**MongoDB Queries**

Mongodb: MongoDB is an open source NoSQL database management program. NoSQL is used as an alternative to traditional relational databases. NoSQL databases are quite useful for working with large sets of distributed data. MongoDB is a tool that can manage document-oriented information, store or retrieve information.

**Mongod:**

(Short for Mongo Daemon) is a background process used by MongoDB server to get things done. This process is responsible for managing the whole MongoDB server tasks such as accepting requests, responding to users, managing memory requirements of MongoDB server operations and other things essential for MongoDB Server to run.

**Mongo:**

On the other hand, is an interactive JavaScript shell interface to MongoDB, which provides a powerful interface for system administrators as well as a way for developers to test queries and operations directly with the database.

It acts like a mongodb client which can be used as a shell to get access to MongoDB database server run by mongod instances.

**Mongo Shell Commands**

* help – show help
* help admin – administrative help
* help connect – connecting to a db help
* help keys – key shortcuts
* help misc – misc things to know
* help mr – mapreduce
* show dbs – show database names
* show collections – show collections in current database
* show users – show users in current database
* show profile – show most recent system.profile entries with time >= 1ms
* show logs – show the accessible logger names
* show log [name] – prints out the last segment of log in memory, ‘global’ is default
* use <db\_name> – set current database

**A] Find and find One (specific values)**

**1) Create Database:**

There is no “create” command in the MongoDB Shell. In order to create a database, you will first need to switch the context to a non-existing database using the use command.

**use myDB**

**show dbs**

**2) Create Collection:** MongoDB creates a collection implicitly when the collection is first referenced in a command.

**3) Create Document:** Collections can be created just like databases, by writing a document to them. They can also be created using the [createCollection](https://docs.mongodb.com/manual/reference/method/db.createCollection/?_ga=2.256400201.337044667.1667293622-350038107.1664378983&_gac=1.20186698.1664596550.CjwKCAjwp9qZBhBkEiwAsYFsb01VsW7rnrmqLHRDLFDZ2ElFv5p0ykHJwVAiSu9oDD0EN-ysdekbChoCMnMQAvD_BwE) command.

**db.user.insert({name: "Ashwini", age: 30})**

The “user” in the command refers to the collection that the document was being inserted in.

4) To find out which database is the current one, enter the db command:

**Db**

**MongoDB CRUD operations:**

## 1) Create Operations:

The create or insert operations are used to insert or add new documents in the collection. If a collection does not exist, then it will create a new collection in the database.

| Method | Description |
| --- | --- |
| **db.collection.insertOne()** | It is used to insert a single document in the collection. |
| **db.collection.insertMany()** | It is used to insert multiple documents in the collection |
|  |  |

**2) Read Operations:**

The Read operations are used to retrieve documents from the collection, or in other words, read operations are used to query a collection for a document. You can perform read operation using the following method provided by the MongoDB

| **db.collection.find()** | It is used to retrieve documents from the collection. |
| --- | --- |

**3) Update Operation:** The update operations are used to update or modify the existing document in the collection. You can perform update operations using the following methods provided by the MongoDB

| **Method** | **Description** |
| --- | --- |
| db.collection.updateOne() | It is used to update a single document in the collection that satisfy the given criteria. |
| db.collection.updateMany() | It is used to update multiple documents in the collection that satisfy the given criteria. |
| db.collection.replaceOne() | It is used to replace single document in the collection that satisfy the given criteria. |

**4) Delete Operation:** The delete operation are used to delete or remove the documents from a collection. You can perform delete operations using the following methods provided by the MongoDB

| **Method** | **Description** |
| --- | --- |
| db.collection.deleteOne() | It is used to delete a single document from the collection that satisfy the given criteria. |
| db.collection.deleteMany() | It is used to delete multiple documents from the collection that satisfy the given criteria. |

Eg: db.user.deleteOne()

[**$or**](https://www.mongodb.com/docs/manual/reference/operator/query/or/#mongodb-query-op.-or) **Queries:**

The [$or](https://www.mongodb.com/docs/manual/reference/operator/query/or/#mongodb-query-op.-or) operator performs a logical OR operation on an array of *one or more* <expressions> and selects the documents that satisfy *at least* one of the <expressions>. The [$or](https://www.mongodb.com/docs/manual/reference/operator/query/or/#mongodb-query-op.-or) has the following syntax:

