Where, AND, OR & CRUD

Where

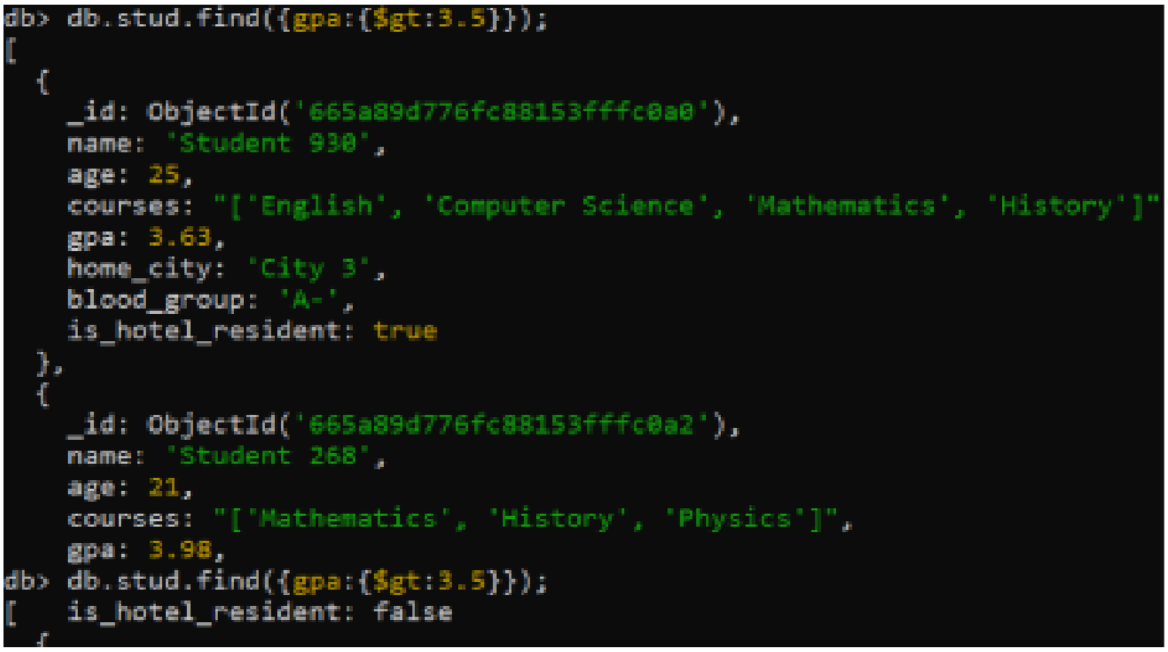
Use the where operator to pass either a string containing a JavaScript expression or a full JavaScript function to the query system. The where provides greater flexibility, but requires that the database processes the JavaScript expression or function for each document in the collection.

In the world of database management, the WHERE clause plays a crucial role in filtering and retrieving specific data from a collection or table. The WHERE clause allows you to apply a condition or set of conditions to narrow down the results, providing you with a subset of the data that meets your criteria.
Querying Documents

The WHERE clause in MongoDB allows you to filter and select specific documents from a collection based on one or more criteria.

Comparison Operators: MongoDB supports various comparison operators like $eq, $gt, $lt, $gte, and $lte to perform advanced queries.

Logical Operators: The $and, $or, and $not operators can be used to combine multiple conditions for more complex queries.



