CRUD OPERATIONS:

CREATE, READ, UPDATE, and DELETE actions are referred to as CRUD operations in MongoDB.

Create: To add new documents to a collection, use the Create operation.

Read: Data from a collection is retrieved using the Read operation.

Update: The Update operation is used to edit existing documents in a collection.

Delete: A collection of documents can be deleted using the Delete procedure.

Perform CRUD Operations in MongoDB:

1. Create Operations

Method	Description
db.collection.insertOne()	It is used to insert a single document in the collection.
db.collection.insertMany()	It is used to insert multiple documents in the collection.
db.createCollection()	It is used to create an empty collection.

2.Read Operation

Method	Description
db.collection.find()	It is used to retrieve documents from the collection.

3,Update Operations

Method	Description
db.collection.updateOne()	It is used to update a single document in the collection that satisfy the given criteria.
db.collection.updateMany()	It is used to update multiple documents in the collection that satisfy the given criteria.
db.collection.replaceOne()	It is used to replace single document in the collection that satisfy the given criteria.

4. Delete Operations

Method	Description
db.collection.deleteOne()	It is used to delete a single document from the collection that satisfy the given criteria.
db.collection.deleteMany()	It is used to delete multiple documents from the collection that satisfy the given criteria.

Experiment:1

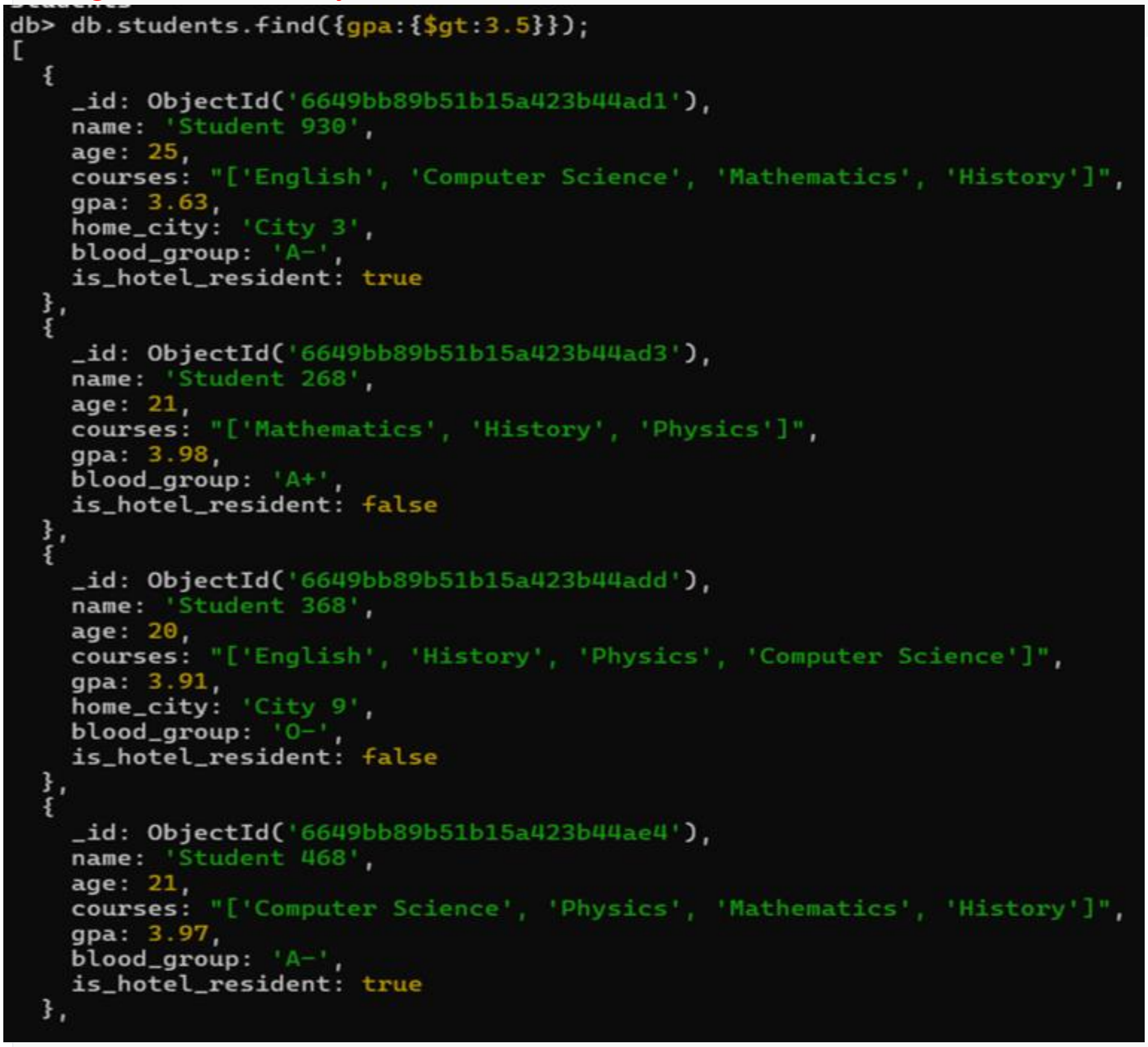
a. Illustration of Where Clause, AND,OR operations in MongoDB.