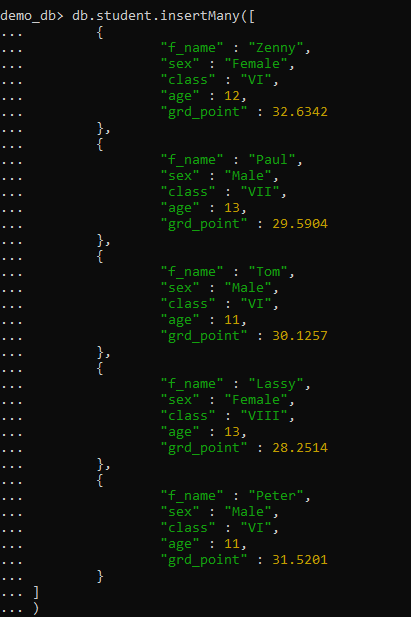
{ $or: [ { <expression1> }, { <expression2> }, ... , { <expressionN> } ] }

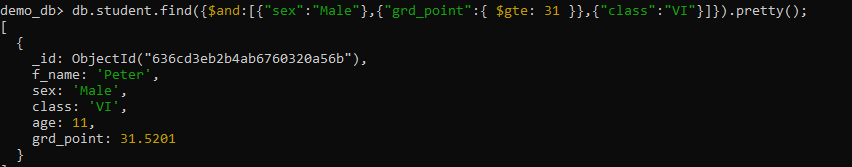
**B] Query criteria**

**Logical Operator - $and**

The MongoDB $and operator performs a logical AND operation on an array of two or more expressions and retrieves the documents which satisfy all the expressions in the array. The $and operator uses short-circuit evaluation. If the first expression (e.g. <expression1>) evaluates to false, MongoDB will not evaluate the remaining expressions.

{ $and: [ { <exp1> }, { <exp2> } , ... , { <expN> } ] }



Above query to select all documents from the collection "student" which satisfying the condition

1. *sex* of student is Female and

2. *class* of the student is VI and

3. *grd\_point* of the student is greater than equal to 31

**Logical Operator - $not**

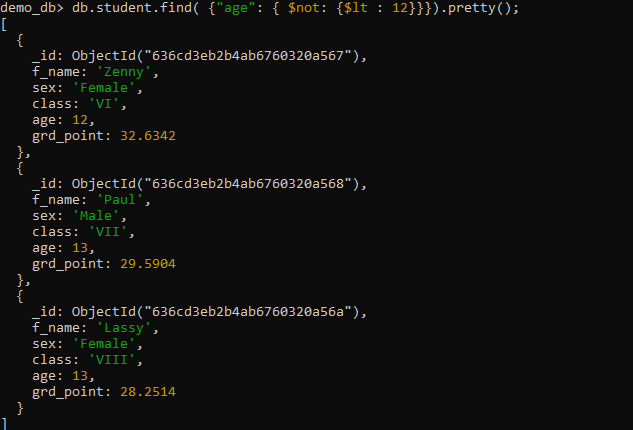
The MongoDB $not operator performs a logical NOT operation on the given expression and fetches selected documents that do not match the expression and the document that do not contain the field as well, specified in the expression.

{ field: { $not: { <expression> } } }

If we want to select all documents from the collection "student" which satisfying the condition -

*age* of the student is at least 12 the following mongodb command can be used :

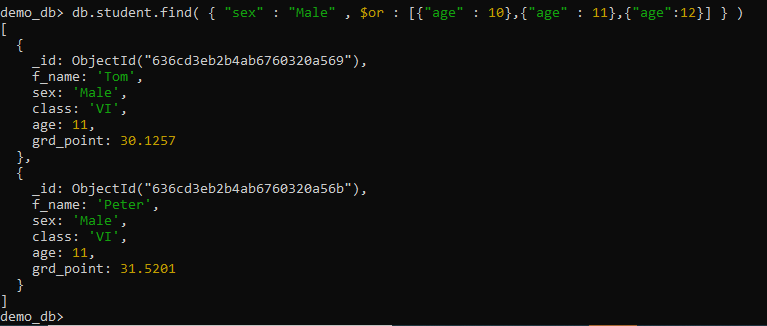
>db.student.find( {"age": { $not: {$lt : 12}}}).pretty();



**$or operator**

The $or operator is used to search multiple expression in a single query with only one matching criterion to be needed in a document. More than one keys and values can be used with the $or operator.

