

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

Where:

In MongoDB, the **WHERE** clause you might be familiar with from SQL isn't used for filtering data. Instead, MongoDB utilizes a query document within the **find()** method to achieve similar functionality. This query document specifies the conditions for selecting documents from a collection.

Filtering works in MongoDB:

- **Collection:** You'll be working with a specific collection that stores your documents. Think of it as a table in a relational database.
- **find() Method:** This method is used to retrieve documents from a collection.
- **Query Document:** This document defines the filtering criteria. It uses field names and comparison operators to specify which documents to match.

Example: `db.students.find({blood_group:"A+"});`

```
mongosh mongodb://127.0.0.1
db> db.students.find({blood_group:"A+"});
[
  {
    _id: ObjectId('66670a750b6b0558dfefe188'),
    name: 'Student 268',
    age: 21,
    courses: "['Mathematics', 'History', 'Physics']",
    gpa: 3.98,
    blood_group: 'A+',
    is_hotel_resident: false
  },
  {
    _id: ObjectId('66670a750b6b0558dfefe18d'),
    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('66670a750b6b0558dfefe193'),
    name: 'Student 172',
    age: 25,
    courses: "['English', 'History', 'Physics', 'Mathematics']",
    gpa: 2.46,
    home_city: 'City 3',
    blood_group: 'A+',
    is_hotel_resident: false
  },
  {
    _id: ObjectId('66670a750b6b0558dfefe194'),
    name: 'Student 647',
    age: 21,
    courses: "['English', 'Physics']",
    gpa: 3.43,
    home_city: 'City 6',
    blood_group: 'A+',
    is_hotel_resident: true
  }
]
```

Here's a breakdown of what the statement does:

- ✓ **db.students**: This refers to the "students" collection within your MongoDB database. It's the collection containing the documents you want to query.
- ✓ **.find()**: This method initiates the search operation within the "students" collection. It tells MongoDB you want to find documents that match specific criteria.
- ✓ **{blood_group:"A+"}**: This curly brace block represents the query document. It defines the condition for selecting documents from the collection.
- ✓ **blood_group**: This specifies the field name in the documents that you want to filter on. In this case, you're looking at the "blood_group" field of each student document.
- ✓ **:"A+"**: This is a comparison operator. The colon : separates the field name from the value you're comparing against. Here, double quotes " indicate a string value, and "A+" is the specific blood group you're interested in.

Essentially, this statement instructs MongoDB to find all documents in the "students" collection where the "blood_group" field has the exact value "A+".

Comparison Operators in MongoDB Queries (Table)

Operator	Description	Example (find students with age > 20)
\$gt	Greater Than	db.students.find({ age: { \$gt: 20 } })
\$gte	Greater Than or Equal To	db.students.find({ age: { \$gte: 20 } })
\$lt	Less Than	db.students.find({ age: { \$lt: 20 } })
\$lte	Less Than or Equal To	db.students.find({ age: { \$lte: 20 } })
\$eq	Equal To	db.students.find({ age: { \$eq: 20 } })
\$ne	Not Equal To	db.students.find({ age: { \$ne: 20 } })

