Where Clause, AND, OR operations

CRUD & Projection

Load the Document

- Download the sample document (student.csv) from dataset
- Import the data to the collections
- You should be able to see the uploaded data in the mongo compass

Where

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

Syntax:

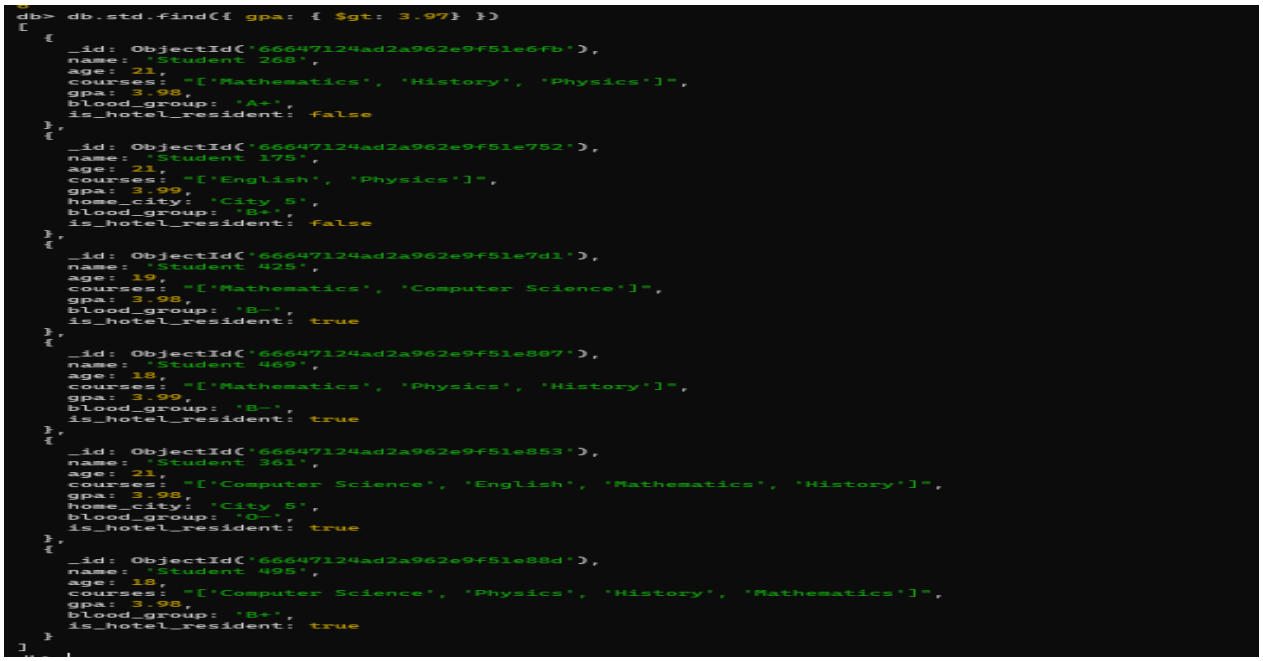
```
db.collectionName.find({ $where: <your_javascript_expression> })
```

Example:

Consider a collection named std with documents containing fields like name, age, and gpa etc …You want to find std from "City 3" and with a gpa greater than 3.97.

```
db> db.std.find({ home_city:"City 4"} ).count()
27
```

```
db> db.std.find({ gpa: { $gt: 3.97} }).count()
6
db>
```



AND Operator ($and)

The $and operator is used to combine multiple filtering conditions within a query document. All specified conditions must be true for a document to be included in the results.

Syntax:

```
db.collectionName.find({
$and: [
  { <condition1> },
  { <condition2> },
  ...
]
})
```

Example: Containing with the std collection,let's find student with a home_city and blood_group

```
db> db.std.find({ $and: [ {home_city:"City 5"},{blood_group:"A+"}]  } );
[
  {
    _id: ObjectId('66647124ad2a962e9f51e72c'),
    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('66647124ad2a962e9f51e84c'),
    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('66647124ad2a962e9f51e8be'),
    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> db.std.find({ $and: [ {home_city:"City 5"},{blood_group:"A+"}]  } ).count()
3
```

OR Operator ($or)

The $or operator is used to combine multiple filtering conditions where at least one condition must be true for a document to be included in the results.

