# CLASS 3: WHERE, AND, OR & CRUD:

## INTRODUCTION:

In this class, we will dive deeper into MongoDB queries, focusing on filtering documents using the where clause, combining multiple conditions with AND and OR operators, and performing CRUD operations (Create, Read, Update, Delete).

## AGENDA:

- Understanding the where Clause
- Using AND Operator for Compound Queries
- Utilizing OR Operator for Flexible Queries
- Performing CRUD Operations:
- Create: Inserting New Documents
- Read: Retrieving Documents from Collections
- Update: Modifying Existing Documents
- Delete: Removing Documents from Collections

## LEARNING OBJECTIVES

- Understand how to filter documents using the where clause.
- Learn how to combine multiple conditions using AND and OR operators.
- Gain proficiency in performing CRUD operations in MongoDB

## RESOURCES:

- MongoDB documentation: https://docs.mongodb.com/manual/
- Sample dataset: https://drive.google.com/drive/folders/1jFBR0g8WpflBJC2r6yUEoS-2M5HEOQVP

🚀 Supercharge Your MongoDB Skills Today! 🚀
Today's session is your ticket to MongoDB mastery. Buckle up, tighten your seatbelts, and get ready to embark on an exhilarating journey through MongoDB queries and CRUD operations. Let's make today the day you conquer MongoDB once and for all!

Let's Dive into MongoDB Filters:
Now that we're geared up and ready to roll, let's kick things off by delving into the fundamental concept of MongoDB filters. Filters act as our guiding compass, allowing us to navigate through vast collections of data and pinpoint exactly what we're looking for.


CRUD operations in MongoDB are the basic operations you can perform to interact with the database. CRUD stands for Create, Read, Update, and Delete. These operations allow you to manage and manipulate data within your MongoDB collections. Here's a detailed explanation of each operation with examples:

## CREATE:

The **insertOne()** and **insertMany()** methods are used to create new documents in a collection. The insertOne function in MongoDB is used to insert a single document into a collection. It is one of the fundamental operations for adding data to a MongoDB database. Here's a detailed explanation of how insertOne works, along with an example:

**Syntax:**

**db.collection.insertOne(document, options)**

- db.collection: The collection in which you want to insert the document.
- document: The document to be inserted. This is a JSON object.
- options (optional): Additional options for the insertion.

```
db> const studentData={ "name":"Alice", "age":20, "courses":["Mathematics","History"],"gpa":3.5,
"home_city":"City 1","blood_group":"A+","is_hotel_resident":true};

db> db.students.insertOne(studentData);
{
  acknowledged: true,
db>
db> db.students.find().count()
501
db>
```

When you run the above command, MongoDB will insert the document into the students collection. The _id field will be automatically added by MongoDB if it's not provided, which uniquely identifies the document in the collection.

```
db> db.students.insertOne(studentData);
{
  acknowledged: true,
  insertedId: ObjectId('6668f62128b5145253cdcdf7')
}
db> db.students.find().count()
503
db>
```

In MongoDB, repeatedly executing the insertOne function with the same document will lead to the insertion of multiple documents, each with a unique _id field, even if the rest of the document's data is identical. This is because MongoDB automatically generates a unique _id for each inserted document if you don't provide one.

The insertMany function in MongoDB is used to insert multiple documents into a collection at once. This function is useful when you need to add several documents efficiently, as it reduces the number of network round-trips compared to inserting each document individually using insertOne.

**Syntax:db.collection.insertMany(documents, options)**

- MongoDB automatically adds an _id field to each document if it's not provided.
- The function returns an object with acknowledged and insertedIds properties.
- Optional options parameter can be used to specify insertion behavior like ordered.

```
db> db.students.find().count()
503
db> db.students.insertMany([
... {
... "name":"Alice",
... "age":20,
... "courses":["Mathematics","Computer Science"],
... "home_city":"City 1",
... "blood_group":"A+",
... "is_hotel_resident":true
... },
... {
... "courses":["History","Mathematics"],
... "home_city":"City 2",
... "blood_group":"B+",
... "is_hotel_resident":false
... },
... {
... "name":"Charlie",
... "age":23,
... "courses":["Computer Science","History"],
... }
... ])
{
  acknowledged: true,
  insertedIds: {
    '0': ObjectId('6668fa2628b5145253cdcdf9'),
    '1': ObjectId('6668fa2628b5145253cdcdfa'),
    '2': ObjectId('6668fa2628b5145253cdcdfb')
  }
}
db> db.students.find().count()
506
```

## READ:

In MongoDB, the primary function used to read documents from a collection is the find function. This function allows you to query the database for documents that match specified criteria and can include various options to control the output, such as sorting, projection, and limiting the number of results. Here, we'll cover the basics of the find function and provide examples to illustrate its usage.

# FIND():

The find function in MongoDB is one of the most commonly used functions, allowing you to query and retrieve documents from a collection. It is highly flexible and powerful, enabling you to specify a wide range of query criteria to filter the documents returned.