AND

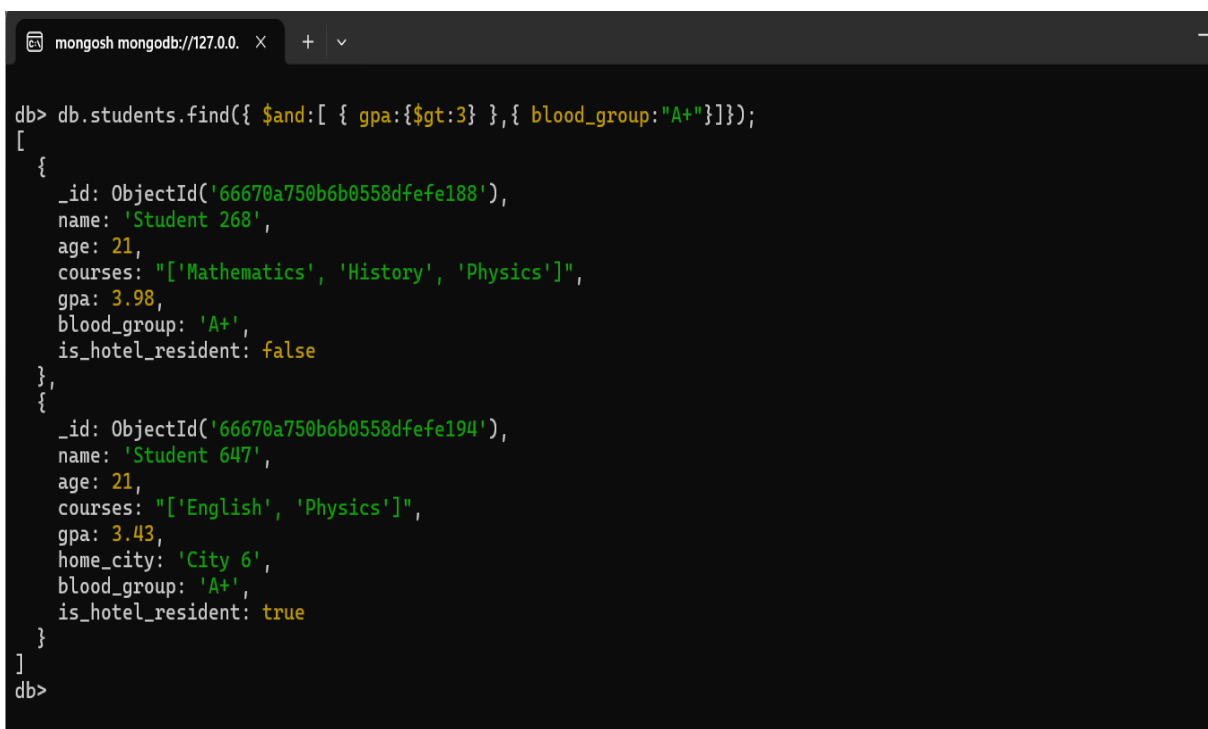
In MongoDB, we can use the **\$and** operator to filter a collection based on multiple conditions. The **\$and** operator ensures that all the specified conditions must be true for a document to be included in the results.

Here's how it works:

- **Define your conditions:** Each condition is a separate document within the \$and array. These documents use comparison operators like \$eq (equal), \$gt (greater than), \$lt (less than), etc., to specify the criteria for each field.
- **Structure the query:** You'll use the find method with a query document containing the \$and operator. Inside the \$and array, you'll define your individual conditions.

Example:

```
db.students.find({ $and:[ { gpa:{ $gt:3 } },{ blood_group:"A+"}]});
```



```
mongosh mongodb://127.0.0.1:27017 > db.students.find({ $and:[ { gpa:{ $gt:3 } },{ blood_group:"A+"}]});
[
  {
    _id: ObjectId('66670a750b6b0558dfefe188'),
    name: 'Student 268',
    age: 21,
    courses: ['Mathematics', 'History', 'Physics'],
    gpa: 3.98,
    blood_group: 'A+',
    is_hotel_resident: false
  },
  {
    _id: ObjectId('66670a750b6b0558dfefe194'),
    name: 'Student 647',
    age: 21,
    courses: ['English', 'Physics'],
    gpa: 3.43,
    home_city: 'City 6',
    blood_group: 'A+',
    is_hotel_resident: true
  }
]
```

Here's a breakdown of what the statement does:

- ✓ `db.students.find({...})`: This part specifies that you're using the find method on the "students" collection within your MongoDB database. The curly braces {} following it enclose the query document that defines the filtering criteria.
- ✓ `$and`: This operator is used to combine multiple conditions. In this case, you want documents that meet all of the specified conditions.
- ✓ Inside the `$and` array:
 - `{ gpa:{$gt:3} }`: This condition filters based on the "gpa" field. The `$gt` operator checks for values greater than 3. So, this part will only select documents where the student's GPA is strictly above 3.
 - `{ blood_group:"A+" }`: This condition filters based on the "blood_group" field. It uses the equality operator (`$eq`) implied by the colon (:), and specifies an exact match for the value "A+". Therefore, this part will only select documents where the student's blood group is exactly "A+".

This query searches the "students" collection for documents where:

- The student's GPA is greater than 3 .
- The student's blood group is specifically "A+".

some common use cases for the `$and` operator:

- Filtering based on multiple fields.
- Combining comparisons.
- Enforcing specific combinations.

