 'English', 'Mathematics']",  
    gpa: 2.01,  
    home_city: 'City 5',  
    blood_group: 'A+',  
    is_hotel_resident: true  
  }  
]
```

Above example is filtered based on some conditions like:

‘home_city: City5’ and ‘blood_group : A+’.

It gives the output as the home_city of city5 and the blood_group that are having A+ 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.

```
db.stud.find({ $or: [ { blood_group: "A+" }, { gpa: { $gt: 3.5 } }] })

{
  _id: ObjectId('665a89d776fc88153fffc0a0'),
  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('665a89d776fc88153fffc0a2'),
  name: 'Student 268',
  age: 21,
  courses: "['Mathematics', 'History', 'Physics']",
  gpa: 3.98,
  blood_group: 'A+',
  is_hotel_resident: false
},
{
  _id: ObjectId('665a89d776fc88153fffc0a7'),
  name: 'Student 177',
  age: 23,
  courses: "['Mathematics', 'Computer Science', 'Physics']",
  gpa: 2.52,
  home_city: 'City 10',
  blood_group: 'A+',
  is_hotel_resident: true
},
{
  _id: ObjectId('665a89d776fc88153fffc0ac'),
  name: 'Student 368',
  age: 20,
  courses: "['English', 'History', 'Physics', 'Computer Science']",
  gpa: 3.91,
```

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

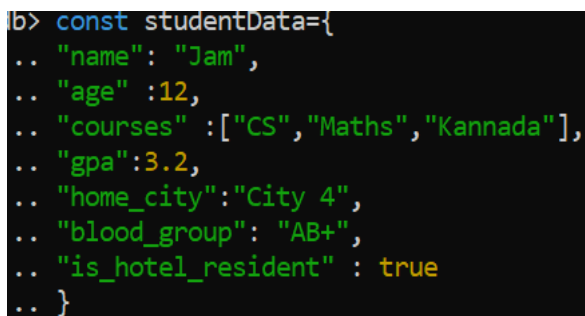
This is applicable for a collection or a document.

- **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": "Jam",  
  "age":12,,  
  
  "courses":["CS", "MATHS", "KANNADA"],  
  "gpa":3.2,  
  "home_city": "City 4",  
  "blood_group": "AB+",  
  "is_hotel_resident":true  
}
```

A terminal window with a black background and green text. It shows the MongoDB insert command: `ib> const studentData={` followed by several lines of document fields: `.. "name": "Jam",`, `.. "age" :12,`, `.. "courses" :["CS", "Maths", "Kannada"],`, `.. "gpa":3.2,`, `.. "home_city":"City 4",`, `.. "blood_group": "AB+",`, `.. "is_hotel_resident" : true`, and `.. }`.

```
ib> const studentData={  
.. "name": "Jam",  
.. "age" :12,  
.. "courses" :["CS", "Maths", "Kannada"],  
.. "gpa":3.2,  
.. "home_city":"City 4",  
.. "blood_group": "AB+",  
.. "is_hotel_resident" : true  
.. }
```

In the above example we are inserting the student details name 'Jam' 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.

```
db> db.students.updateOne( { name:"Sam" } , { $set:{  
gpa:3} } )  
{  
  acknowledged: true,  
  insertedId: null,  
  matchedCount: 1,  
  modifiedCount: 1,  
  upsertedCount: 0  
}  
db> |
```

As shown in above figure it updates the existing 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.

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

Here we use the command deleteOne. Then it deletes the specified group of data, here 'Sam' 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.stud.find({},{name:1,age:1});”
```

Then the output is shown below.

```
db> db.stud.find({}, {name:1, age:1})
[
  {
    _id: ObjectId('665a89d776fc88153fffc09c'),
    name: 'Student 948',
    age: 19
  },
  {
    _id: ObjectId('665a89d776fc88153fffc09d'),
    name: 'Student 157',
    age: 20
  },
  {
    _id: ObjectId('665a89d776fc88153fffc09e'),
    name: 'Student 316',
    age: 20
  },
  {
    _id: ObjectId('665a89d776fc88153fffc09f'),
    name: 'Student 346',
    age: 25
  },
  {
    _id: ObjectId('665a89d776fc88153fffc0a0'),
    name: 'Student 930',
    age: 25
  },
  {
    _id: ObjectId('665a89d776fc88153fffc0a1'),
    name: 'Student 305',
    age: 24
  },
  {
    _id: ObjectId('665a89d776fc88153fffc0a2'),
    name: 'Student 268',
    age: 21
  }
]
```

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.

```
db> db.stud.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
  },
]
```

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 any sorting 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.

```
db> db.stud.find({}, {_id:0}).limit(3);
{
  {
    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
  }
}
```

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 3 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 30.

```
db> db.stud.find({age:{$lt:30}})
[
  {
    _id: ObjectId('665a89d776fc88153fffc09c'),
    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('665a89d776fc88153fffc09d'),
    name: 'Student 157',
    age: 20,
    courses: "['Physics', 'English']",
    gpa: 2.27,
    home_city: 'City 4',
    blood_group: 'O-',
    is_hotel_resident: true
  },
  {
    _id: ObjectId('665a89d776fc88153fffc09e'),
    name: 'Student 316',
    age: 20,
    courses: "['Physics', 'Computer Science', 'Mathematics', 'History']",
    gpa: 2.32,
    blood_group: 'B+',
    is_hotel_resident: true
  },
  {
    _id: ObjectId('665a89d776fc88153fffc09f'),
    name: 'Student 346',
    age: 25,
    courses: "['Mathematics', 'History', 'English']",
    gpa: 3.31,
    home_city: 'City 8',
    blood_group: 'O-',
  }
]
```

It gives an exact output of the students whose age is lesser then 30.

- 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 2” with blood group “B+”.

```
db.stud.find({
  $and:[
    {home_city: "City 2"},
    {blood_group: "B+" }
  ]
});

_id: ObjectId('665a89d776fc88153fffc0b4'),
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('665a89d776fc88153fffc0eb'),
name: 'Student 367',
age: 19,
courses: "['English', 'Physics', 'History', 'Mathematics']",
gpa: 2.81,
home_city: 'City 2',
blood_group: 'B+',
is_hotel_resident: false
,
```

In the above given example shows the collections of the students who are from the city 2 and the bloodGroup are of type B+.

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 an example to find the Student who are hostel residents OR have a GPA less than 3.0. It means it takes either the students who are hostel residents or the students whose GPA is less than 3.0.

```
db> db.stud.find({
... $or:[
... {is_hostel_resident:true},
... {gpa:{$lt:3.0}}
... ]
... });
[
  {
    _id: ObjectId('665a89d776fc88153fffc09d'),
    name: 'Student 157',
    age: 20,
    courses: "['Physics', 'English']",
    gpa: 2.27,
    home_city: 'City 4',
    blood_group: 'O-',
    is_hotel_resident: true
  },
  {
    _id: ObjectId('665a89d776fc88153fffc09e'),
    name: 'Student 316',
    age: 20,
    courses: "['Physics', 'Computer Science', 'Mathematics', 'History']",
    gpa: 2.32,
    blood_group: 'B+',
    is_hotel_resident: true
  },
  {
    _id: ObjectId('665a89d776fc88153fffc0a3'),
    name: 'Student 563',
    age: 18,
    courses: "['Mathematics', 'English']",
    gpa: 2.25,
    blood_group: 'AB+',
    is_hotel_resident: false
  },
  {
    _id: ObjectId('665a89d776fc88153fffc0a4'),
    name: 'Student 440',
    age: 21,
```

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 3.0