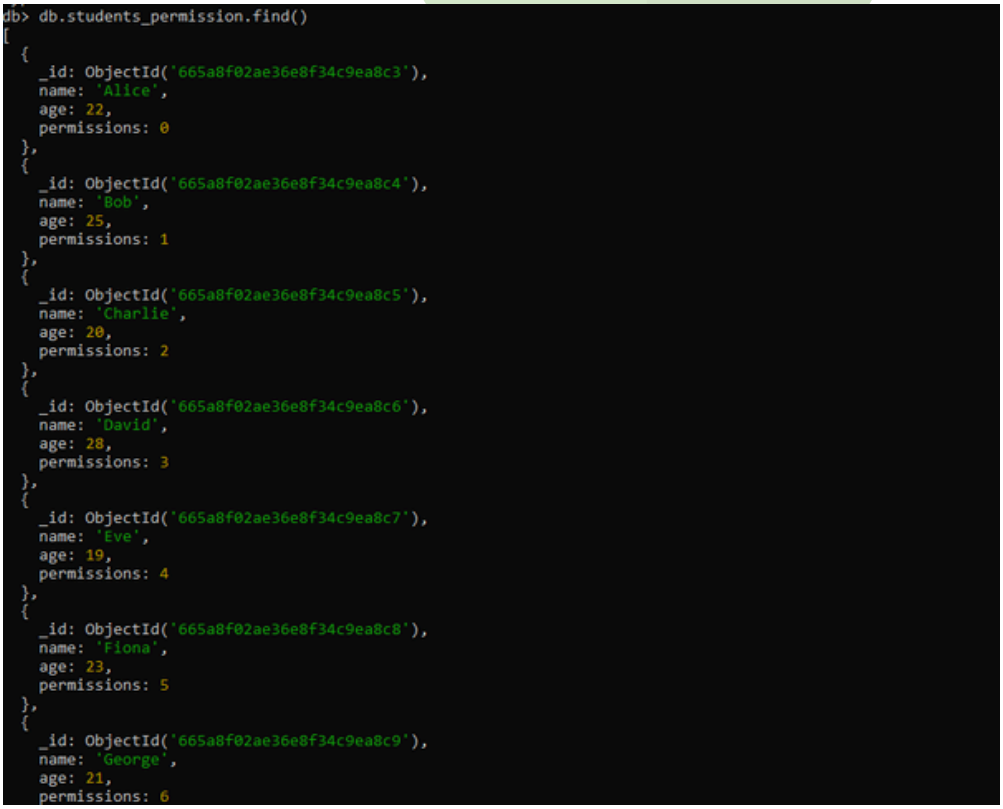
The basic syntax for the find function is: **db.collection.find(query, projection)**
**query:** Specifies selection criteria using query operators.
**projection (optional):** Specifies which fields to include or exclude in the returned documents.

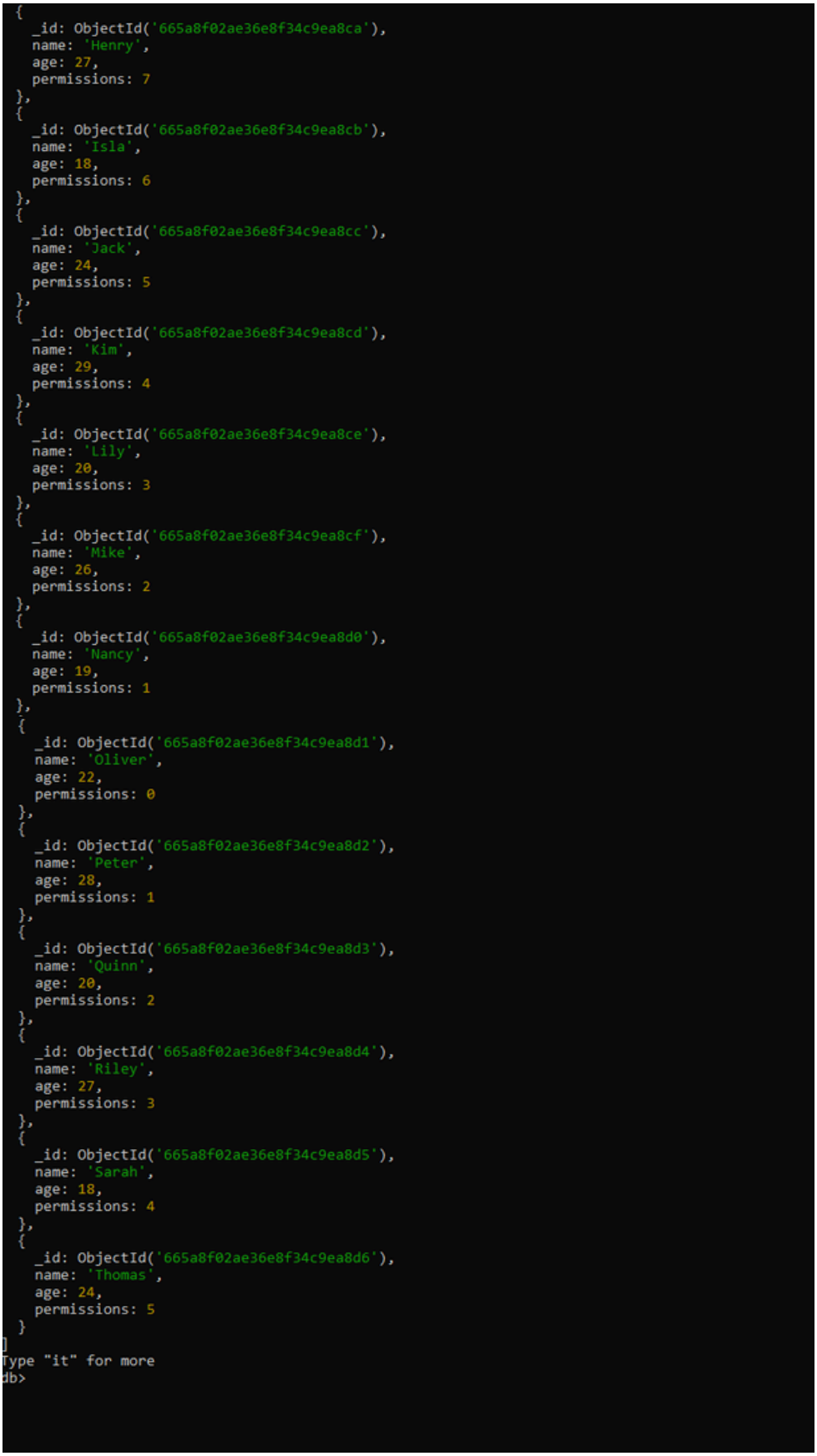
**Retrieve All Documents:** To retrieve all documents in a collection:

**Syntax:**db.myCollection.find().
This returns all documents in myCollection.

To illustrate the use of the find function in MongoDB with a collection named students_permission, let's go through an example. Suppose our **students_permission** collection has documents with fields _id, name, age, and permissions.
Here's an example of what you might see in the MongoDB shell when you run the find command:
**Explanation**:

- **Command:** db.students_permission.find()
- **Result:** A list of all documents in the students_permission collection.
- **Fields:** Each document includes _id, name, age, and permissions.