b. Execute the Commands of MongoDB and operations in MongoDB : Insert, Query, Update, Delete

and Projection.

Where Clause in MongoDB:

In MongoDB, **the where clause** is not used as in **traditional SQL queries**. Instead, MongoDB uses the **$where operator**, which allows for the execution of **JavaScript functions** to perform **complex queries**. However, the use of $where is discouraged due to performance implications, as it can be slower than traditional query methods and may not leverage indexes efficiently.



```
db> db.students.find({gpa:{$gt:3.5}});
[
  {
    _id: ObjectId('6649bb89b51b15a423b44ad1'),
    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('6649bb89b51b15a423b44ad3'),
    name: 'Student 268',
    age: 21,
    courses: "['Mathematics', 'History', 'Physics']",
    gpa: 3.98,
    blood_group: 'A+',
    is_hotel_resident: false
  },
  {
    _id: ObjectId('6649bb89b51b15a423b44add'),
    name: 'Student 368',
    age: 20,
    courses: "['English', 'History', 'Physics', 'Computer Science']",
    gpa: 3.91,
    home_city: 'City 9',
    blood_group: 'O-',
    is_hotel_resident: false
  },
  {
    _id: ObjectId('6649bb89b51b15a423b44ae4'),
    name: 'Student 468',
    age: 21,
    courses: "['Computer Science', 'Physics', 'Mathematics', 'History']",
    gpa: 3.97,
    blood_group: 'A-',
    is_hotel_resident: true
  },
```

The MongoDB query **`db.students.find({gpa:{$gt:3.5}});`** is used to retrieve documents from the `students` collection where the **`gpa`** field is **$gt(greater than)** 3.5.

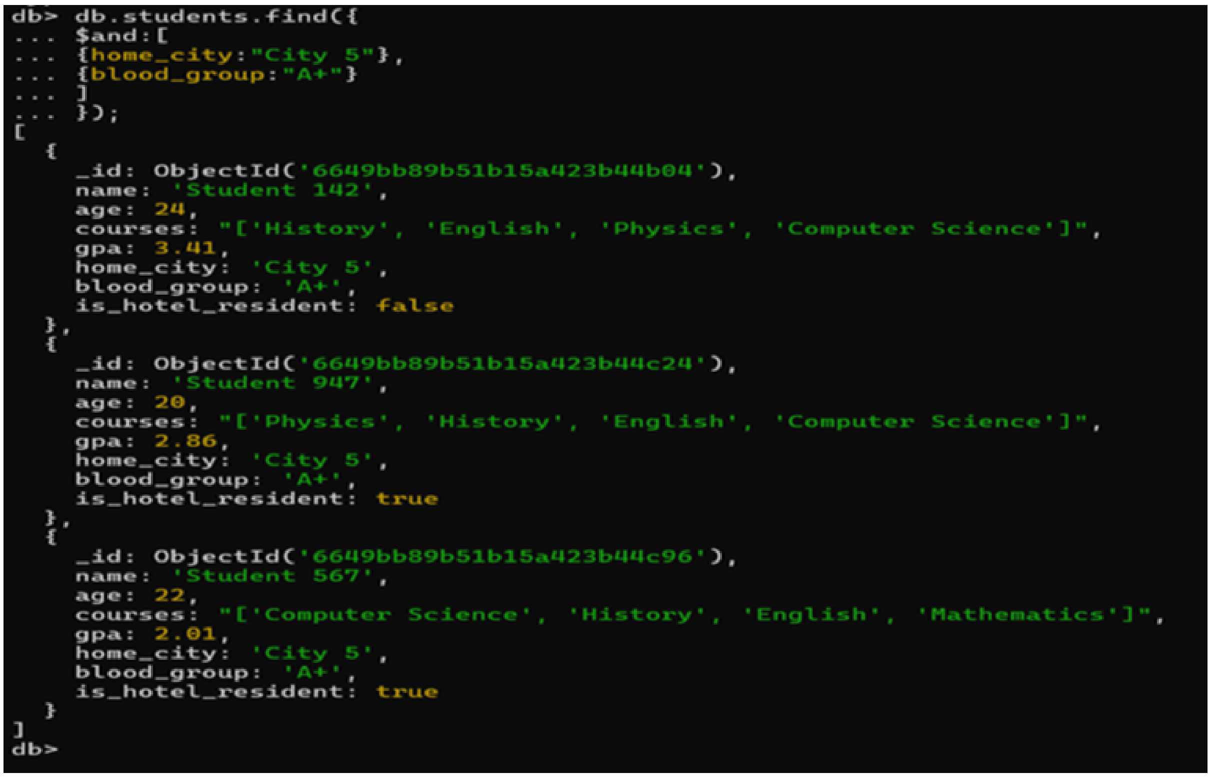
Each document represents a student and includes fields such as `_id` (the unique identifier), **`name`, `age`, `courses`, `gpa`, `home_city`, `blood_group`**, and **`is_hotel_resident`.**