Syntax:

```
db.collectionName.find({
  $or: [
    { <condition1> },
    { <condition2> },
    ...
  ]
```

```
})
```

Example:

Suppose you want to find all students who are hotel residents OR have a GPA less than 3

```
db> db.std.find({ $or:[{is_hotel_resident:true},{ gpa:{$lt:3.0}}]}).count()
374
db>
```

Choosing the Right Operator

- Use $and when all specified conditions must be met for a document to qualify.
- Use $or when at least one condition must be met for inclusion.
- Consider built-in comparison operators ($eq, $gt, $lt, etc.) for better performance over $where in most cases.

Additional Consideration

- MongoDB supports short-circuit evaluation for certain operators. For example, in $and, if the first condition evaluates to false, the remaining conditions won't be evaluated.
- When using $or with indexes, ensure each clause within the $or array can leverage an index for optimal performance.

CRUD and Projection

This explanation provides a comprehensive breakdown of CRUD (Create, Read, Update, Delete) operations and Projection in MongoDB, along with examples and additional information:

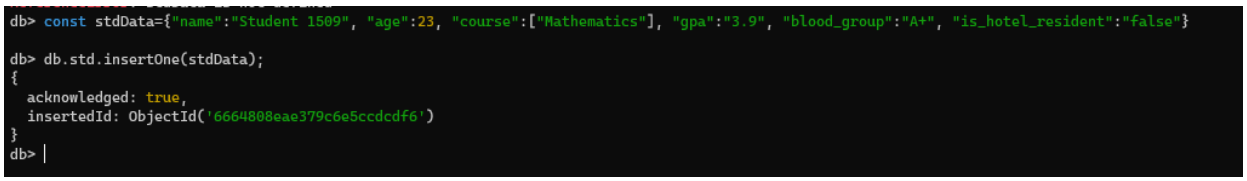
1. **Insert:** Adds new documents to a collection.

Syntax:

```
db.collectionName.insertOne({ documentData1: value1, documentData2: value2, ... });

db.collectionName.insertMany([{ documentData1: value1, documentData2: value2, ... }, ...]);
```

Example:



Information:

- insertOne adds a single document.

- insertMany adds an array of documents in one operation.

- Consider using validation rules within the insert methods to ensure data integrity.

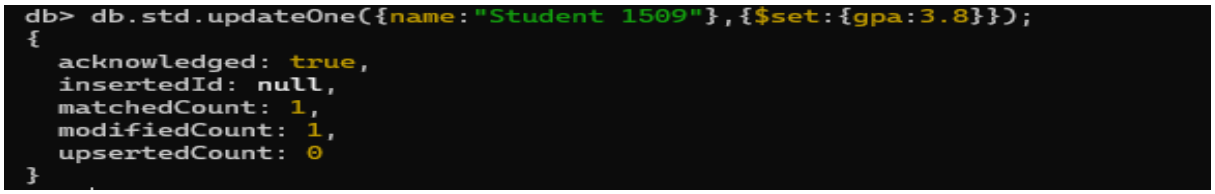
2. Update: Modifies existing documents based on criteria.

Syntax:

```
db.collectionName.updateOne({ filterObject }, { updateObject });

db.collectionName.updateMany({ filterObject }, { updateObject });
```

Example:



Information:

- updateOne modifies a single document matching the filter.

- updateMany modifies multiple documents matching the filter.

```
db> db.std.updateMany({gpa:{$lt:3.0}},{$inc:{gpa:0.5}});
```

- Consider using the $inc operator to increment or decrement numerical fields.

- Utilize update validators to ensure valid data after modifications.

3. Delete: Removes documents from a collection.

Syntax:

```
db.collectionName.deleteOne({ filterObject });

db.collectionName.deleteMany({ filterObject });
```

Example:

Delete a student by name

```
db> db.std.deleteOne({name:"John Doe"});
```

Information:

- deleteOne removes a single document matching the filter.

- deleteMany removes multiple documents matching the filter.

```
db> db.std.deleteMany({is_hotel_resident:false});
```

- Be cautious with delete operations, as they're permanent. Consider archiving documents instead of deletion for potential retrieval needs.

4. Projection: Specifies which fields to include or exclude when retrieving documents.

Syntax:

```
db.collectionName.find({ filterObject }, { projectionObject });
```

Example:

```
// Find all products, excluding the `_id` field:

db.products.find({}, { _id: 0 });
```

Information:

- Projection helps limit data transfer and improve performance by selectively returning needed fields.

- Use 1 to include a field, 0 to exclude it.

Additional Considerations:

- These are basic examples. You can modify queries and operations based on your specific needs.

- For complex scenarios, consider using aggregation pipelines for data manipulation and transformation.

- Explore features like indexes, sorting, and limiting for efficient data retrieval.