Few Commands to test after connections

Command		Notes
show dbs		All Databases are shown
use db		Connect and use db
show collections		Show all tables
db.foo.insert({"bar" : "baz"})		Insert a record to collection. Create Collection if not exists
db.foo.find()		Print all rows
db.foo.remove()		Remove foo table

Add, Update & Delete Data:

"show dbs"

```
test> show dbs
admin     40.00 KiB
config   108.00 KiB
db        56.00 KiB
local     72.00 KiB
test> |
```

It shows student data base present in the collection.

"use db"

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

Data base is switched to db.

“show collections”

```
db> show collections
students
db>
```

The collection name is students.

To create students collections:

db.create collection(“students”)

In the above ex the collection name is students.

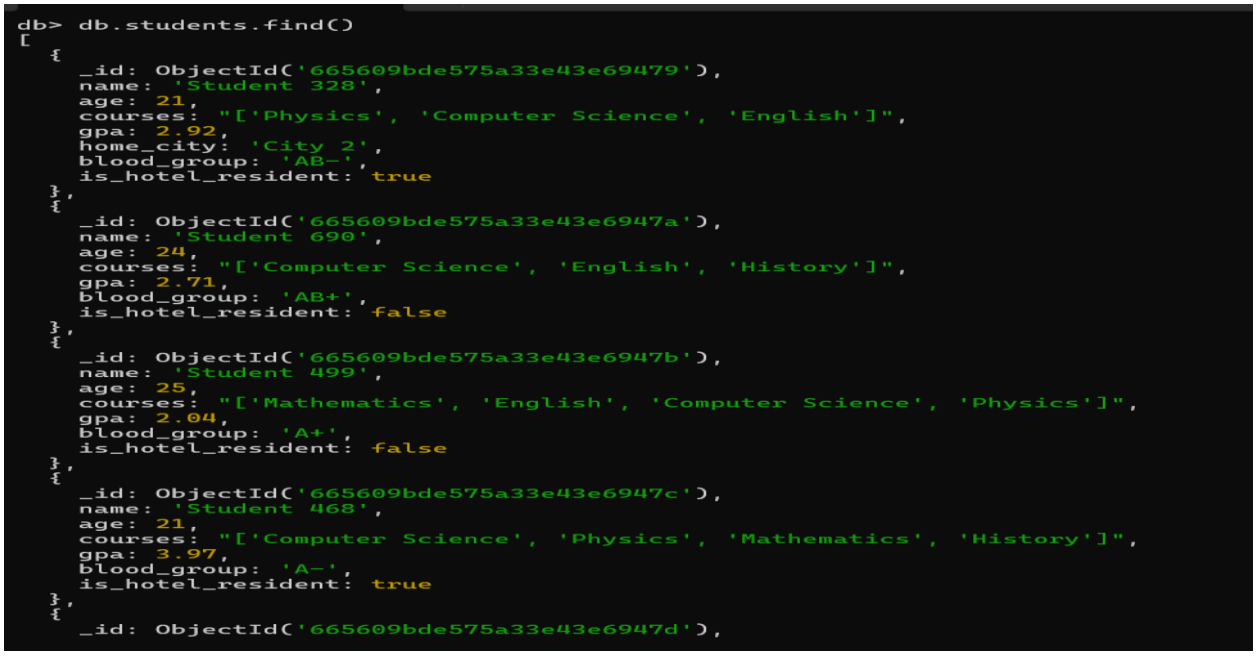
“db.students.find().count()”

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

It shows the total collections of students in the database.

To find the collection of the database use command.

“db.students.find()”



Documents:

The representation of a document varies by programming language, but most languages have a data structure that is a natural fit, such as a map, hash, or dictionary.

{"greeting" : "Hello, world!"}

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.

A database has its own permissions, and each database is stored in separate files on disk.