This basic example demonstrates how to retrieve all documents from a collection using the find function in MongoDB. You can extend this by adding more complex query criteria, projections, and cursor methods to filter and manipulate the data as needed.

## EXPLORING THE MONGODB FIND FUNCTION: FILTERS AND PROJECTIONS

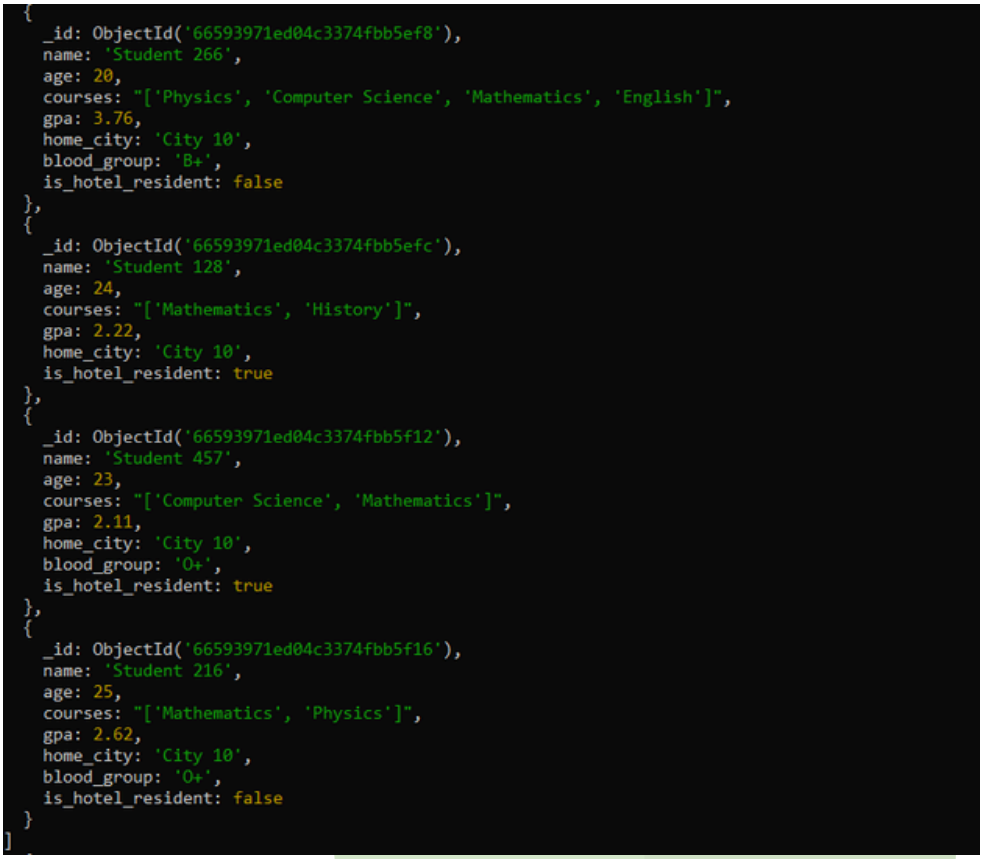
Understanding how to use filters and projections with the find function can greatly enhance your ability to retrieve and manipulate data efficiently.

