CURD:

Each letter in CURD represents meaning for it-

C-Create/Insert

R-Remove

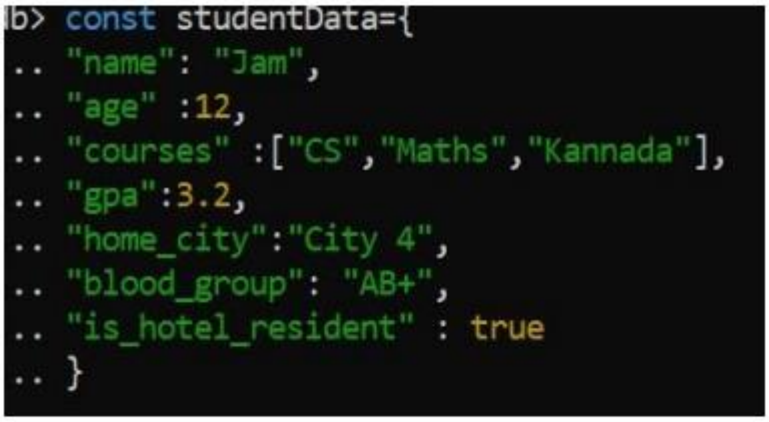
U-Update

D-Delete

Insert:

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

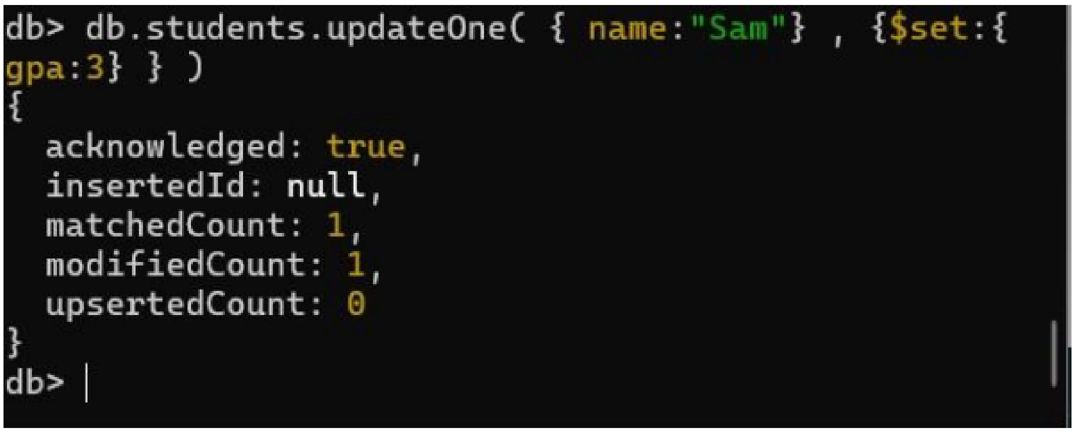
Const studentData={ “name”: “Jam”, “age”:12,, “courses”:[“CS”, “MATHS”, “KANNADA”], “gpa”:3.2, “home_city”: “City 4”, “blood_group”: “AB+”, “is_hotel_resident”:true }



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

Update:

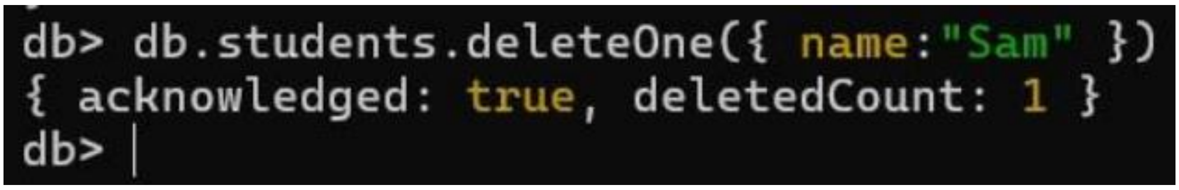
Here we can update any data that are present in the student collections. To update we use **‘$set’** command.