These fields provide information about the students' personal details, academic performance, and other attributes. For example, one student has a **GPA of 3.63,** is from **City 3**, has **blood type A-,** and is a **hotel resident**, while another student has a **GPA of 3.98**, is **not a hotel resident**, and has **blood type A+.**

AND OPERATION IN MONGODB:

In MongoDB, the **`$and` operator** is used **to combine multiple conditions in a query**, requiring that all conditions must be met for a document to be returned. It is often used when you need to specify several criteria for selecting documents from a collection. For example, **`db.collection.find({ $and: [ { field1: value1 }, { field2: value2 } ] })`** will return documents where **`field1` is equal to `value1` and `field2` is equal to `value2`.**

The MongoDB query **`db.students.find({ $and: [{ home_city: "City 5" }, { blood_group:**



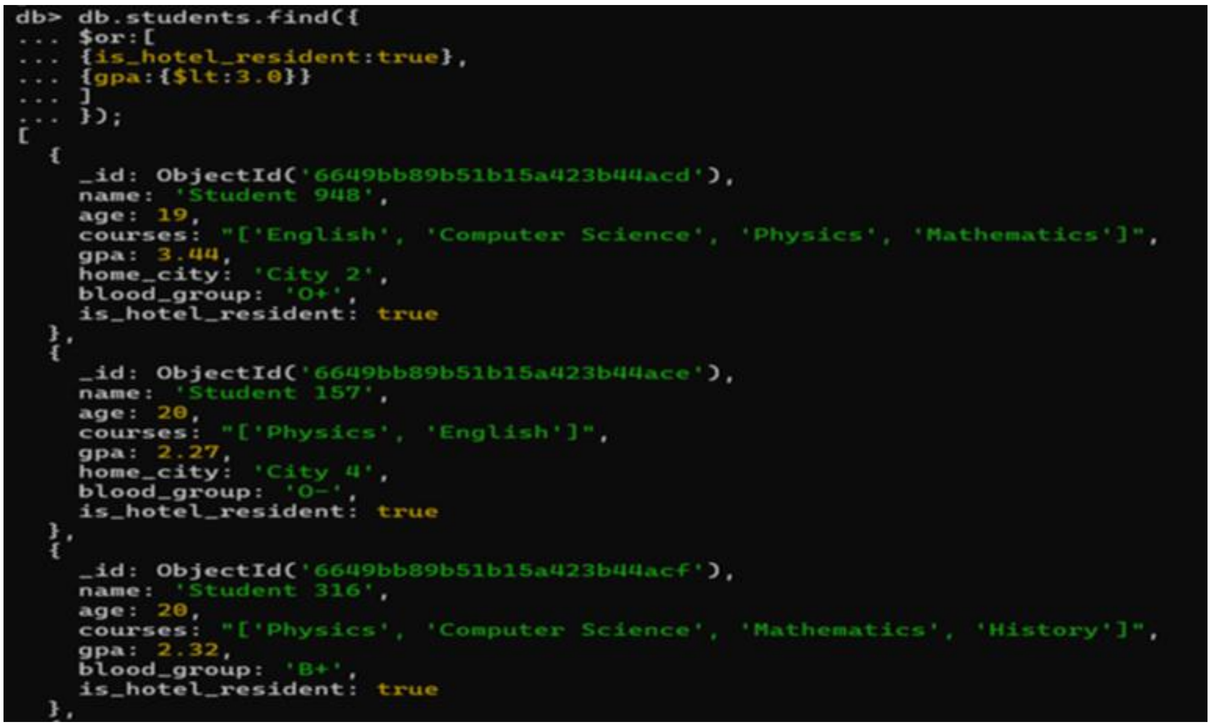
```
db> db.students.find({
... $and:[
... {home_city:"City 5"},
... {blood_group:"A+"}
... ]
... });
[
  {
    _id: ObjectId('6649bb89b51b15a423b44b04'),
    name: 'Student 142',
    age: 24,
    courses: "['History', 'English', 'Physics', 'Computer Science']",
    gpa: 3.41,
    home_city: 'City 5',
    blood_group: 'A+',
    is_hotel_resident: false
  },
  {
    _id: ObjectId('6649bb89b51b15a423b44c24'),
    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('6649bb89b51b15a423b44c96'),
    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
  }
]
db>
```