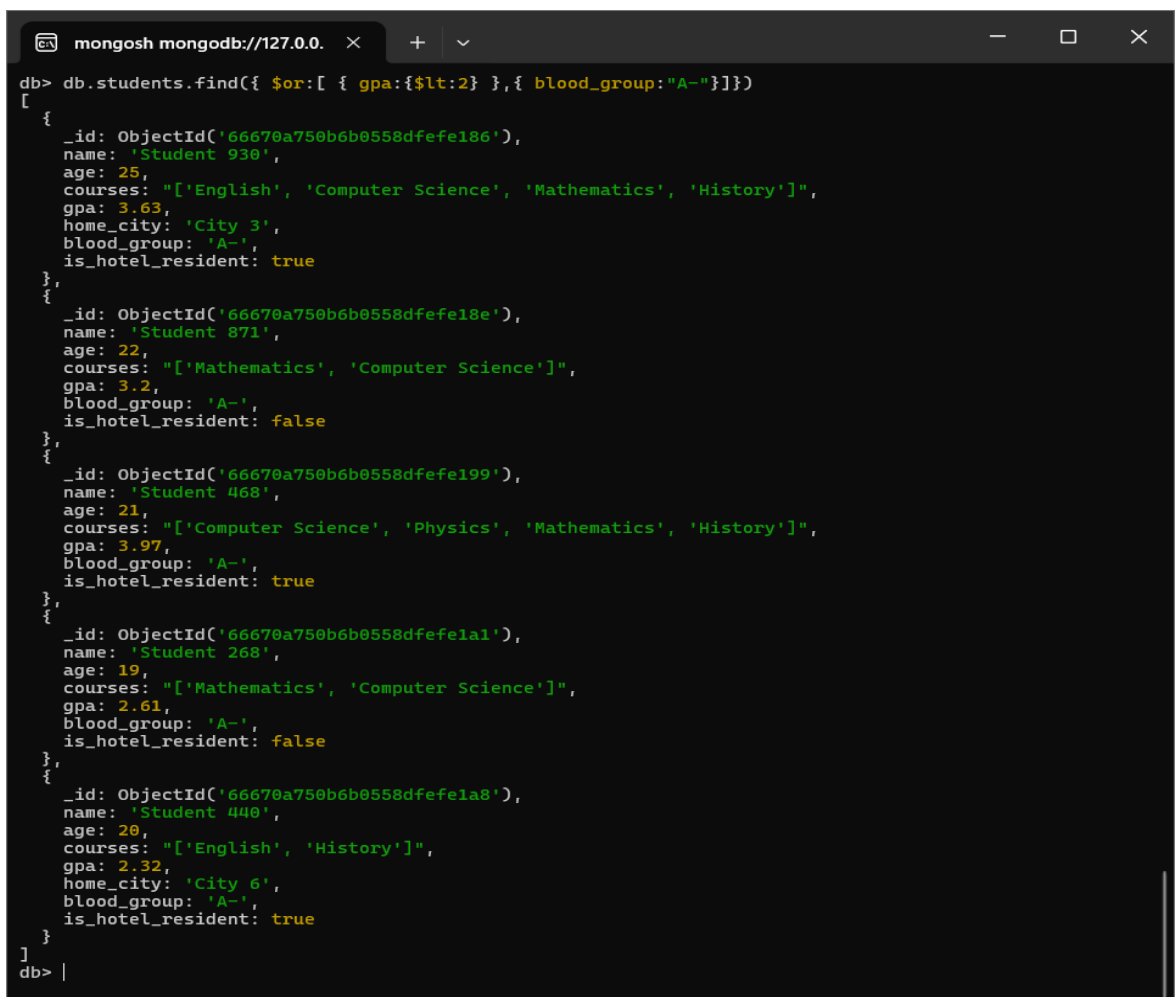
OR

In MongoDB, you can use the `$or` operator to filter a collection based on multiple conditions where **at least one** condition needs to be true for a document to be included in the results. This allows for a more flexible search where you can specify alternative criteria.

Here's how it works:

1. **Define your conditions:** Each condition is a separate document within the `$or` array. These documents use comparison operators like `$eq` (equal), `$gt` (greater than), `$lt` (less than), etc., to specify the criteria for each field.
2. **Structure the query:** You'll use the `find` method with a query document containing the `$or` operator. Inside the `$or` array, you'll define your individual conditions.