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

Delete:

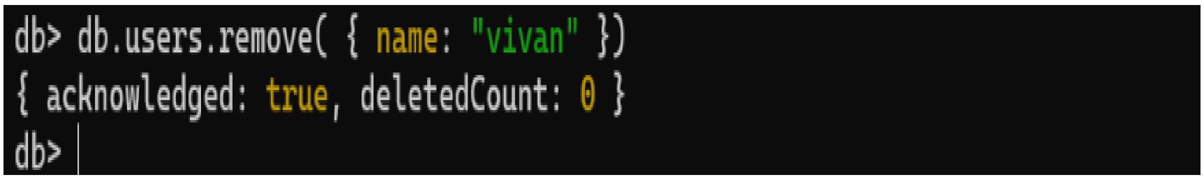
The deleteOne() method in MongoDB deletes the first document from the collection that matches the given selection criteria. It will delete/remove a single document from the collection. If you use this method in the capped collection, then this method will give a WriteError exception, so to delete documents from the capped collection use the drop() method.



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

Remove:

The remove() method removes documents from the database. It can remove one or all documents from the collection that matches the given query expression. If you pass an empty document({}) in this method, then it will remove all documents from the specified collection. It takes four parameters and returns an object that contains the status of the operation



```
db> db.users.remove( { name: "vivan" })
{ acknowledged: true, deletedCount: 0 }
db> |
```

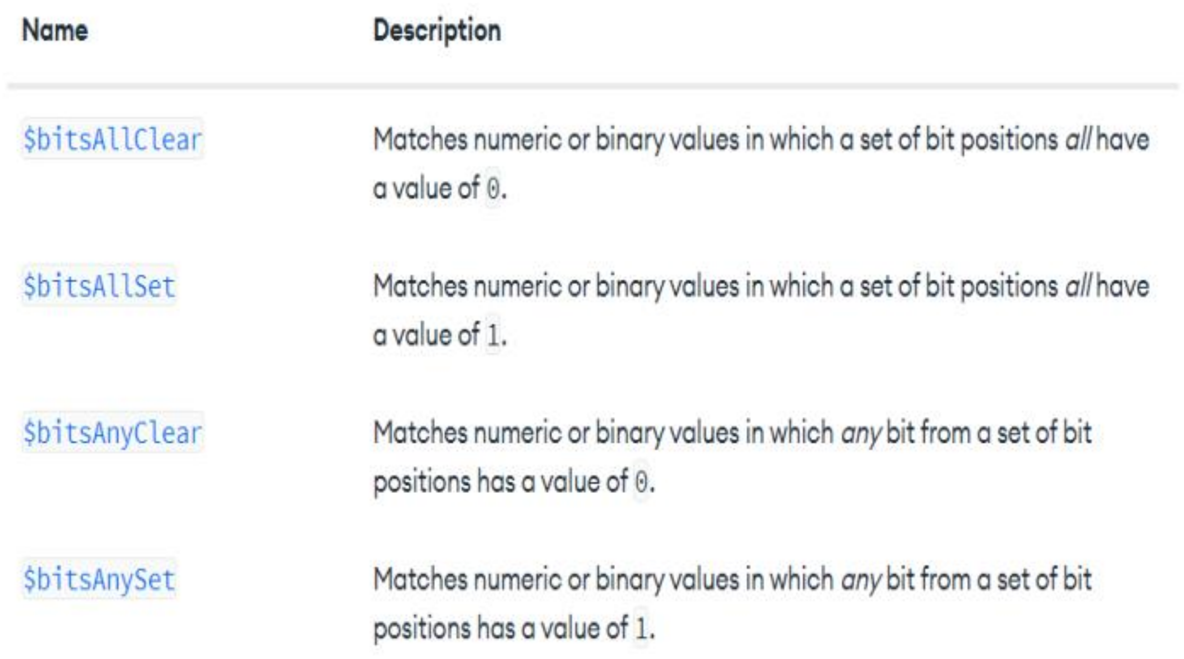
-

Bitwise Values:

- In our example it’s a 32 bit each bit representing different things
- Bitwise value 7 means all access 7->111

Bit 3	Bit 4	Bit 5
cafe	campus	lobby

Bitwise Types:

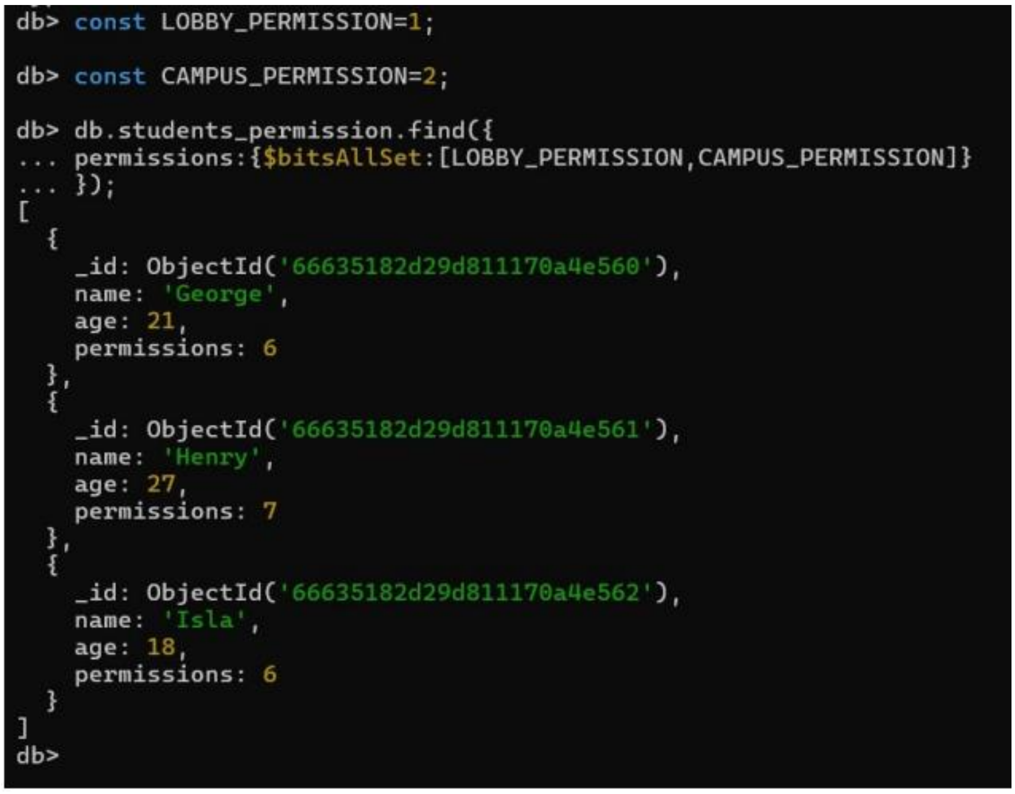


Name	Description
`$bitsAllClear`	Matches numeric or binary values in which a set of bit positions *all* have a value of `0`.
`$bitsAllSet`	Matches numeric or binary values in which a set of bit positions *all* have a value of `1`.
`$bitsAnyClear`	Matches numeric or binary values in which *any* bit from a set of bit positions has a value of `0`.
`$bitsAnySet`	Matches numeric or binary values in which *any* bit from a set of bit positions has a value of `1`.

.

QUERY:

In MongoDB, a query is a command used to retrieve documents from a collection that match certain criteria. Queries are written in MongoDB's query language and can include conditions, projections, and other modifiers to specify exactly which documents to retrieve. MongoDB queries can be complex, allowing for precise filtering, sorting, and aggregation of data.



```
db> const LOBBY_PERMISSION=1;

db> const CAMPUS_PERMISSION=2;

db> db.students_permission.find({
... permissions:{$bitsAllSet:[LOBBY_PERMISSION,CAMPUS_PERMISSION]}
... });
[
  {
    _id: ObjectId('66635182d29d811170a4e560'),
    name: 'George',
    age: 21,
    permissions: 6
  },
  {
    _id: ObjectId('66635182d29d811170a4e561'),
    name: 'Henry',
    age: 27,
    permissions: 7
  },
  {
    _id: ObjectId('66635182d29d811170a4e562'),
    name: 'Isla',
    age: 18,
    permissions: 6
  }
]
db>
```

1.Constant Defination:

const LOBBY_PERMISSION = 1;

const CAMPUS_PERMISSION = 2;

Two constants are defined: LOBBY_PERMISSION with a value of 1 and CAMPUS_PERMISSION with a value of 2. These constants are used to represent specific permissions.

2.Query Execution:

db.students_permission.find({
permissions:{$bitsAllSet:[LOBBY_PERMISSION,CAMPUS_PERMISSION]

This line runs a query on the students_permission collection in the database. The query uses the $bitsAllSet operator to find documents where both the LOBBY_PERMISSION and CAMPUS_PERMISSION bits are set in the permissions field.

Explanation of $bitsAllSet Operator:

• The $bitsAllSet operator checks if all of the specified bit positions in a binary representation of a number are set to 1.

• In this example, LOBBY_PERMISSION (1) and CAMPUS_PERMISSION (2) correspond to bit positions 0 and 1, respectively.

• The permissions field value is checked to ensure both of these bits are set.