"A+" }] })`** is used to retrieve documents from the **`students` collection** where both the **`home_city` field is "City 5" and the `blood_group` field is "A+"**. The query returns three documents that match these criteria. Each document represents a student and includes fields such as **`_id` (the unique identifier), `name`, `age`, `courses`, `gpa`, `home_city`, `blood_group`, and `is_hotel_resident`. These fields provide information about the **students' personal details, academic performance, and other attributes**.

For example: one student from **City 5** with **blood type A+** has a **GPA of 3.41** and is **not a hotel resident**, while another student with the **same blood type and city is a hotel resident** with a **lower GPA**.

OR OPERATION IN MONGODB:

In MongoDB, the **`$or` operator** performs a **logical OR operation** between multiple expressions in a query, matching documents that satisfy at least one of the specified conditions. This allows for the construction of queries where documents can meet any of the criteria specified within the `$or` array. For example, **`db.collection.find({ $or: [ { field1: value1 }, { field2: value2 } ] })`** will return documents where `field1` is equal to `value1` or `field2` is equal to `value2`.



```
db> db.students.find({
... $or:[
... {is_hotel_resident:true},
... {gpa:{$lt:3.0}}
... ]
... });
[
  {
    _id: ObjectId('6649bb89b51b15a423b44acd'),
    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('6649bb89b51b15a423b44ace'),
    name: 'Student 157',
    age: 20,
    courses: "['Physics', 'English']",
    gpa: 2.27,
    home_city: 'City 4',
    blood_group: 'O-',
    is_hotel_resident: true
  },
  {
    _id: ObjectId('6649bb89b51b15a423b44acf'),
    name: 'Student 316',
    age: 20,
    courses: "['Physics', 'Computer Science', 'Mathematics', 'History']",
    gpa: 2.32,
    blood_group: 'B+',
    is_hotel_resident: true
  },
```

The MongoDB query **`db.students.find({ $or: [ { is_hotel_resident: true }, { gpa: { $lt: 3.0 } } ] })`** is used to retrieve documents from the `students` collection where either the **`is_hotel_resident` field is true or the `gpa` field is less than 3.0.** The query returns multiple documents that match these criteria. Each document represents a student and includes fields such as **`_id` (the unique identifier), `name`, `age`, `courses`, `gpa`, `home_city`, `blood_group`, and `is_hotel_resident`**. These fields provide information about the students' personal details, academic performance, and other attributes.

For example, :one student is a **hotel resident with a GPA of 3.44,** while another student has a **GPA of 2.27 and is also a hotel resident**.

CRUD:

C - Create / Insert

R – Remove

U - update

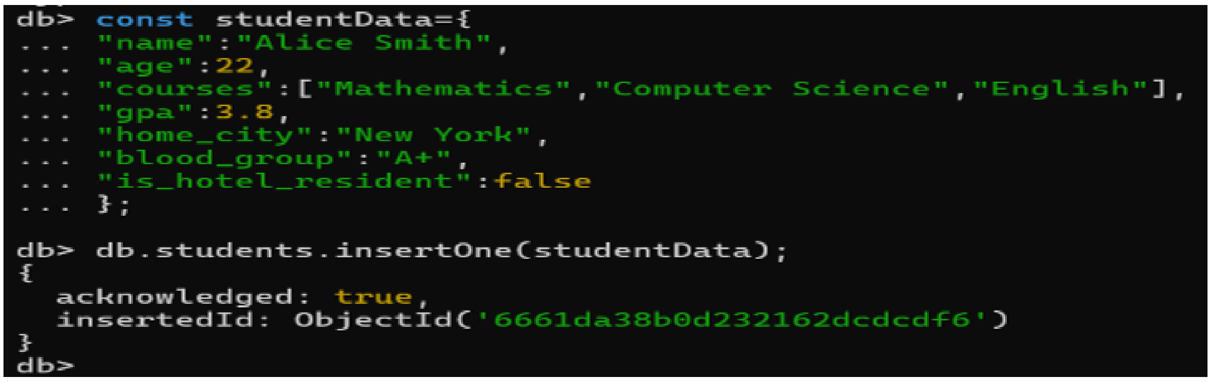
D - Delete This is applicable for a Collection (Table) or a Document (Row)

INSERTONE:

In MongoDB, the `` `insertOne` `` method is used to **insert a single document into a collection**. It takes a single parameter, which is an object representing the document to be inserted. Upon **successful insertion**, it returns an **object with an `acknowledged` field set to true** and

an **`insertedId` field** containing the **unique identifier (`ObjectId`)** assigned to the newly inserted document. **This method is useful for adding individual documents to a collection in MongoDB.**

The JavaScript object **`studentData`** contains information about a student, including their **name, age, courses, GPA, home city, blood group, and hotel residency status**. Using the **`insertOne`** method in MongoDB, this data is inserted into the `students` collection as a single document. The `insertedId` field in the returned object confirms that the document was successfully inserted and includes the unique identifier (`ObjectId`) assigned to the



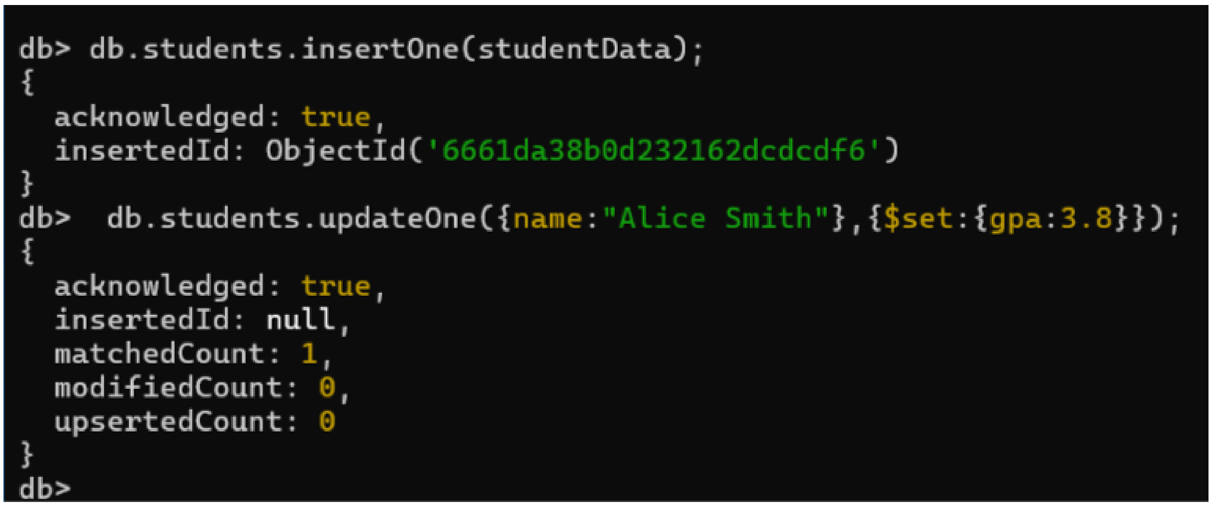
```
db> const studentData={
... "name":"Alice Smith",
... "age":22,
... "courses":["Mathematics","Computer Science","English"],
... "gpa":3.8,
... "home_city":"New York",
... "blood_group":"A+",
... "is_hotel_resident":false
... };

db> db.students.insertOne(studentData);
{
  acknowledged: true,
  insertedId: ObjectId('6661da38b0d232162dcdcdf6')
}
db>
```

newly inserted document. This identifier can be used to uniquely identify and retrieve the inserted document from the collection.