And

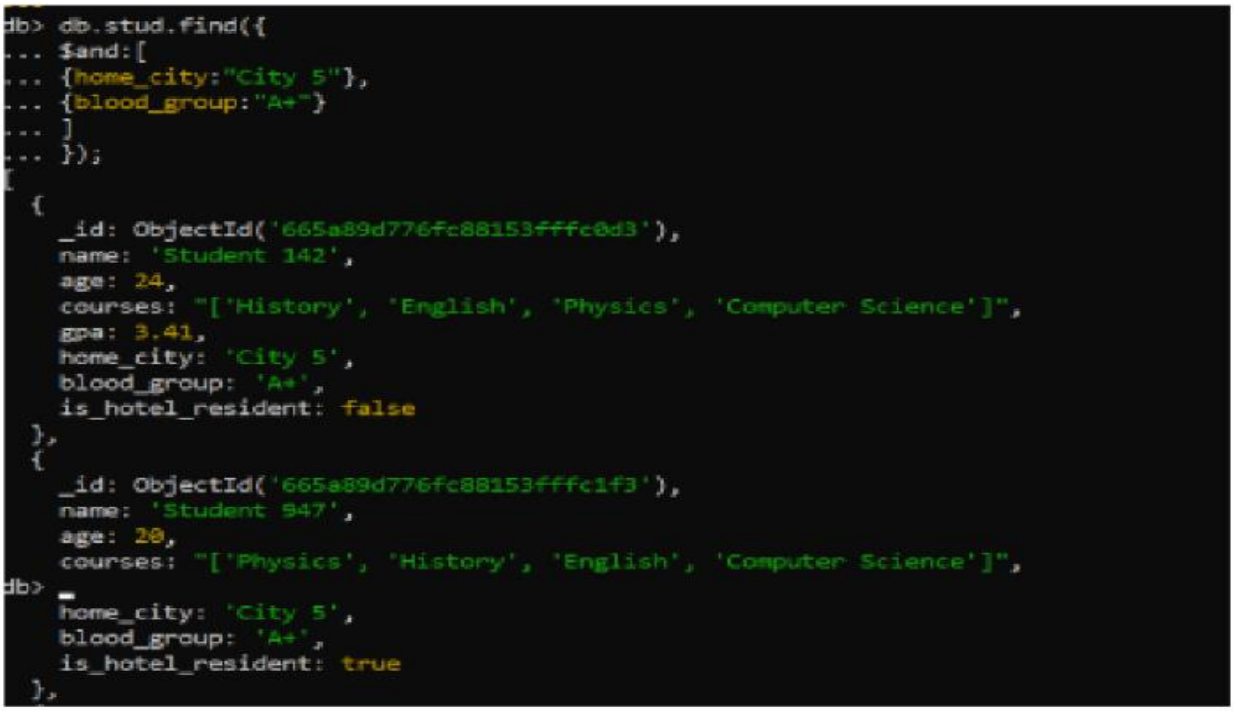
and performs a logical AND operation on an array of two or more expressions (e.g. <expression1> , <expression2> , etc.) and selects the documents that satisfy all the expressions in the array. The and operator uses short-circuit evaluation.

Examples:

db.inventory.find({ price: 1.99, qty: { $lt: 20 } , sale: true })

db.inventory.update({ $and: [{ price: { $ne: 1.99 } }, { price: { $exists: true } }] }, { $set: { qty: 15 } })

db.inventory.update({ price: { $ne: 1.99, $exists: true } } , { $set: { qty: 15 } })



OR

>.The or operator performs a logical OR operation on an array of *one or more* <expressions> and selects the documents that satisfy *at least* one of the <expressions

You can use or for deployments hosted in the following environments:

- MongoDB Atlas: The fully managed service for MongoDB deployments in the cloud
- MongoDB Enterprise: The subscription-based, self-managed version of MongoDB
- MongoDB Community: The source-available, free-to-use, and self-managed version of MongoDB

Syntax

- The or operator has the following syntax:

```
{ or: [ { <expression1> }, { <expression2> }, ... , { <expressionN> } ] }
```

- Consider the following example:

```
db.inventory.find( { or: [ { quantity: { $lt: 20 } }, { price: 10 } ] } )
```

Crud:

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

Insert

insert() operation successfully inserts a document, the operation adds an entry on the oplog (operations log). If the operation fails, the operation does not add an entry on the oplog.

EXAMPLE:

db.products.insert({ item: "card", qty: 15 })

Insert Multiple Documents

The following example performs a bulk insert of three documents by passing an array of documents to the insert() method. By default, MongoDB performs

an *ordered* insert. With *ordered* inserts, if an error occurs during an insert of one of the documents, MongoDB returns on error without processing the remaining documents in the array.