The result of the query is an array of documents (JSON objects). Each document represents a student with the following fields:

- **_id**: A unique identifier for the document.
- **name**: The student's name.
- **age:** The student's age.
- **permissions**: A numeric value representing the permissions assigned to the student.

Sample Output:

```
[
 {
  _id: ObjectId('66635182d29d811170a4e560'),
  name: 'George',
  age: 21,
  permissions: 6
 },
 {
  _id: ObjectId('66635182d29d811170a4e561'),
  name: 'Henry',
  age: 27,
  permissions: 7
 },
 {
  _id: ObjectId('66635182d29d811170a4e562'),
  name: 'Isla',
  age: 18,
  permissions: 6
 }
]
```

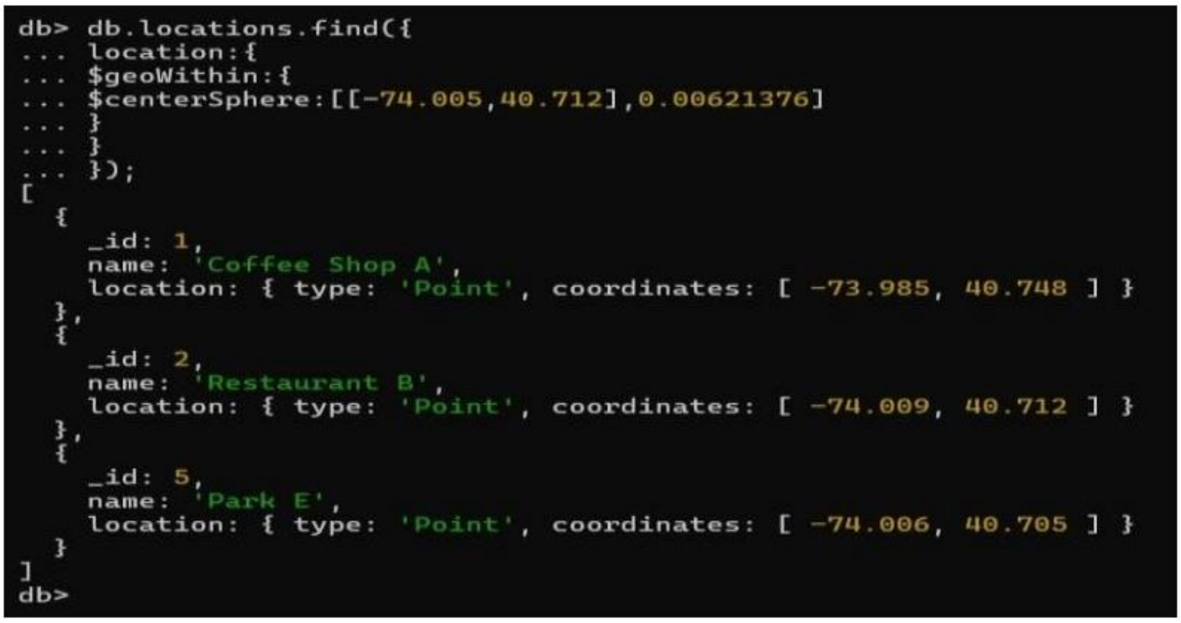
GEOSPATIAL

A geospatial refers to the capability to store and query data based on its geographic location. MongoDB supports various geospatial queries, including finding points within a specified distance of a location, finding objects within a specified polygon, and finding the nearest objects to a location. Geospatial indexes can be created to efficiently perform these types of queries on geospatial data.

```
_id : 1
name : “Coffee Shop A”
location : Object
    type : “Point”
    coordinates : Array(2)
```

GEOSPATIAL QUERY:

A geospatial query is used to retrieve documents based on their geographical location. These queries utilize special operators like **$geoNear**, **$geoWithin**, and **$near** to find.



```
db> db.locations.find({
... location:{
... $geoWithin:{
... $centerSphere:[[-74.005,40.712],0.00621376]
... }
... }
... });
[
  {
    _id: 1,
    name: 'Coffee Shop A',
    location: { type: 'Point', coordinates: [ -73.985, 40.748 ] }
  },
  {
    _id: 2,
    name: 'Restaurant B',
    location: { type: 'Point', coordinates: [ -74.009, 40.712 ] }
  },
  {
    _id: 5,
    name: 'Park E',
    location: { type: 'Point', coordinates: [ -74.006, 40.705 ] }
  }
]
db>
```