UPDATEONE:

In MongoDB, the **`updateOne`** method is used **to update a single document that matches a**



```
db> db.students.insertOne(studentData);
{
  acknowledged: true,
  insertedId: ObjectId('6661da38b0d232162dcdcdf6')
}
db>  db.students.updateOne({name:"Alice Smith"},{$set:{gpa:3.8}});
{
  acknowledged: true,
  insertedId: null,
  matchedCount: 1,
  modifiedCount: 0,
  upsertedCount: 0
}
db>
```

specified filter. It takes two parameters: **a filter object to identify the document to update, and an update object containing the modifications to apply.**

Only the first document that matches the filter is updated. The method returns an object indicating whether the **update operation was acknowledged and the number of documents modified.**

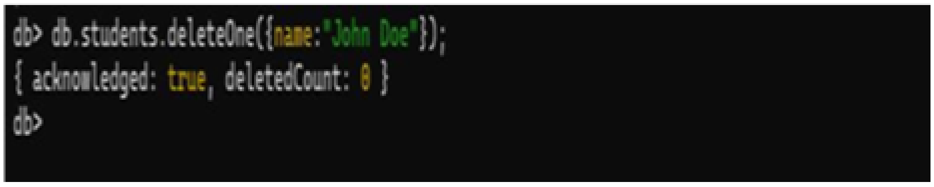
The first MongoDB operation inserts a new document into the `students` collection using

`insertOne`, which is acknowledged with **`acknowledged: true`** and provides the **`insertedId`** of the new document. The second operation uses **`updateOne`** to find a document with the name **"Alice Smith**" and update its **`gpa` field to 3.8.** This update operation is also acknowledged with `acknowledged: true`, and it indicates that one document was matched **(`matchedCount: 1`)**, but no actual modifications were made **(`modifiedCount: 0`)** because the existing **`gpa` value was already 3.8.**

DELETEONE:

In MongoDB, the **`deleteOne` method is used to delete a single document that matches a specified filter.** It takes a filter object to identify the document to **delete and removes only the first document that matches the filter**. The method returns an object indicating whether the delete operation was acknowledged and the number of documents deleted, which is **typically either 0 or 1.**

The MongoDB **`deleteOne`** method attempts to delete a single document from the `students` collection where the `name` field is "John Doe". The operation is acknowledged

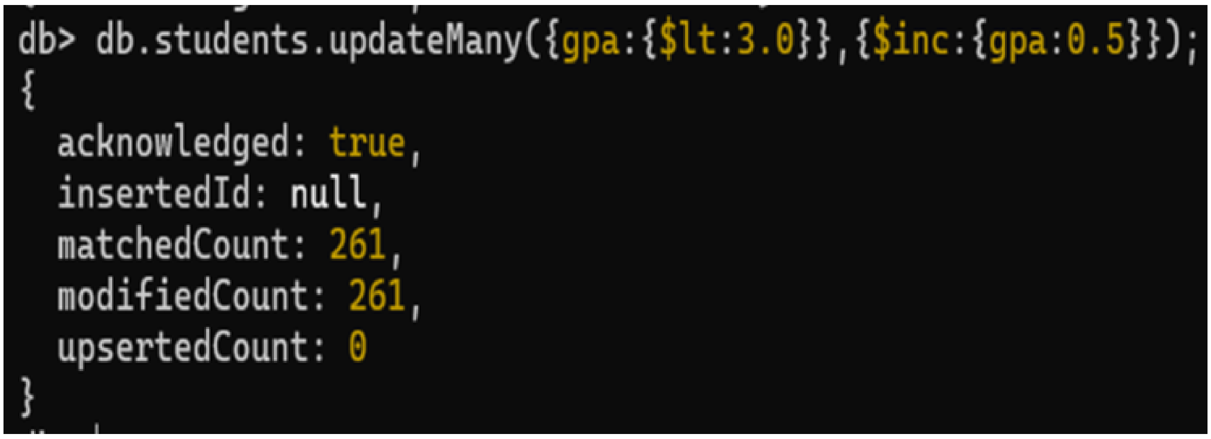


```
db> db.students.deleteOne({name:"John Doe"});
{ acknowledged: true, deletedCount: 0 }
db>
```

(`acknowledged: true`), but the `**deletedCount` is 0**, indicating that no document was deleted. This could happen if there is no document in the collection that matches the specified filter. **The method is designed to delete only the first document that matches the filter, so if there are multiple documents with the same name, only the first one encountered would be deleted.**

UPDATEMANY:

In MongoDB, the `**updateMany**` method **is used to update multiple documents that match a specified filter**. It takes two parameters: a filter object to identify the documents to update, and an update object containing the modifications to apply. All documents that match the filter will be updated. The method returns an object indicating whether the update operation was acknowledged and the number of documents modified.