If we want to fetch documents from the collection "student" which containing the value of "age " either 10 or 11 or 13, the following mongodb command can be used :



**C] Type-specific queries**

Sometimes, you need to deal with highly unstructured data where **data types are unpredictable**. In this case, you need to use the $type operator.

The $type is an **element query operator**that allows you to select documents where the value of a field is an instance of a specified BSON type.

The $type operator has the following syntax:

{ field: { $type: <BSON type> } }

Example:

db.products.insertMany([

{ "\_id" : 1, "name" : "xPhone", "price" : "799", "releaseDate" : ISODate("2011-05-14T00:00:00Z"), "spec" : { "ram" : 4, "screen" : 6.5, "cpu" : 2.66 }, "color" : [ "white", "black" ], "storage" : [ 64, 128, 256 ] },

{ "\_id" : 2, "name" : "xTablet", "price" : NumberInt(899), "releaseDate" : ISODate("2011-09-01T00:00:00Z"), "spec" : { "ram" : 16, "screen" : 9.5, "cpu" : 3.66 }, "color" : [ "white", "black", "purple" ], "storage" : [ 128, 256, 512 ] },

{ "\_id" : 3, "name" : "SmartTablet", "price" : NumberLong(899), "releaseDate" : ISODate("2015-01-14T00:00:00Z"), "spec" : { "ram" : 12, "screen" : 9.7, "cpu" : 3.66 }, "color" : [ "blue" ], "storage" : [ 16, 64, 128 ] },

{ "\_id" : 4, "name" : "SmartPad", "price" : [599, 699, 799], "releaseDate" : ISODate("2020-05-14T00:00:00Z"), "spec" : { "ram" : 8, "screen" : 9.7, "cpu" : 1.66 }, "color" : [ "white", "orange", "gold", "gray" ], "storage" : [ 128, 256, 1024 ] },

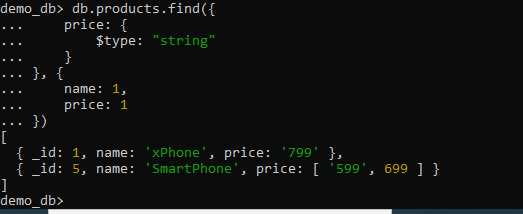
{ "\_id" : 5, "name" : "SmartPhone", "price" : ["599",699], "releaseDate" : ISODate("2022-09-14T00:00:00Z"), "spec" : { "ram" : 4, "screen" : 9.7, "cpu" : 1.66 }, "color" : [ "white", "orange", "gold", "gray" ], "storage" : [ 128, 256 ] },

{ "\_id" : 6, "name" : "xWidget", "spec" : { "ram" : 64, "screen" : 9.7, "cpu" : 3.66 }, "color" : [ "black" ], "storage" : [ 1024 ] }

])

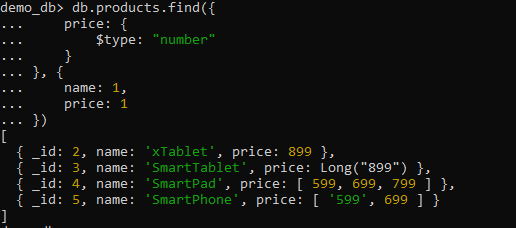
Using the $type operator example

The following example uses the $type operator to query documents from the products collection where the price field is the string type or is an array containing an element that is a string type.



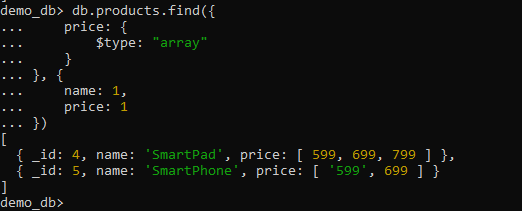
2) Using the $type operator with the number alias example

The following example uses the $type operator with the number alias to select documents where the value of the price field is the BSON type int, long, or double or is an array that contains a number:



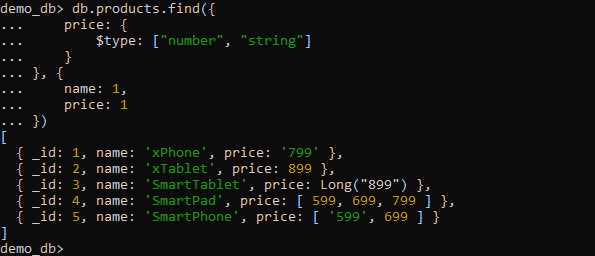
3) Using the $type operator to query documents with array type example

The following query use the $type operator to select the documents in which the price field is an array:



4) Using the $type operator to query documents with multiple types

The following query uses the $type operator to select documents where the price field is either number or string or an array that has an element is number or string:



**D] Cursors (Limit, skip, and Sort)**

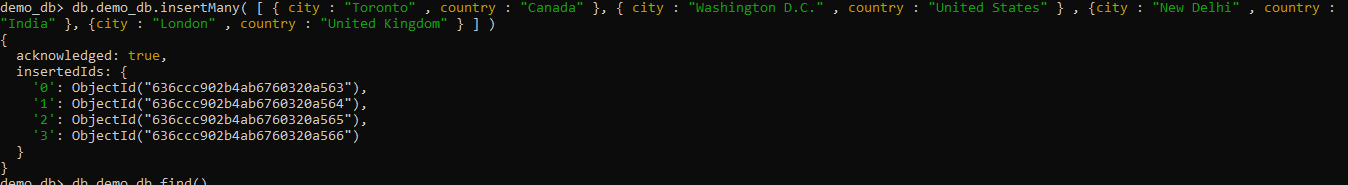
The limit () Method

In MongoDB, the ‘limit ()’ method is used to limit the records or documents present inside a collection. It accepts only one argument which is of number type. Depending on the value of the number, we can limit the number of documents to be displayed. This argument is an optional field inside the ‘limit ()’ method, and when not specified then by default, it will display all the documents from the collection.

The following is the basic syntax for ‘limit ()’ method in MongoDB.

>db.COLLECTION\_NAME.find().limit(NUMBER)

Example  
The following is an example on using the ‘limit ()’ statement. Let’s insert 4 documents into a collection known as ‘demo\_db’.





* Here, we are using ‘demo\_db’ as the collection name of MongoDB.
* Next, we are using the ‘save ()’ method to insert four records together as the bulk write operation in the MongoDB.
* Next, we are using the ‘find ()’ to display all the saved records.
* Next, we are using the ‘limit (2)’ method with the argument value as 2 which will limit the display of records to two as shown below.

The skip () Method

In MongoDB, the ‘skip ()’ method is used to skip the number of documents. Like the ‘limit ()’ method, it accepts only one argument which is of number type. Depending on the value of the number, we can skip the number of documents to be displayed. This argument is an optional field inside the ‘skip ()’ method, and when not specified, then it will display all documents from the collection as the default value in ‘skip ()’ method is 0.

The following is the basic syntax for ‘skip ()’ method in the MongoDB.

>db.COLLECTION\_NAME.find().limit(NUMBER).skip(NUMBER)

The following is an example on using the ‘skip ()’ statement.

Refer demo\_db collection to perform skip operation



We are using the ‘limit (2)’ along with ‘skip (1) ‘method to make sure only two documents are displayed after skipping one document as shown below.

The sort () Method

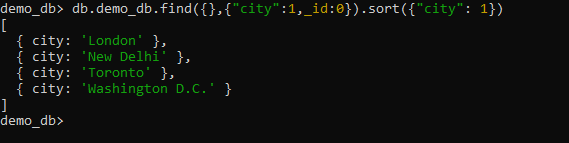
In MongoDB, the ‘sort ()’ method is used to sort the documents inside a collection. It accepts a document which contains a list of fields along with their sorting order. We use 1 and -1 in order to specify the sorting order where 1 corresponds to the ascending order and -1 corresponds to the descending order. Also, it should be noted that if we do not specify any sorting preference, then ‘sort ()’ method will display the documents in the ascending order.

The following is the basic syntax for ‘sort ()’ method in the MongoDB.

db.COLLECTION\_NAME.find().sort({KEY:1})

The following is an example on using the ‘sort ()’ statement.

Refer demo\_db collection to perform skip operation



We are using the ‘sort ({“city”: -1})’ method to display the cities in the descending order as the argument value is -1.

Lastly, we are using the ‘sort ({“city”: 1})’ method to display the cities in the ascending order as the argument value is 1.