Example: `db.students.find({ $or: [{ gpa:{ $lt:2 } }, { blood_group:"A-" }] })`



```
mongosh mongodb://127.0.0.1
db> db.students.find({ $or: [ { gpa:{ $lt:2 } }, { blood_group:"A-" } ] })
[
  {
    _id: ObjectId('66670a750b6b0558dfefe186'),
    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('66670a750b6b0558dfefe18e'),
    name: 'Student 871',
    age: 22,
    courses: ['Mathematics', 'Computer Science'],
    gpa: 3.2,
    blood_group: 'A-',
    is_hotel_resident: false
  },
  {
    _id: ObjectId('66670a750b6b0558dfefe199'),
    name: 'Student 468',
    age: 21,
    courses: ['Computer Science', 'Physics', 'Mathematics', 'History'],
    gpa: 3.97,
    blood_group: 'A-',
    is_hotel_resident: true
  },
  {
    _id: ObjectId('66670a750b6b0558dfefe1a1'),
    name: 'Student 268',
    age: 19,
    courses: ['Mathematics', 'Computer Science'],
    gpa: 2.61,
    blood_group: 'A-',
    is_hotel_resident: false
  },
  {
    _id: ObjectId('66670a750b6b0558dfefe1a8'),
    name: 'Student 440',
    age: 20,
    courses: ['English', 'History'],
    gpa: 2.32,
    home_city: 'City 6',
    blood_group: 'A-',
    is_hotel_resident: true
  }
]
```

Here's a breakdown of what the statement does:

- ✓ `$or`: This operator is used to combine multiple conditions. In this case, you want documents that meet **at least one** of the specified conditions.
- ✓ Inside the `$or` array:
 - `{ gpa:{$lt:2} }`: This condition filters based on the "gpa" field. The `$lt` operator checks for values less than 2. So, this part will only select documents where the student's GPA is strictly below 2 (e.g., 1.9, 1.5, 0.8).
 - `{ blood_group:"A-" }`: This condition filters based on the "blood_group" field. It uses the equality operator (`$eq`) implied by the colon (:), and specifies an exact match for the value "A-".

This query searches the "students" collection for documents where:

- The student's GPA is less than 2 .
- **OR**
- The student's blood group is exactly "A-".

some common use cases for the `$or` operator:

- Finding documents based on multiple fields.
- Matching a range or specific values.
- Handling missing data.

1.b.Execute the Commands of MongoDB and operations in MongoDB : Insert, Query, Update, Delete and Projection. (Note: use any collection)

CURD

In MongoDB, CRUD operations are the backbone of interacting with your data stored in collections (similar to tables in relational databases). These operations allow you to manage and manipulate your documents (which act like rows) within those collections.

- **Create/Insert:** This involves inserting new documents into a collection. MongoDB provides methods like `insertOne` for adding a single document and `insertMany` for inserting multiple documents at once. If the collection doesn't already exist, MongoDB will create it for you during the insert operation.

```
mongosh mongodb://127.0.0.1:27026/
db> const studentData={ "name":"Ali", "age":21,
... "courses":["English","computer science"],
... "gpa":4,
... "home_city":"USA",
... "blood_group":"A-",
... "is_hostel_resident":false
... };
db> db.students.insertOne(studentData);
```

Inserting the Document:

- ✓ The `db.students.insertOne(studentData)` line interacts with MongoDB.
- ✓ It uses the `db` object, which likely represents a connection to your MongoDB database.
- ✓ The `insertOne` method is then called on the `students` collection, passing the `studentData` object as the document to insert.

```
{
  acknowledged: true,
  insertedId: ObjectId('6667627d45d4ed933046b79a')
}
db>
```

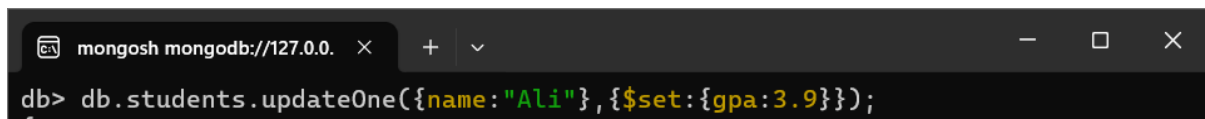
This is the response from MongoDB after successfully inserting the document.

- **acknowledged: true:** This confirms that the insert operation was acknowledged by the server. In some configurations, this might not always be true, so it's good practice to check for it.
- **insertedId: ObjectId('666814fcc0f6dc1a5346b799'):** This field provides the unique identifier assigned to the newly inserted document. MongoDB uses a special data type called ObjectId to automatically generate unique IDs for documents within a collection.

➤ Update:

Modifying existing documents within a collection is update. MongoDB offers methods like `updateOne` and `updateMany` to update documents based on certain conditions. You can update specific fields within a document or even replace the entire document.