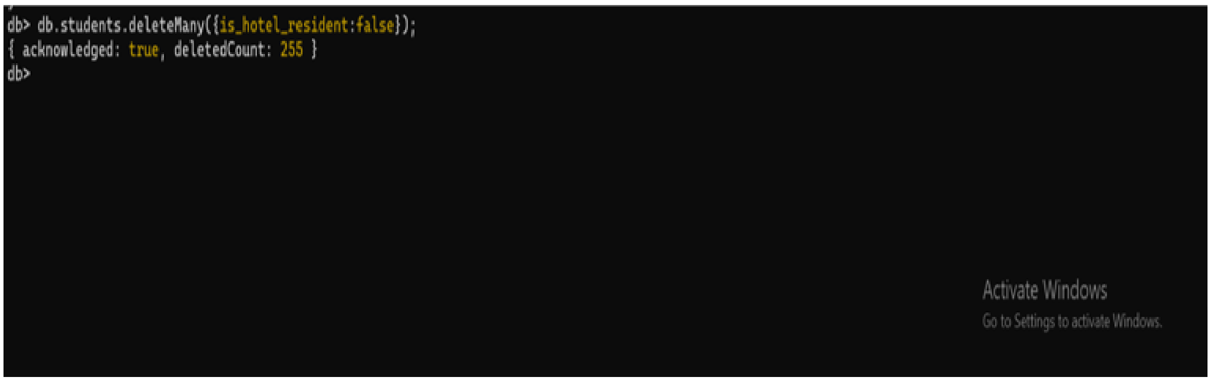


```
db> db.students.updateMany({gpa:{$lt:3.0}},{$inc:{gpa:0.5}});
{
  acknowledged: true,
  insertedId: null,
  matchedCount: 261,
  modifiedCount: 261,
  upsertedCount: 0
}
```

The MongoDB operation `**updateMany**` is used to update multiple documents in the `students` collection where the **`gpa` is less than 3.0**. The **update increments the `gpa` field by 0.5 for each matched document**. The operation is acknowledged with `acknowledged: true` and indicates that 261 documents were matched **(`matchedCount: 261`)** and modified **(`modifiedCount: 261`).** This means that all documents with a `**gpa` less than 3.0** were successfully updated to increase their `**gpa` by 0.5**.

DELETEMANY:

In MongoDB, the `**deleteMany**` method is used to **delete multiple documents that match a specified filter**. It takes a filter object to identify the documents to **delete, and removes all documents that match the filter**. The method returns an object indicating whether the delete operation was acknowledged and the number of documents deleted.