**Filtering Based on a Single Field:**

To filter documents based on a single field, specify the field and value in the find query.
For example: Find All Students from "City 10":

```
db> db.students.find({"home_city":"City 10"})
[
  {
    _id: ObjectId('66593971ed04c3374fbb5e68'),
    name: 'Student 440',
    age: 21,
    courses: "['History', 'Physics', 'Computer Science']",
    gpa: 2.06,
    home_city: 'City 10',
    blood_group: 'O-',
    is_hotel_resident: true
  },
  {
    _id: ObjectId('66593971ed04c3374fbb5e6b'),
    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('66593971ed04c3374fbb5e7b'),
    name: 'Student 127',
    age: 19,
    courses: "['History', 'English', 'Computer Science', 'Mathematics']",
    gpa: 2.56,
    home_city: 'City 10',
    blood_group: 'AB+',
    is_hotel_resident: false
  },
  {
    _id: ObjectId('66593971ed04c3374fbb5e8b'),
    name: 'Student 533',
    age: 21,
    courses: "['English', 'Physics']",
    gpa: 3.26,
    home_city: 'City 10',
    blood_group: 'AB-',
    is_hotel_resident: false
  },
  {
    _id: ObjectId('66593971ed04c3374fbb5e8d'),
    name: 'Student 404',
    age: 25,
    courses: "['Mathematics', 'Physics', 'Computer Science', 'English']",
    gpa: 2.59,
    home_city: 'City 10',
    blood_group: 'AB-',
    is_hotel_resident: false
  },
  {
    _id: ObjectId('66593971ed04c3374fbb5e96'),
    name: 'Student 860',
    age: 25,
    courses: "['Physics', 'English', 'Mathematics', 'History']",
    gpa: 2.03,
    home_city: 'City 10',
    blood_group: 'O-',
    is_hotel_resident: true
  },
  {
    _id: ObjectId('66593971ed04c3374fbb5e9a'),
    name: 'Student 522',
    age: 18,
    courses: "['Computer Science', 'History', 'English', 'Mathematics']",
    gpa: 3.28,
    home_city: 'City 10',
    blood_group: 'O+',
    is_hotel_resident: false
  },
  {
    _id: ObjectId('66593971ed04c3374fbb5e9b'),
    name: 'Student 789',
    age: 19,
    courses: "['Physics', 'English']",
    gpa: 2.71,
    home_city: 'City 10',
    blood_group: 'B-',
    is_hotel_resident: false
  },
```

```
{
  _id: ObjectId('66593971ed04c3374fbb5ea9'),
  name: 'Student 925',
  age: 20,
  courses: "['English', 'History', 'Mathematics']",
  gpa: 2.06,
  home_city: 'City 10',
  blood_group: 'AB+',
  is_hotel_resident: false
},
{
  _id: ObjectId('66593971ed04c3374fbb5eb0'),
  name: 'Student 781',
  age: 22,
  courses: "['History', 'Mathematics', 'Physics']",
  gpa: 2.41,
  home_city: 'City 10',
  blood_group: 'O+',
  is_hotel_resident: false
},
{
  _id: ObjectId('66593971ed04c3374fbb5ebe'),
  name: 'Student 857',
  age: 20,
  courses: "['Physics', 'English']",
  gpa: 3.79,
  home_city: 'City 10',
  blood_group: 'AB-',
  is_hotel_resident: true
},
{
  _id: ObjectId('66593971ed04c3374fbb5ec1'),
  name: 'Student 212',
  age: 25,
  courses: "['History', 'Mathematics', 'Computer Science', 'English']",
  gpa: 3.73,
  home_city: 'City 10',
  blood_group: 'AB+',
  is_hotel_resident: false
},
{
  _id: ObjectId('66593971ed04c3374fbb5ec5'),
  name: 'Student 110',
  age: 24,
  courses: "['Physics', 'History', 'English', 'Mathematics']",
  gpa: 2.71,
  home_city: 'City 10',
  blood_group: 'B+',
  is_hotel_resident: true
},
{
  _id: ObjectId('66593971ed04c3374fbb5ec7'),
  name: 'Student 920',
  age: 25,
  courses: "['Mathematics', 'History', 'English']",
  gpa: 2.4,
  home_city: 'City 10',
  blood_group: 'B+',
  is_hotel_resident: true
},
{
  _id: ObjectId('66593971ed04c3374fbb5ece'),
  name: 'Student 821',
  age: 25,
  courses: "['Mathematics', 'Physics']",
  gpa: 2.63,
  home_city: 'City 10',
  blood_group: 'O+',
  is_hotel_resident: false
},
{
  _id: ObjectId('66593971ed04c3374fbb5ed3'),
  name: 'Student 780',
  age: 25,
  courses: "['English', 'History', 'Computer Science', 'Physics']",
  gpa: 3.38,
  home_city: 'City 10',
  blood_group: 'A+',
  is_hotel_resident: true
},
```




The db.students.find({"home_city":"City 10"}) command in MongoDB is used to retrieve documents from the students collection where the home_city field is equal to "City 10". Here's a breakdown of this command:

**Explanation**

- db.students.find(): This is the basic structure of the find command in MongoDB, used to query documents from a collection. In this case, students is the collection name.
- {"home_city": "City 10"}: This is the query parameter passed to the find function. It specifies the condition that the home_city field must be equal to "City 10".

**What It Does**

When executed, this command will:

1. Search the students collection.
2. Find all documents where the home_city field has the value "City 10".
3. Return a cursor to all matching documents.

By using the find function in this way, you can effectively control which fields are included in your query results, making it easier to work with the specific data you need.

**Extending to More Complex Queries:**

Once you are comfortable with basic queries, you can start using comparison operators, logical operators, and projections.

**Comparison Operators in MongoDB:**

Comparison operators in MongoDB allow you to filter documents based on specific conditions. These operators can be used in queries to find documents that meet certain criteria. The most common comparison operators are:

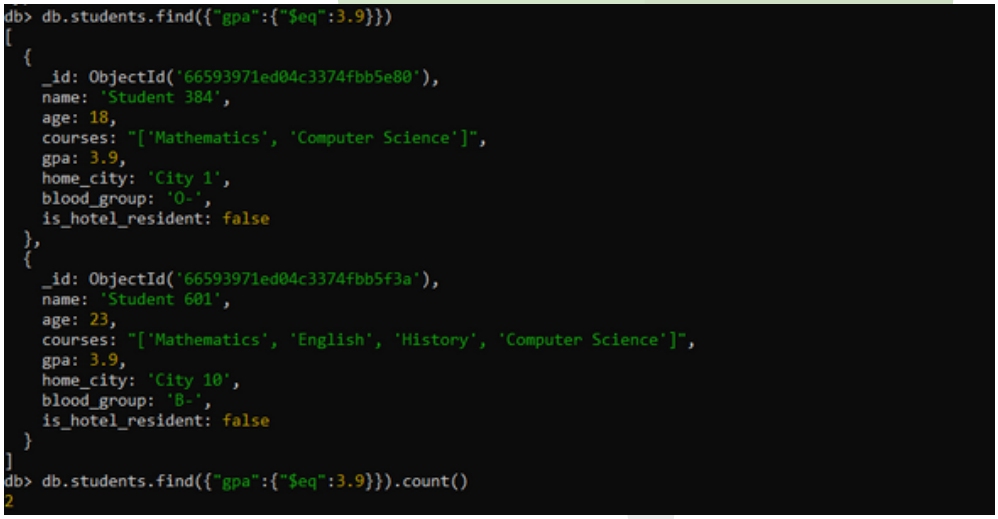
- $eq: Equal to
- $ne: Not equal to
- $gt: Greater than
- $gte: Greater than or equal to
- $lt: Less than
- $lte: Less than or equal to

Let's use the students collection as an example to demonstrate how these operators work.

**1.$eq: Equal to:**

Finds documents where the field is equal to a specified value.

