

Add, Update, and Delete Data:

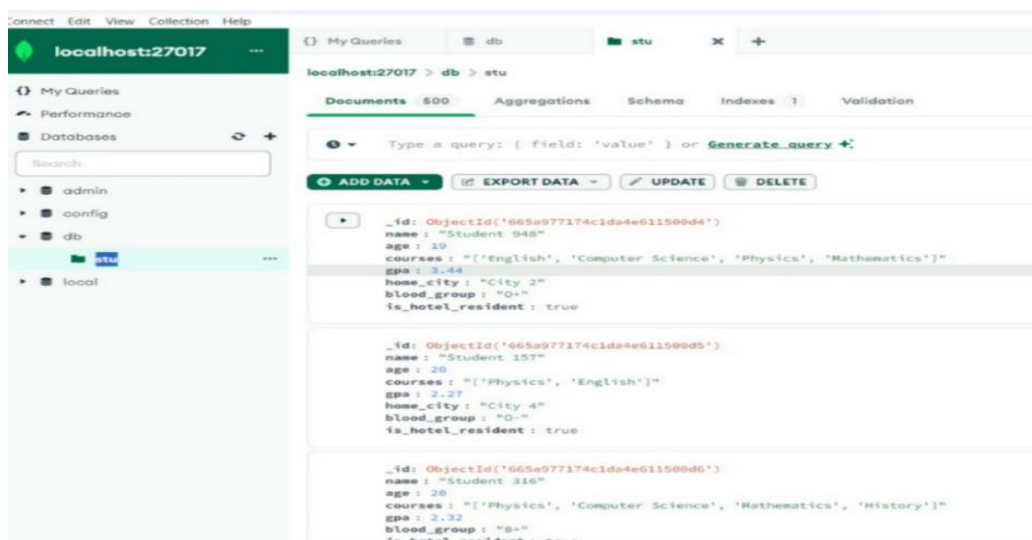
First step is we want to switch our database to the given collection by using command.

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

Now database is switched to db

To find the data is present in the given set we can use show collections

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



In the above example collections name is stu.

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

If we want to know the total number of collections we have to give

```
db.stu .find().count()
```

If we want to find the collections use this command

```
db.stu.find()
```

```
b> 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-'
}
```

Collections:

MongoDB database stores its data in collections. A collection holds one or more BSON documents. Documents are analogous to records or rows in a relational database table. Each document has one or more fields; fields are similar to the columns in a relational database table.

Database: A database is an organized collection of data stored in a computer system and usually controlled by a database management system (DBMS). The data in common databases is modeled in tables, making querying and

processing efficient. Structured query language (SQL) is commonly used for data querying and writing.

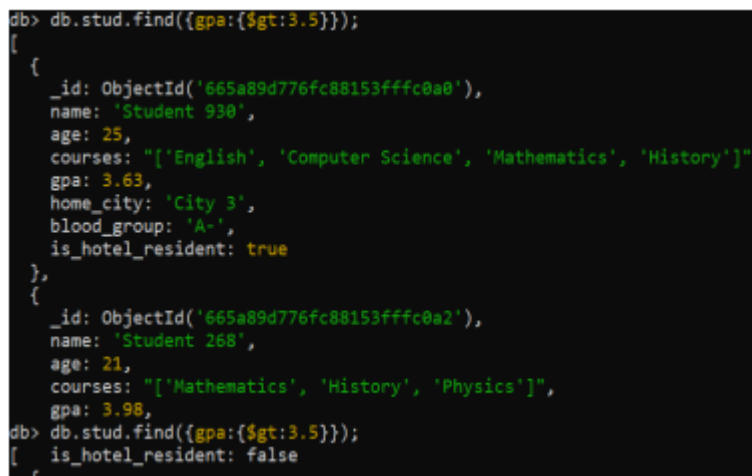
WHERE,AND,OR &CURD:

WHERE:

to find the collections which is greater than 3.5 we use this command

“db.stud.find({gpa:{\$gt:3.5}});”

\$gt----- greater than



```
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,
    is_hotel_resident: false
  }
]
```

The gpa having greater than 3.5 will appear in the output window.

AND:

Given a collection you want to filter a subset based on multiple conditions.

To find all students who live in “City 5” AND have a based group of “A+”

We use this command to find the given all students who lived in “city 5” and group “A+”.

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
  },
]

```

Here the output will give the students who are living in “city 5” and group “A+”.

OR:

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

To find all the students who are all having either of the blood_group:”A+” or \$gpa:”3.5”.