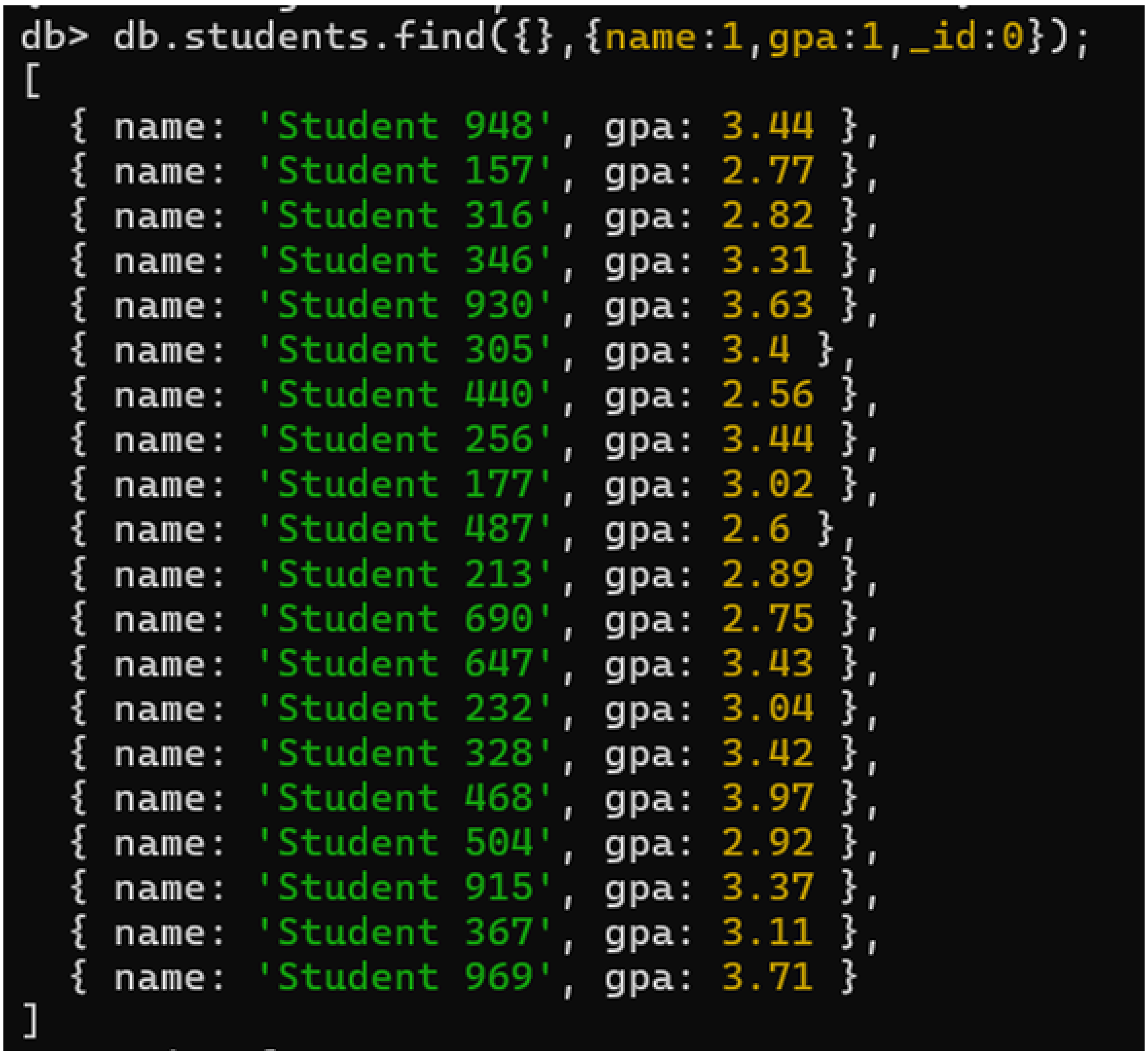


```
db> db.students.deleteMany({is_hotel_resident:false});
{ acknowledged: true, deletedCount: 255 }
db>
```

The MongoDB operation **`deleteMany`** is used to delete multiple documents from the `students` collection where the `is_hotel_resident` field is `false`. The operation is acknowledged with `**acknowledged: true`** and indicates that **255 documents were deleted (`deletedCount: 255`)**. This means that all documents in the collection where `is_hotel_resident` is `false` were successfully removed.

PROJECTION:

In MongoDB**, projection is the process of specifying which fields of a document should be returned in the query result**. It allows you to **retrieve only the necessary data, reducing network traffic and improving performance**. By default, queries return all fields in matching documents, but projection allows you to limit the fields returned to only those you specify.



```
db> db.students.find({},{name:1,gpa:1,_id:0});
[
  { name: 'Student 948', gpa: 3.44 },
  { name: 'Student 157', gpa: 2.77 },
  { name: 'Student 316', gpa: 2.82 },
  { name: 'Student 346', gpa: 3.31 },
  { name: 'Student 930', gpa: 3.63 },
  { name: 'Student 305', gpa: 3.4 },
  { name: 'Student 440', gpa: 2.56 },
  { name: 'Student 256', gpa: 3.44 },
  { name: 'Student 177', gpa: 3.02 },
  { name: 'Student 487', gpa: 2.6 },
  { name: 'Student 213', gpa: 2.89 },
  { name: 'Student 690', gpa: 2.75 },
  { name: 'Student 647', gpa: 3.43 },
  { name: 'Student 232', gpa: 3.04 },
  { name: 'Student 328', gpa: 3.42 },
  { name: 'Student 468', gpa: 3.97 },
  { name: 'Student 504', gpa: 2.92 },
  { name: 'Student 915', gpa: 3.37 },
  { name: 'Student 367', gpa: 3.11 },
  { name: 'Student 969', gpa: 3.71 }
]
```

The MongoDB query `**db.students.find({}, { name: 1, gpa: 1, _id: 0 })**` retrieves documents from the `students` collection but includes only the `name` and `gpa` fields in the result. The **`_id` field**, which is included in all documents by default, is **excluded (`_id: 0`) 9 from the result**. This projection results in a list of objects where each object contains only the **`name` and `gpa` fields** for each student, without the **`_id` field.**