

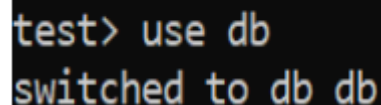
Add, update and Delete data

Collection:

A Collection is a grouping of documents. Each document within a collection can have different fields, and collections are analogous to tables in relational databases.

Here first we need 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.

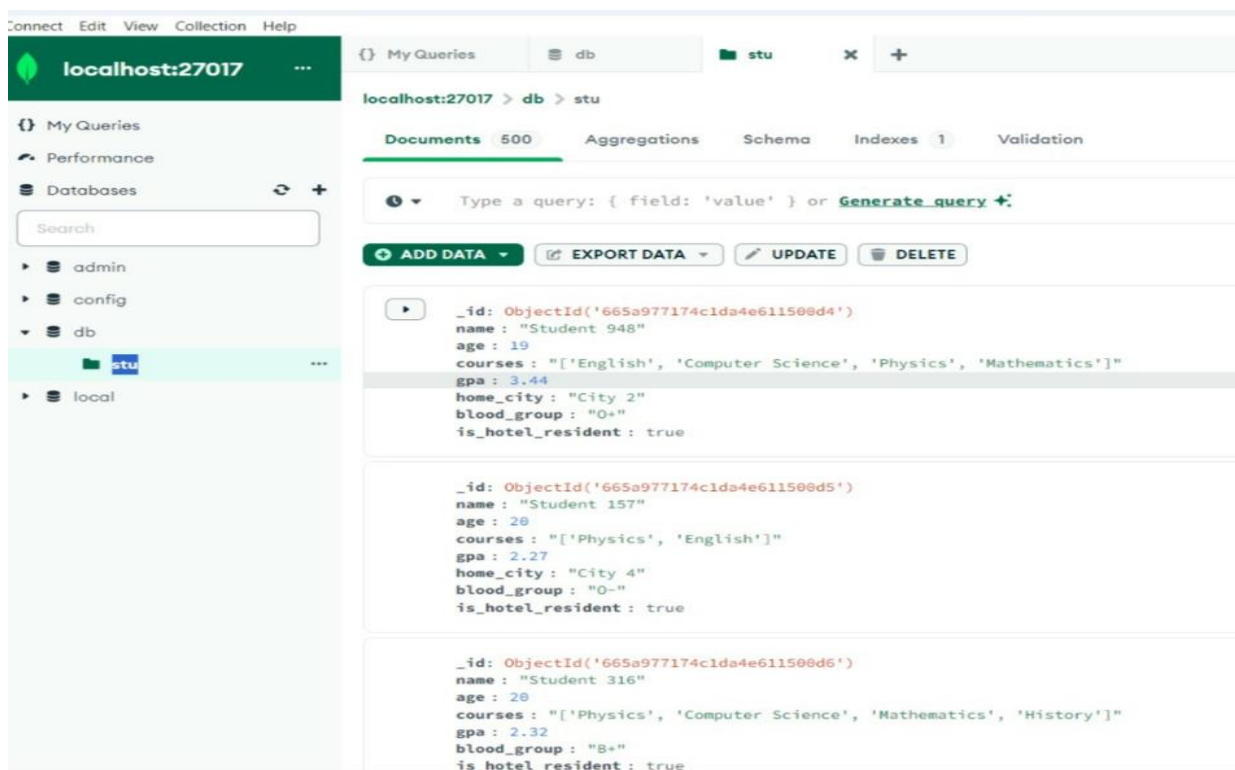
Next we need to find whether the given data is present in the collection or not.

Here the collection name is about the information of students. We can use the command.

“Show collections”

```
db> show collections
stu
stud
```

The above example shows the collection name is stud or stu.



Find:

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

“db.stud.find().count()”

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

The above example shows the total number of students in the collection in the database.

Another way,

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 collections of students in database.

There is a difference in both the examples .Firstly, we can get the number of collections, in the secondly we get entire information about that collection.

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 the 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}});
[
  is_hotel_resident: false
]
{
  _id: ObjectId('665a89d776fc88153fffc0a0'),
  name: 'Student 930',
  age: 25,
  student 368'
```

Here in the above command we use \$gt this represents the greater than and it gives the information that are belongs to greater than 3.5gpa . Similarly, for lesser than we use

And:

In the given 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+” 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']",

```

Above example is filtered based on some conditions like: 'home_city: City5' and 'blood_group : A+'.

OR:

The \$or operator is used to specify a compound query with multiple conditions, where at least one condition must be satisfied for a document to match.

Here the example:

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

```
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+',
```

CURD:

C-Create/Insert

R-Remove

U-Update

D-Delete

This is applicable for a collection.

- Insert/Create:

To insert the data into collection we use the below 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  
}”
```

Using the above command we are inserting the student details we can see this in below example:

```
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 inserted the student details name 'Jam' and other information to The collection of database called studentData.

- Update:

The update command in MongoDB allows you to modify documents within a collection. Here we can update any data within the collection.

To update we use '\$set' command.

Here we can see the example :

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

- Delete:

The delete command removes documents from a collection. Here to delete the documents from a collection we use “delete /remove”

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

- Update Many:

```
db.students.updateMany({ gpo: { $lt: 3.0 } }, { $inc: { gpo: 0.5 } });
```

- Delete Many:

```
db.students.deleteMany({ is_hotel_resident: false });
```

Projection:

This is used when we don't need all columns or attributes. It allows you to select specific fields from documents rather than retrieving the entire set of data.

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.

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 command like,

“db.stud.find({}, {name:1, age:1});”

```
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
  }
]
```

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: 'B+'
  }
]
```

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