```

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
},
]

```

CURD:

C-Create/Insert

R-Remove

U-Update In the above example we insert thr

D-Delete

INSERT:

This command will show how to insert a data to the collections.

Const studentData={

“name”: “Jam”,

“age”:12,,

“courses”:[“CS”, “MATHS”, “KANNADA”],

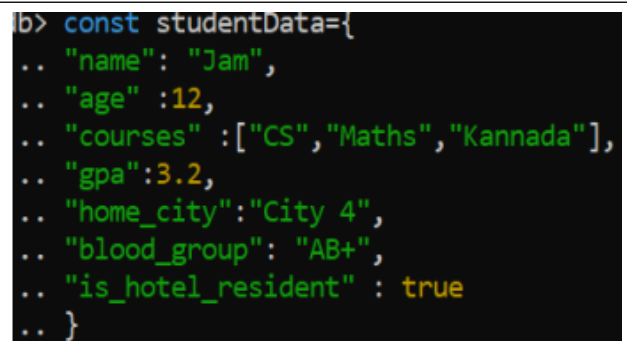
“gpa”:3.2,

“home_city”: “City 4”,

“blood_group”: “AB+”,

“is_hotel_resident”:true

}



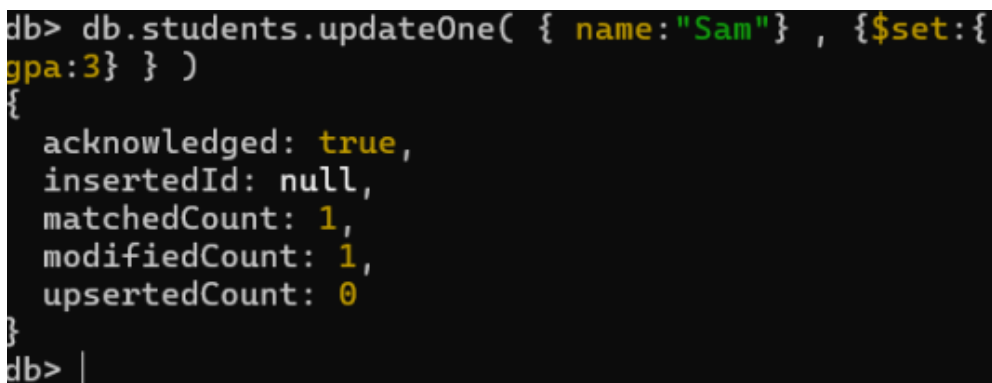
```
db> const studentData={  
  .. "name": "Jam",  
  .. "age" :12,  
  .. "courses" :["CS","Maths","Kannada"],  
  .. "gpa":3.2,  
  .. "home_city":"City 4",  
  .. "blood_group": "AB+",  
  .. "is_hotel_resident" : true  
  .. }  
}
```

In the above example we insert a name and other information of “Jam” to the database. The collection of database called studentData.

Update:

Update cpmmand is used to update the new data to the dataset.

\$set is used to update the command.



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

Here the data of “Sam” will be updated to the database.

Delete:

Delete command is used to delete the data from the present collections.

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

Here the data “Sam” will be deleted from the present database.

Projection:

This is used when we don't need all columns or attributes.

```
db> db.students.deleteOne({ name:"Sam" })
{ acknowledged: true, deletedCount: 1 }
db> db.students.find({}, {name:1 , gpa:1 })
[
  {
    _id: ObjectId('66587b4a0a3749dfd07d78a0'),
    name: 'Student 948',
    gpa: 3.44
  },
  {
    _id: ObjectId('66587b4a0a3749dfd07d78a1'),
    name: 'Student 157',
    gpa: 2.27
  },
  {
    _id: ObjectId('66587b4a0a3749dfd07d78a2'),
    name: 'Student 316',
    gpa: 2.32
  }
]
```

In the above example it shows only the name and gpa, because the command is given as 'name:1' and 'gpa:1'.

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.

LIMIT AND SELECTIONS:

• Limit:

The limit operator is used with the find method. It's chained after the filter criteria or any sorting operations.

Syntax:

```
db.collection.find({filter},  
{projection}).limit(number)
```

```
type it for more  
db> db.students.find({}, {_id:0}).limit(5)  
{  
  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  
},  
}
```

If we want to get a first 5 document we use limit(5).

• Selectors:

- Comparison gt and lt
- AND operator
- OR operator

Comparison gt lt:

To find the students who are greater than 20 we use this command.

```
b> db.stud.find({age:{$gt:20}});

{
  _id: ObjectId('665a89d776fc88153fffc09f'),
  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('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('665a89d776fc88153fffc0a1'),
  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
}
```

AND operator:

And operator is used to find the student 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
```