Datatype:

"zip": "90210" In MongoDB, documents are stored in BSON (Binary JSON), which supports a rich set of data types. Understanding these data types is crucial for designing efficient and effective data models. Here is a summary of the key BSON data types available in MongoDB:

"address": {

"street": "123 Main St",

"city": "Anytown",

"state": "CA",

}

Datatype:

More details link

- Date
- Int32
- Decimal
- Timestamp

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.students.find({gpa:{$gt: 3.5}});”

```
db> db.students.find({gpa:{$gt:3.5}});
[
  {
    _id: ObjectId('665609bde575a33e43e6947c'),
    name: 'Student 468',
    age: 21,
    courses: "['Computer Science', 'Physics', 'Mathematics', 'History']",
    gpa: 3.97,
    blood_group: 'A-',
    is_hotel_resident: true
  },
  {
    _id: ObjectId('665609bde575a33e43e6947e'),
    name: 'Student 652',
    age: 18,
    courses: "['History', 'Computer Science']",
    gpa: 3.93,
    home_city: 'City 8',
    blood_group: 'B+',
    is_hotel_resident: false
  },
  {
    _id: ObjectId('665609bde575a33e43e69481'),
    name: 'Student 239',
    age: 21,
    courses: "['English', 'Mathematics', 'Computer Science', 'Physics']",
    gpa: 3.75,
    home_city: 'City 7',
    blood_group: 'AB-',
    is_hotel_resident: false
  },
  {
    _id: ObjectId('665609bde575a33e43e69485'),
    name: 'Student 384',
    age: 18,
    courses: "['Mathematics', 'Computer Science']",
    gpa: 3.9,
    home_city: 'City 1',
    blood_group: 'O-',
    is_hotel_resident: false
  },
```

Here $gt represent the greater than and it gives the information that are belongs to greater than 3.5 gpa.

- **AND**:

Given a Collection you want to FILTER a subset based on multiple conditions.

To find students belongs to “city 5” AND “blood group is” A+”.Using command,

db.students.find({

$and:[

{home_city:”City 5”},

{blood_group:”A+”}

]

});

```
db> db.students.find({ $and:[
... {home_city:"City 5"},
... {blood_group:"A+"}
... ]
... });
[
  {
    _id: ObjectId('665609bde575a33e43e6949c'),
    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('665609bde575a33e43e695bc'),
    name: 'Student 947',
    age: 20,
    courses: "['Physics', 'History', 'English', 'Computer Science']",
    gpa: 2.86,
    home_city: 'City 5',
    blood_group: 'A+',
    is_hotel_resident: true
  },
  {
    _id: ObjectId('665609bde575a33e43e6962e'),
    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
  },
  {
    _id: ObjectId('665df4511655576509ad81e7'),
    name: 'Student 142',
    age: 24,
    courses: "['History', 'English', 'Physics', 'Computer Science']",
```

- **OR:**

  Given a collection you want to filter a subset based on multiple condition that is place OR is used.

  To find students belongs to “city 5” AND “blood group is” A+”.Using command.

  db.students.find({

  $or:[

{blood_group:”A+”},
{gpa:{$gt: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
,
_id: ObjectId('665a89d776fc88153fffc0ac'),
name: 'Student 368',
age: 20,
courses: "['English', 'History', 'Physics', 'Computer Science']",
gpa: 3.91,
```

- **CRUD:**

  C-Create/Insert

  R-Remove

  U-Update

  D-Delete

This is applicable for a collection or a document.

- **Insert:**

To insert the data into collection we use following command:

Const studentData={

“name”:”Alice Smith”,

“age”:22,,

“courses”:[“Mathematics”,”Computer Science”,”English”],

“gpa”:3.8,

“home_city”:”New York”,

“blood_group”:”A+”,

“is_hostel_resident”:false

};

```
test> const studentData = {
...       "name":"Alice Smith",
...       "age":22,
...       "courses":["Mathematics","Computer Science","English"],
...       "gpa":3.8,
...       "home_city":"New York",
...       "blood_group":"A+",
...       "is_hotel_resident":false
...       };

test> db.stu.insertOne(studentData);
{
  acknowledged: true,
  insertedId: ObjectId('665b529e49389824aecdcdf7')
}
test>
```

- **Update:**

To update the given students data using query.

db.students.updateOne({name:”Alice Smith”},{$set:{gpa:3.8}});

```
db> db.students.updateOne({name:"Alice Smith"},{$set:{gpa:3.8}});
{
  acknowledged: true,
  insertedId: null,
  matchedCount: 1,
  modifiedCount: 0,
  upsertedCount: 0
}
```

- **Delete:**

  In this delete operation is used to delete one student data from the given collection.

  db.students.deleteOne({name:"Sam"})

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

- **Projections:**

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

```
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
```

- **Benefits of Projection:**
  - ✓ Reduced data transferred between the database and your appliction.
  - ✓ Simplifies your code by focusing on the specific information you need.