

## PART 2:

### ❖ Add - Update – Delete:

Let's load the document:

- Download the student csv from this [link](#)
- Import the data to the collection created [link](#)
- You should be able to see the uploaded data in mongo compass.

Installation of mongo shell:

- Mongo Shell download [link](#)
- All the work is expected to do it in mongo shell not in mongo compass

Few commands to check after download of mongo shell:

Command	Expected Output	Notes
show dbs	<pre>admin 40.00 KiB config 72.00 KiB db 128.00 KiB local 40.00 KiB</pre>	All Databases are shown
use db	<pre>switched to db db</pre>	Connect and use db
show collections	<pre>Students</pre>	Show all tables
db.foo.insert({"bar": "baz"})		Insert a record to collection. Create Collection if not exists

Command	Notes
<code>db.foo.batchInsert([{"_id": 0}, {"_id": 1}, {"_id": 2}])</code>	Insert more than one document
<code>db.foo.find()</code>	Print all rows
<code>db.foo.remove()</code>	Remove foo table

#### ❖ Key points

- **Document:** A record in MongoDB is a document and are stored in BSON (Binary JSON) format, which allows for a rich set of data types and structures.
- **Collections:** A collection is a group of documents. MongoDB automatically creates the collections when the first document is inserted if it does not already exist.
- **Database:** A database is a container for collections.
- **Datatype:** each document will be in JSON format which will be as follows. Where each attributes inside can be of multiple data types.

```
{
  "name" : "John Doe",
  "address" : {
    "street" : "123 Park Street",
    "city" : "Anytown",
    "state" : "NY"
  }
}
```

**[A database contains a set of collections each collection contains a set of documents]**

### ❖ AND Operator:

```
db> db.stu.find({
... $and:[
... {home_city:"City 3"},
... {blood_group:"B+"}
... ]
... })
```

Let' see the output:

```
{
  _id: ObjectId('665759022abe60278b88a01b'),
  name: 'Student 165',
  age: 20,
  courses: "['English', 'History', 'Mathematics', 'Computer Science']",
  gpa: 2.92,
  home_city: 'City 3',
  blood_group: 'B+',
  is_hotel_resident: true
},
{
  _id: ObjectId('665759022abe60278b88a167'),
  name: 'Student 237',
  age: 19,
  courses: "['Physics', 'Mathematics', 'English', 'Computer Science']",
  gpa: 2.65,
  home_city: 'City 3',
  blood_group: 'B+',
  is_hotel_resident: false
},
```

Here, the \$and operator checks for students belonging to “City 3” and having a blood group “B+” in that specified home city.

```
db> db.stu.find({ $and: [ { home_city: "City 3" }, {blood_group:"B+"} ] }).count()
5
db> |
```

Here , only 5 students from City 3 having a blood group “B+”

❖ OR:

```
// Find all students who are hotel residents OR have a GPA less than 3.
db.students.find({
  $or: [
    { is_hotel_resident: true },
    { gpa: { $lt: 3.0 } }
  ]
});
```

Let's see the output:

```
{
  _id: ObjectId('665759022abe60278b889ffc'),
  name: 'Student 328',
  age: 21,
  courses: "['Physics', 'Computer Science', 'English']",
  gpa: 2.92,
  home_city: 'City 2',
  blood_group: 'AB-',
  is_hotel_resident: true
},
{
  _id: ObjectId('665759022abe60278b889ffc'),
  name: 'Student 690',
  age: 24,
  courses: "['Computer Science', 'English', 'History']",
  gpa: 2.71,
  blood_group: 'AB+',
  is_hotel_resident: false
},
```

Here the \$or operator checks for students either belonging to “hotel resident” or checks for student having a “gpa less than 3.0”

```
db> db.stu.find({ $or: [ { is_hotel_resident: true }, { gpa: { $lt: 3.0 } } ] }).count()
374
db> |
```

Here the output of 374 students having either gpa less than 3.0 or present in hotel resident.

## ❖ Add - Update -Delete:

### CRUD:

- C-create/insert
- R-remove
- U-update
- D-delete

This is applicable for a Collection (Table) or a Document (Row)

### • INSERT:

```
db> const stuData={
...   "name":"Chandana",
...   "age":20,
...   "courses":["Mathematics","Computer Science","English"],
...   "gpa":3.8,
...   "home_city":"New York",
...   "blood_group":"B+",
...   "is_hostel_resident":true
... };

db> db.stu.insertOne(stuData);
{
  acknowledged: true,
  insertedId: ObjectId('6661d7bd48f9e36487cdcdf6')
}
db> db.stu.find().count()
501
```

Here, we are inserting a new student data into a collection.

And after mentioning the details of student we are using the command called insertOne to insert one student details.

After it has successfully uploaded, we can check the count for confirmation.

To insert the data into the collection we use the following command:

```
Const stuData= {  
  "name": "Chandana",  
  "age": "20",  
  "course": ["Mathematics" "Computer Science" "English"],  
  "gpa": 3.8,  
  'home_city': "New York",  
  "blood_group": "B+",  
  "is_hotel_resident": true  
}
```

And after mentioning the details of the student, we use the

'insertOne' command to insert a single student's details.

After it has successfully uploaded, we can check the count for confirmation.

- **Update:**

Here, we can update any data that are present in the collections.

To update we use '\$set' command.

```
db> db.stu.updateOne({name: "Chandana"}, {$set: {gpa: 2.8}});  
{  
  acknowledged: true,  
  insertedId: null,  
  matchedCount: 1,  
  modifiedCount: 1,  
  upsertedCount: 0  
}
```

- **Delete:**

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

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

- **Projection:**

- This is used when we don't need all columns or attributes.
- The projection document is used as the second argument to the **find** method.
- Include field names with a value of **1** to specify fields to be returned.

### Get Selected Attributes

```
db> db.stu.find({}, {name:1,gpa:1})
[
  {
    _id: ObjectId('665759022abe60278b889fe7'),
    name: 'Student 948',
    gpa: 3.44
  },
  {
    _id: ObjectId('665759022abe60278b889fe8'),
    name: 'Student 157',
    gpa: 2.27
  },
  {
    _id: ObjectId('665759022abe60278b889fe9'),
    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'.

## Ignore Attributes:

Here, we have ignored the Id and home city of a student.

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

- **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 Selectors:**

- **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);
```



```

db> db.stu.find({}, {_id:0,home_city:0}).limit(1)
[
  {
    name: 'Student 948',
    age: 19,
    courses: "['English', 'Computer Science', 'Physics', 'Mathematics']",
    gpa: 3.44,
    blood_group: 'O+',
    is_hotel_resident: true
  }
]
db> |

```

Here, to get only first document we have used the limit(1);

### ➤ Selectors:

- **Comparison greater than(gt) and less than(lt):**

To find all the students with age greater then 20

```

db> db.stu.find({age:{$gt:20}});
[
  {
    _id: ObjectId('665759022abe60278b889fea'),
    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('665759022abe60278b889feb'),
    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
  }
]

```

```

db> db.stu.find({age:{$lt:20}}).count()
124
db> db.stu.find({age:{$gt:20}}).count()
310
db>

```

Here, we have 310 students who are older than 20 and 124 students who are elder than 20.