MONGO DB: REPORT 3

WHERE, AND, OR & CRUD OPERATIONS:

WHERE Operation->

- Given a Collection you want to FILTER a subset based on a condition. That is the place WHERE is used.

```
// Find all students with GPA greater than 3.5
db.students.find({ gpa: { $gt: 3.5 } });

// Find all students from "City 3"
db.students.find({ home_city: "City 3" });
```

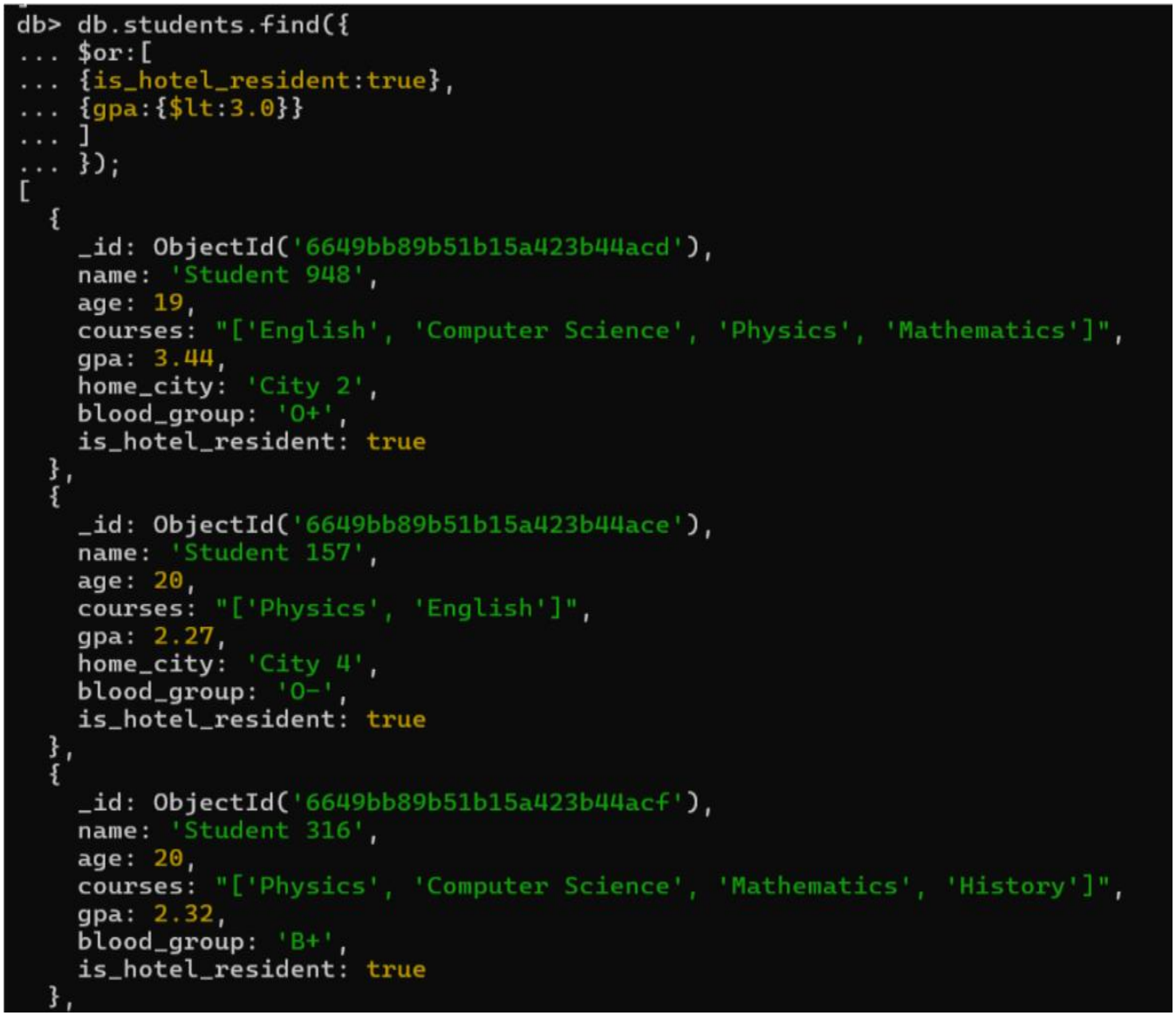
AND Operation->

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

```
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>
```

OR Operation->

- Given a Collection you want to FILTER a subset based on multiple conditions but Any One is Sufficient.



```
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
  },
```

CRUD Operation->

- C - Create / Insert
- R - Remove
- U - update
- D - Delete

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

Insert Documents

There are 2 methods to insert documents into a MongoDB database.

`insertOne()`

To insert a single document, use the `insertOne()` method.

This method inserts a single object into the database.

```
// Define the student data as a JSON document
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
};

// Insert the student document into the "students" collection
db.students.insertOne(studentData);
```

Update Documents

To update an existing document we can use the `updateOne()` or `updateMany()` methods.

The first parameter is a query object to define which document or documents should be updated.

The second parameter is an object defining the updated data.

`updateOne()`

The `updateOne()` method will update the first document that is found matching the provided query.

Let's see what the "like" count for the post with the title of "Post Title 1":

```
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>
```

Delete Documents

We can delete documents by using the methods `deleteOne()` or `deleteMany()`.

These methods accept a query object. The matching documents will be deleted.

`deleteOne()`

The `deleteOne()` method will delete the first document that matches the query provided.

Example

```
db.posts.deleteOne({ title: "Post Title 5" })
```

`deleteMany()`

The `deleteMany()` method will delete all documents that match the query provided.

Example

```
db.posts.deleteMany({ category: "Technology" })
```

PROJECTION

MongoDB provides a special feature that is known as **Projection**. It allows you to select only the necessary data rather than selecting whole data from the document. For example, a document contains 5 fields, i.e.,

```
{
name: "Roma",
age: 30,
branch: EEE,
department: "HR",
salary: 20000
}
```

But we only want to display the *name* and the *age* of the employee rather than displaying whole details. Now, here we use projection to display the name and age of the employee.
One can use projection
with `db.collection.find()` method. In this method, the second parameter is the projection parameter, which is used to specify which fields are returned in the matching documents.

Syntax:

```
db.collection.find({}, {field1: value2,
field2: value2, ..})
```

- If the value of the field is set to 1 or true, then it means the field will include in the return document.
- If the value of the field is set to 0 or false, then it means the field will not include in the return document.
- You are allowed to use projection operators, but `find()` method does not support following projection operators, i.e., `$`, `$elemMatch`, `$slice`, and `$meta`.
- There is no need to set `_id` field to 1 to return `_id` field, the `find()` method always return `_id` unless you set a `_id` field to 0.

```
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 }
]
```