When you execute the query **db.students.find({ "gpa": { "$eq": 3.9 } })**, MongoDB will search the students collection and return documents where the gpa field is equal to 3.9. In this case, the result will be:



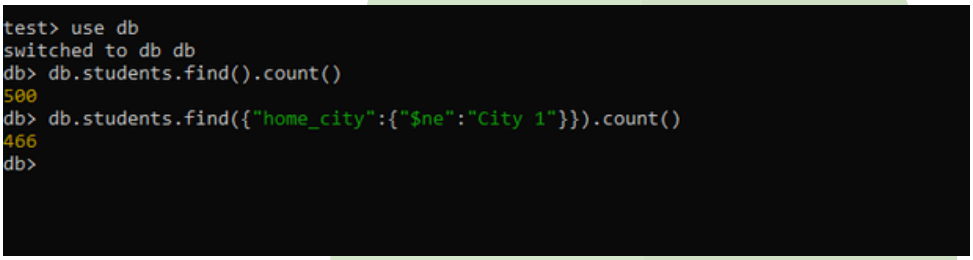

This query demonstrates how to use the equality comparison operator ($eq) in MongoDB to retrieve documents based on specific field values.

**2.$ne: Not equal:**

Finds documents where the field is not equal to a specified value. The query db.students.find({ "home_city": { "$ne": "City 2" } }) is used to retrieve documents from the students collection where the value of the home_city field is not equal to "City 2". Let's break down the components of this query:

- db.students.find(): This is the find function in MongoDB used to search for documents in the students collection.
- { "home_city": { "$ne": "City 2" } }: This is the query parameter passed to the find function. It specifies the condition that the home_city field must not be equal to "City 2".
  - "home_city": This specifies the field on which the condition is applied.
  - { "$ne": "City 2" }: This is an embedded query document that specifies the inequality condition using the $ne operator. The $ne operator selects documents where the value of the field is not equal to the specified value, in this case, "City 2".

**.count()**: This is a method used to count the number of documents returned by the query.

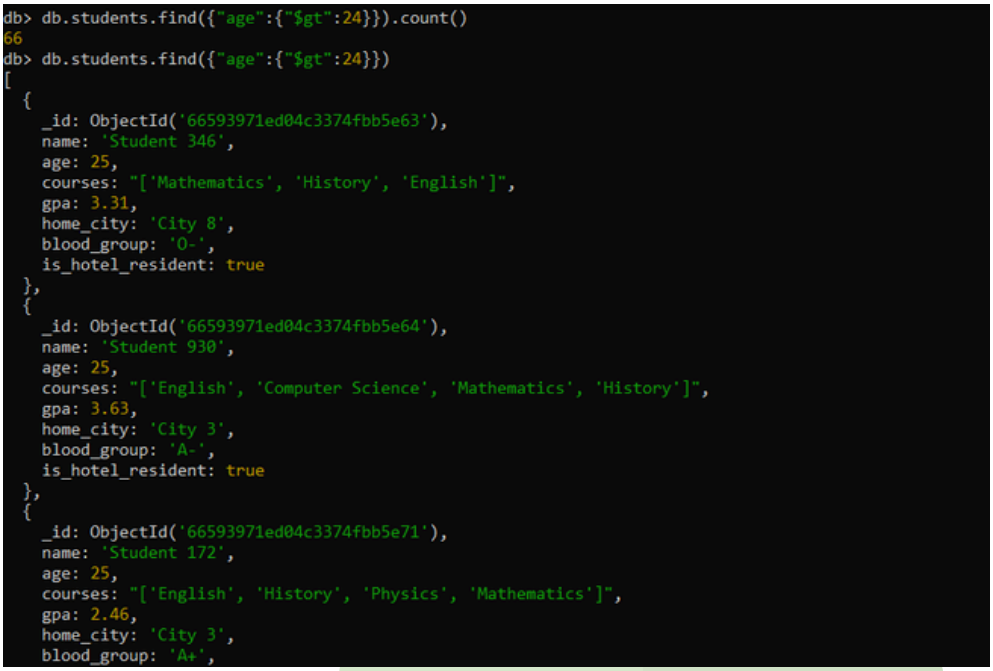



**Explanation:**

- Initially, we used the count() function to determine the total number of documents in the students collection, regardless of their field values.
- The count result showed that there were 500 documents in total.
- We then performed a query using the find() function with the **$ne** operator to exclude documents where the home_city field is equal to "City 1".
- This query filtered out documents where the home_city is "City 1" and returned the remaining documents.
- Finally, we used the count() function again on the result of the query to determine the number of documents that met the specified criteria, which was 466.

**3. $gt (Greater Than):**

Matches values greater than a specified value. Executing the query **db.students.find({ "age": { "$gt": 24 } }).count()** on the collection 'students' would return the following documents count:




The MongoDB query **db.students.find({ "age": { "$gt": 24 } }).count()** is counting the number of documents in the students collection where the value of the age field is greater than 24. Here's a breakdown of how this query works:

- db.students.find(): This command retrieves documents from the students collection.
- { "age": { "$gt": 24 } }: This is the query object passed to the find() function. It specifies the condition to filter documents based on the age field.
  - "age": Specifies the field to which the condition applies, which is age in this case.
  - { "$gt": 24 }: Defines the condition using the $gt operator, which stands for "greater than". It specifies that the value of the age field must be greater than 24 for a document to be included in the results.
- .count(): This method is applied to the result of the find() function. It calculates the total count of documents that match the specified condition.

**Explanation:**

- Out of the 500 documents in the collection, only 66 documents have an age greater than 24.
- The query filters the documents based on the specified condition (age greater than 24) and counts the number of matching documents.
- Therefore, the count result is 66, indicating that there are 66 documents where the age field is greater than 24 out of the total 500 documents in the collection.

**4.$gte: Greater than or equal:**

Matches values greater than or equal to a specified value.
For example:Find students with a GPA of 3.9 or higher.

```
db> db.students.find({"gpa":{"$gte":3.8}}).count()
51
```

Given that the result is 55 out of 500, it implies that there are 55 documents in the students collection where the GPA (gpa field) is greater than or equal to 3.8.
This indicates a significant subset of the total documents in the collection that meet the specified criteria, showcasing the power of MongoDB queries in filtering and retrieving specific data.