UpdateOne:

A screenshot of a MongoDB shell terminal window. The title bar shows 'mongosh mongodb://127.0.0.1'. The command prompt 'db>' is followed by the command 'db.students.updateOne({name: "Ali"}, {\$set: {gpa: 3.9}});'. The command is highlighted in yellow.

- ✓ `db.students` specifies the collection named "students" where the update will be performed.
- ✓ The `updateOne` method takes two arguments. The first argument is a filter document that defines which document to update. Here, `{name: "Ali"}` targets the document where the name field **exactly matches** "Ali".
- ✓ The second argument is the update document that specifies how to modify the document that matches the filter.
- ✓ In this case, `{$set: {gpa: 3.9}}` uses the \$set operator to set the gpa field to 3.9 for the matching document.


```
{
  acknowledged: true,
  insertedId: null,
  matchedCount: 0,
  modifiedCount: 0,
  upsertedCount: 0
}
```

- **acknowledged: true:** Confirms successful acknowledgement of the update operation.
- **insertedId: null:** Since this is an update, no new document is inserted.
- **matchedCount: 0:** This is the key part. It shows that **no documents matched** the filter criteria (name exactly matching "Ali").
- **modifiedCount: 0:** As no documents matched, there were also zero documents modified (GPA wasn't updated for any student).
- **upsertedCount: 0:** The upsertedCount applies to scenarios where an update might insert a new document if no match is found. In this case, with updates being attempted, this value is 0 as no new documents were inserted.

UpdateMany:

```
db> db.students.updateMany({gpa:{$gt:3.0}},{$inc:{gpa:0.9}});
{
```

- ✓ **db.students** specifies the collection named "students" where the update will be performed.
- ✓ The **updateMany** method takes two arguments. The first argument is a filter document that defines which documents to update.
- ✓ In this case, **{gpa:{\$gt:3.0}}** targets documents where the gpa field is greater than (\$gt) 3.0.
- ✓ The second argument is the update document that specifies how to modify the documents that match the filter. Here, **{\$inc:{gpa:0.9}}** uses the **\$inc** operator to increment the **gpa** field by 0.9 for each matching document.

```
{
  acknowledged: true,
  insertedId: null,
  matchedCount: 21,
  modifiedCount: 21,
  upsertedCount: 0
}
```

This is the response from MongoDB after executing the update operation

- **acknowledged: true:** Similar to previous examples, this confirms successful acknowledgement of the update operation.
- **insertedId: null:** Since this is an update, no new document is inserted. Therefore, insertedId is null.
- **matchedCount: 21:** This indicates that **21 documents** matched the filter criteria (GPA greater than 3.0).
- **modifiedCount: 21:** This confirms that **all 21 matching documents** were successfully updated (their GPA was incremented by 0.9).
- **upsertedCount: 0:** The upsertedCount applies to scenarios where an update might insert a new document if no match is found. In this case, with documents being updated, this value is 0 as no new documents were inserted.

➤ Delete:

This operation removes documents entirely from a collection. The deleteOne method removes a single document matching a particular filter, while deleteMany removes multiple documents based on criteria.

DeleteOne:



```
mongosh mongodb://127.0.0.1
db> db.students.deleteOne({name:"Ali"});
```

- ✓ **db.students** specifies the collection named "students" where the deletion will occur.
- ✓ The **deleteOne** method takes a filter document as its argument.

- ✓ In this case, `{ name: "Ali" }` targets the document where the name field exactly matches "Ali".

```
{ acknowledged: true, deletedCount: 0 }  
db> |
```

This is the response from MongoDB after executing the delete operation.

- **acknowledged: true:** Similar to previous examples, this confirms successful acknowledgement of the delete operation.
- **deletedCount: 0:** This is the key part here. It indicates that zero documents were deleted.

DeleteMany:

```
db> db.students.deleteMany({ is_hostel_resident:false});
```

- ✓ `db.students` specifies the collection named "students" where the deletion will occur.
- ✓ The `deleteMany` method takes a filter document as its argument.
- ✓ In this case, `{ is_hostel_resident: false }` targets all documents where the `is_hostel_resident` field is set to false

```
{ acknowledged: true, deletedCount: 2 }  
db> |
```

This is the response from MongoDB after executing the delete operation.

- **acknowledged: true:** Similar to previous examples, this confirms successful acknowledgement of the delete operation.
- **deletedCount: 2:** This indicates that two documents matched the filter criteria (being non-hostel residents) and were deleted.