The documents in the array do not need to have the same fields. For instance, the first document in the array has an `_id` field and a `type` field. Because the second and third documents do not contain an `_id` field, `mongod` will create the `_id` field for the second and third documents during the insert:

```
db.products.insert(
   [
     { _id: 11, item: "pencil", qty: 50, type: "no.2" },
     { item: "pen", qty: 20 },
     { item: "eraser", qty: 25 }
   ]
)
```

EXAMPLE

```
{ "_id" : 11, "item" : "pencil", "qty" : 50, "type" : "no.2" }
{ "_id" : ObjectId("51e0373c6f35bd826f47e9a0"), "item" : "pen", "qty" : 20 }
{ "_id" : ObjectId("51e0373c6f35bd826f47e9a1"), "item" : "eraser", "qty" : 25 }
```

```
b> const studentData={
.. "name": "Jam",
.. "age" :12,
.. "courses" :["CS","Maths","Kannada"],
.. "gpa":3.2,
.. "home_city":"City 4",
.. "blood_group": "AB+",
.. "is_hotel_resident" : true
.. }
```

UPDATE:

The update() method in MongoDB updates a document or multiple documents in the collection. When the document is updated the _id field remains unchanged. This method can be used for a single updating of documents as well as multiple documents.

Parameters:

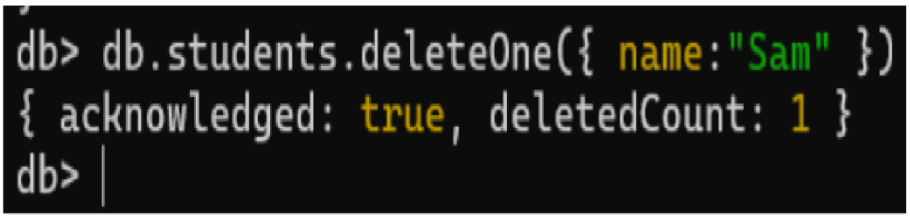
- **SELECTION_CRITERIA**: The selection filter for the update.
- **$set:** Keyword to update the following specify document value.
- **UPDATED_DATA:** New value that will replace the existing document.

```
db> db.students.updateOne( { name:"Sam"} , {$set:{
gpa:3} } )
{
  acknowledged: true,
  insertedId: null,
  matchedCount: 1,
  modifiedCount: 1,
  upsertedCount: 0
}
db>
```

Here the data of “sam” will be updated to the database.

Delete

To delete a record, or document as it is called in MongoDB, we use the deleteOne() method. The first parameter of the deleteOne() method is a query object defining which document to delete.



Here the data "Sam" will be deleted from the present database.

Projection

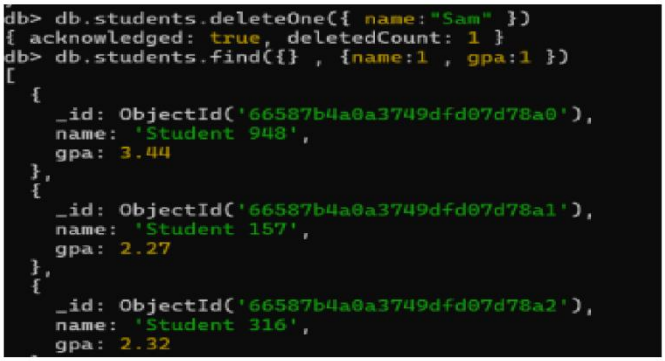
MongoDB projection is a powerful tool that can be used to extract only the fields you need from a document—not all fields. It enables you to: Project concise and transparent data. Filter the data set without impacting the overall database performance.

Basic syntax of projection:

```
db.collection_name.find({},{<field> : <value>})
```

To set a projection:

1. In the Query Bar, click Options.
2. Enter the projection document into the Project field. To include fields: Specify the field name and set to 1 in the project document. ...
3. Click Find to run the query and view the updated results. Note.



Operators in MongoDB Projection

Using the MongoDB projection method to retrieve specific data from a document will positively impact database performance because it reduces the workload of the find query, minimizing resource usage.

To reduce the workload, MongoDB offers the below operators that can be used within a projection query:

- $
- $elemMatch
- $slice
- $meta