**5.$lt: Lesser than:**

Matches value less than or equal to a specified value.
Now consider students younger than 19.

```
db> db.students.find().count()
500
db> db.students.find({"age":{"$lt":19}}).count()
58
```

**Explanation:**

1. Total Documents: Initially, the collection has 500 documents.
2. Filtered Documents: The query db.students.find({ "age": { "$lt": 19 } }) applies a filter to select only those documents where the age field is less than 19.
3. Count of Filtered Documents: By appending .count(), we get the number of documents that meet the criteria, which is 58.

**6. $lte (Less Than or Equal):**

Matches value less than or equal to a specified value.

```
db> db.students.find({"gpa":{"$lte":2.1}}).count()
24
db> db.students.find().count()
500
db>
```

Example Scenario:
Given the result of 24 out of 500, this means:

- Out of a total of 500 documents in the student's collection, 24 documents have a GPA less than or equal to 2.1.
- This query effectively filters the data set to show only those documents where students have a lower GPA.

## Logical Operators in MongoDB:

MongoDB provides a range of logical operators that enable complex query conditions, allowing you to perform more precise data retrieval. These logical operators are often used to combine multiple conditions in a query. Here's a brief overview of the main logical operators and how to use them:

**1. $and:**

The $and operator joins query clauses with a logical AND, returning all documents that match the conditions of both clauses.

**Example:** Find all students with gpa greater than 3.5 gpa and are hotel residents.

```
db> db.students.find().count()
500
db> db.students.find({ $and: [ { "gpa": { "$gt": 3.5 } }, { "is_hotel_resident":true}] }).count()
60
db> db.students.find({ $and: [ { "gpa": { "$gt": 3.5 } }, { "is_hotel_resident":true}] })
[
  {
    _id: ObjectId('66593971ed04c3374fbb5e64'),
    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('66593971ed04c3374fbb5e77'),
    name: 'Student 468',
    age: 21,
    courses: "['Computer Science', 'Physics', 'Mathematics', 'History']",
    gpa: 3.97,
    blood_group: 'A-',
    is_hotel_resident: true
  },
  {
    _id: ObjectId('66593971ed04c3374fbb5e83'),
    name: 'Student 969',
```

**Explanation of the Command:**

- Filter Condition 1: { "GPA": { "$gte": 3.5 } }
  - This part of the query selects documents where the GPA field is greater than or equal to 3.5.
- Filter Condition 2: { "hotel_resident": true }
  - This part of the query selects documents where the hotel_resident field is true.
- Logical Operator: $and
  - The $and operator ensures that only documents that satisfy both conditions are selected.

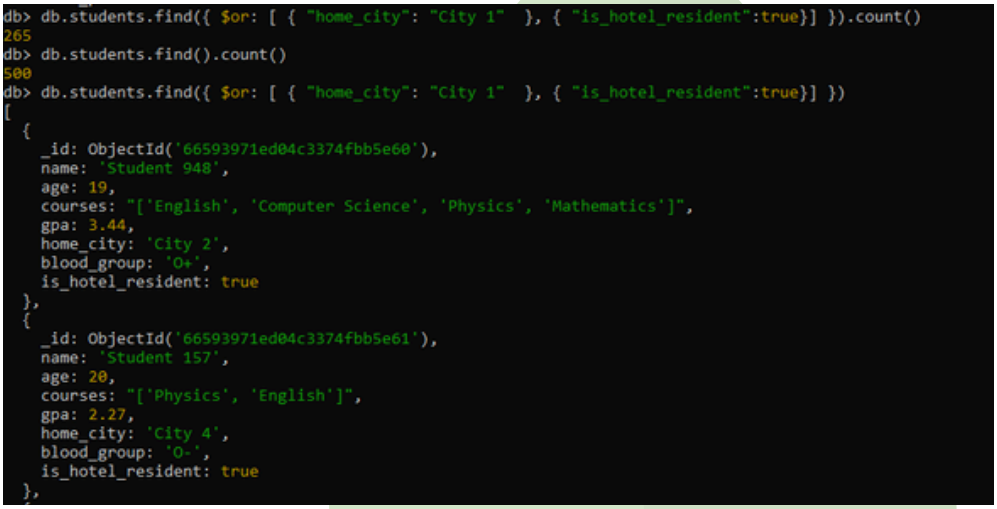
This exercise demonstrates how to use the $and logical operator to combine multiple query conditions, providing a powerful way to filter and retrieve specific subsets of data from a MongoDB collection.

Would you like to explore more about Mon

**2. $or:**

The $or operator joins query clauses with a logical OR, returning all documents that match the conditions of at least one clause.

**Example:** Find all students who are hotel residents or City 1.



```
db> db.students.find({ $or: [ { "home_city": "City 1"  }, { "is_hotel_resident":true}] }).count()
265
db> db.students.find().count()
500
db> db.students.find({ $or: [ { "home_city": "City 1"  }, { "is_hotel_resident":true}] })
[
  {
    _id: ObjectId('66593971ed04c3374fbb5e60'),
    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('66593971ed04c3374fbb5e61'),
    name: 'Student 157',
    age: 20,
    courses: "['Physics', 'English']",
    gpa: 2.27,
    home_city: 'City 4',
    blood_group: 'O-',
    is_hotel_resident: true
  },
```

**Explanation of the Command:**

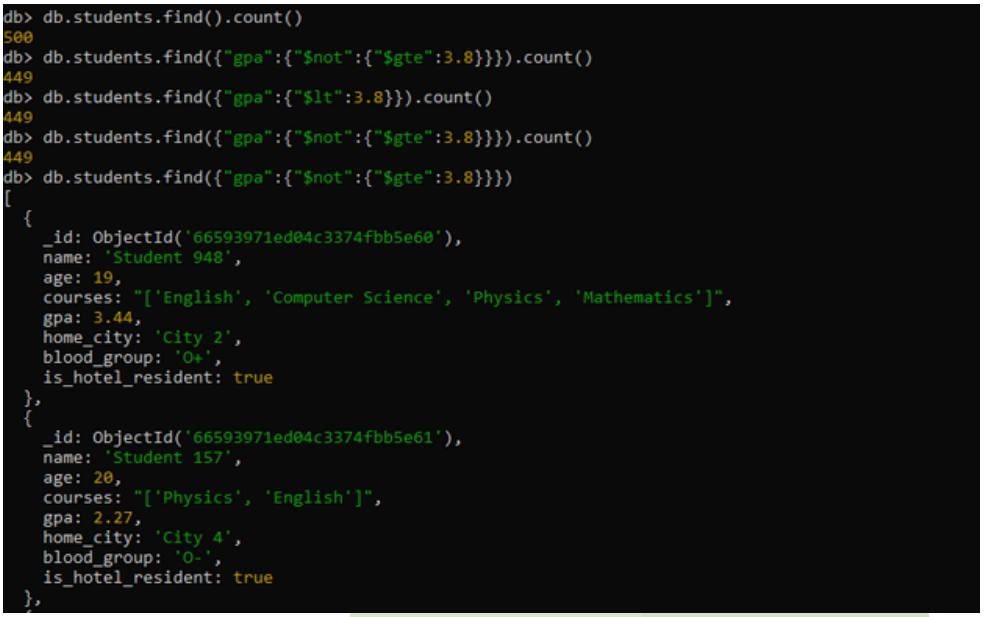
- Filter Condition 1: { "hotel_resident": true }
  - This part of the query selects documents where the hotel_resident field is true.
- Filter Condition 2: { "home_city": "City 1" }
  - This part of the query selects documents where the home_city field is "City 1".
- Logical Operator: $or
  - The $or operator ensures that documents matching either of the two conditions are selected.

This query illustrates how to use the **$or** logical operator to combine multiple query conditions, providing a way to retrieve data based on either one or more criteria being met.

**2. $not:**

The $not operator inverts the effect of a query expression and returns documents that do not match the condition.

**Example:** Find the count of all students students with a GPA less than 3.8. using $not.



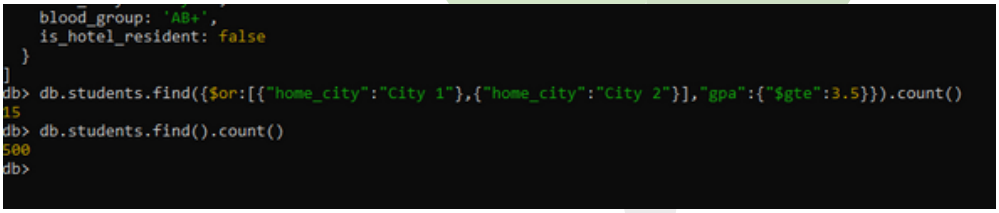

**Explanation of the Command:**

- Filter Condition: { "gpa": { "$not": { "$gte": 3.8 } } }
  - This condition selects documents where the GPA is not greater than or equal to 3.8, which means it will include students whose GPA is less than 3.8.

**Combining Logical Operators:**

Logical operators can be combined to form more complex queries.

Suppose we want to find students who are from either "City 1" or "City 2" and have a GPA greater than or equal to 3.5.




## UPDATE:

The updateOne function in MongoDB is used to update a single document in a collection that matches a specified filter. This function allows you to modify specific fields of a document or replace the entire document, depending on the update operations you specify.

**syntax:db.collection.updateOne(filter, update, options)**

- db.collection: The collection in which you want to update the document.
- filter: A query that matches the document to update.
- update: The modifications to apply. This can be an object with update operators or a replacement document.
- options (optional): Additional options for the update operation.

Let's say we have a students collection, and we want to update a specific student's information.

```
db> db.students.updateOne(
... {"name":"Alice"},
... {"$set":{"gpa":3.7,"home_city":"City A"}}
... )
{
  acknowledged: true,
  insertedId: null,
  matchedCount: 1,
  modifiedCount: 1,
  upsertedCount: 0'6668fa5928b5145253cdcdfe')
} }
db> db.students.find({"name":"Alice","gpa":3.7})
[
  {
    _id: ObjectId('6668f32828b5145253cdcdf6'),
    name: 'Alice',
    age: 20,
    courses: [ 'Mathematics', 'History' ],
    gpa: 3.7,
    home_city: 'City A',
    blood_group: 'A+',
    is_hotel_resident: true
  }
]
```

- When you run the above command, MongoDB will find the first document that matches the filter and update the specified fields.
- Return Value
- The updateOne function returns an object that contains information about the update operation. This includes:
- matchedCount: The number of documents that matched the filter.
- modifiedCount: The number of documents that were actually modified.
- upsertedId (if applicable): The ID of the upserted document (if an upsert was performed).

**Update Operators**

MongoDB provides several update operators that you can use in the update document:

- $set: Sets the value of a field in a document.
- $unset: Removes a field from a document.
- $inc: Increments the value of a field by a specified amount.
- $mul: Multiplies the value of a field by a specified amount.
- $rename: Renames a field.
- $addToSet: Adds elements to an array only if they do not already exist in the array.
- $pop: Removes the first or last element of an array.
- $pull: Removes all array elements that match a specified query.
- $push: Adds an element to an array.

```
db> db.students.find({"name":"Bob"})
[
  {
    _id: ObjectId('6668fa2628b5145253cdcdfa'),
    name: 'Bob',
    age: 24,
    courses: [ 'History', 'Mathematics' ],
    home_city: 'City 2',
    blood_group: 'B+',
    is_hotel_resident: false
  },
  {
    _id: ObjectId('6668fa5928b5145253cdcdfd'),
    name: 'Bob',
    age: 22,
    courses: [ 'History', 'Mathematics' ],
    home_city: 'City 2',
    blood_group: 'B+',
    is_hotel_resident: false
  }
]
db> db.students.updateOne( { "name": "Bob" }, { "$inc": { "age": 1 } } )
db>
  acknowledged: true,
  insertedId: null,
  matchedCount: 1,
  modifiedCount: 1,
  upsertedCount: 0
}
```

This increments Bob's age by 1.

```
db> db.students.find({ name : Bob , age :25})
[
  {
    _id: ObjectId('6668fa2628b5145253cdcdfa'),
    name: 'Bob',
    age: 25,
    courses: [ 'History', 'Mathematics' ],
    home_city: 'City 2',
    blood_group: 'B+',
    is_hotel_resident: false
  }
]
db>
```

By using updateOne, you can efficiently update specific fields of a document or replace a document in your MongoDB collections.

## DELETE:

The delete functions in MongoDB allow you to remove documents from a collection. There are two primary delete operations in MongoDB:

1. deleteOne: Deletes a single document that matches the given filter.
2. deleteMany: Deletes all documents that match the given filter.

**deleteOne:**

The deleteOne function removes the first document that matches the specified filter.

**Syntax: db.collection.deleteOne(filter, options)**

- db.collection: The collection from which you want to delete a document.
- filter: A document that specifies the criteria used to select the document to delete.
- options (optional): Additional options for the deletion.

Let's say we have a students collection, and we want to delete a student named "Alice".

```
db> db.students.find().count()
508
db> db.students.deleteOne({"name":"Alice"})
{ acknowledged: true, deletedCount: 1 }
db> db.students.find().count()
507
db>
```

The deleteOne function returns an object that contains information about the deletion. This includes:

- acknowledged: A boolean value indicating whether the operation was acknowledged by the server.
- deletedCount: The number of documents deleted (either 0 or 1).

**deleteMany**:

The deleteMany function removes all documents that match the specified filter.

**Syntax: db.collection.deleteMany(filter, options)**

- db.collection: The collection from which you want to delete documents.
- filter: A document that specifies the criteria used to select the documents to delete.
- options (optional): Additional options for the deletion.

Let's say we want to delete all students who are hotel residents.

```
db> db.students.deleteMany({"is_hotel_resident":true})
{ acknowledged: true, deletedCount: 251 }
db> db.students.find().count()
256
db>
```

**Explanation of the Example**

- db.students: Refers to the students collection in the current database.
- deleteMany: The function used to delete multiple documents.
- Filter: { "is_hotel_resident": true } - This query matches all documents where the is_hotel_resident field is true.

**Return Value**

The deleteMany function returns an object that contains information about the deletion. This includes:

- acknowledged: A boolean value indicating whether the operation was acknowledged by the server.
- deletedCount: The number of documents deleted.

By using deleteOne and deleteMany, you can efficiently manage and remove unwanted data from your MongoDB collections, ensuring your database remains clean and relevant.

# PROJECTION

Projection in MongoDB is used to specify or restrict the fields that should be returned from a query. Instead of returning the entire document, you can choose to return only certain fields, which can help reduce the amount of data transferred over the network and improve query performance.

**Syntax : db.collection.find(query, projection)**

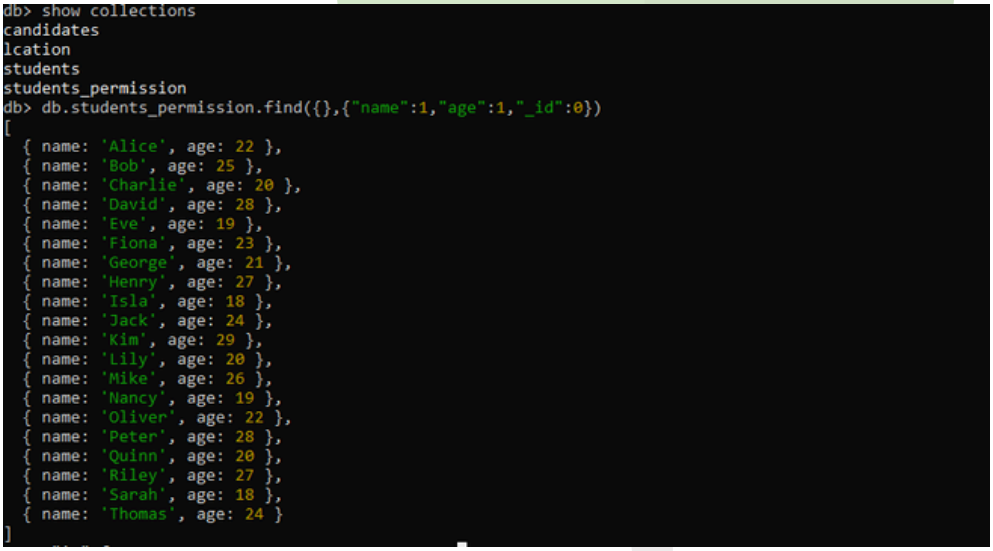
- db.collection: The collection from which you want to read the documents.
- query (optional): A document that specifies the criteria used to select documents. If omitted, all documents in the collection are returned.
- projection (optional): A document that specifies the fields to include or exclude from the returned documents.

Basic Projection

Include Fields

To include specific fields in the returned documents, you set those fields to 1 in the projection document.

Examples:Return only the name and age fields from documents in the students collection:




- Query: {} - This matches all documents.
- Projection: { "name": 1, "age": 1, "_id": 0 } - This includes the name and age fields and excludes the _id field.

**Explanation**

- name: 1: Include the name field.
- age: 1: Include the age field.
- _id: 0: Exclude the _id field (it's included by default if not specified).

Exclude Fields

- To exclude specific fields from the returned documents, you set those fields to 0 in the projection document.

By mastering CRUD operations and logical operators in MongoDB, you can perform comprehensive data management tasks and construct powerful, flexible queries. These capabilities enable efficient data manipulation, retrieval, and maintenance, crucial for developing robust, data-driven applications. MongoDB's rich query language and flexible schema design provide a strong foundation for handling diverse and dynamic datasets, making it an ideal choice for modern application development.