Getting Started with MongoDB

Import Example Dataset

Overview

The examples in this guide use the `restaurants` collection in the `test` database. The following is a sample document in the `restaurants` collection:

```
{
  "address": {
     "building": "1007",
     "coord": [ -73.856077, 40.848447 ],
     "street": "Morris Park Ave",
     "zipcode": "10462"
  },
  "borough": "Bronx",
  "cuisine": "Bakery",
  "grades": [
     { "date": { "$date": 1393804800000 }, "grade": "A", "score": 2 },
     { "date": { "$date": 1378857600000 }, "grade": "A", "score": 6 },
     { "date": { "$date": 1358985600000 }, "grade": "A", "score": 10 }
,
     { "date": { "$date": 1322006400000 }, "grade": "A", "score": 9 },
     { "date": { "$date": 1299715200000 }, "grade": "B", "score": 14 }
  ],
  "name": "Morris Park Bake Shop",
  "restaurant_id": "30075445"
}
```

Use the following procedure to populate the `restaurants` collection.

Prerequisites

You must have a running `mongod` instance in order to import data into the database.

Procedure

1

Retrieve the `restaurants` data.
Retrieve the dataset from https://raw.githubusercontent.com/mongodb/docs-assets/primer-dataset/dataset.json and save to a file named `primer-dataset.json`.

2

Import data into the collection.
In the system shell or command prompt, use `mongoimport` to insert the documents into the`restaurants` collection in the `test` database. If the collection already exists in the `test` database, the operation will **drop** the `restaurants` collection first.

```
mongoimport --db test --collection restaurants --drop --file <pathTo>/prim
er-dataset.json
```

The `mongoimport` connects to a `mongod` instance running on localhost on port number `27017`.

To import data into a `mongod` instance running on a different host or port, specify the hostname or port by including the `--host` and the `--port` options in your `mongoimport` command.

MongoDB Shell (mongo)

The `mongo` shell is an interactive JavaScript interface to MongoDB and is a component of the MongoDB package. You can use the `mongo` shell to query and update data as well as perform administrative operations.

Start `mongo`

Once you have installed and have started MongoDB, connect the `mongo` shell to your running MongoDB instance. Ensure that MongoDB is running before attempting to launch the `mongo` shell.

On the same system where the MongoDB is running, open a terminal window (or a command prompt for Windows) and run the `mongo` shell with the following command:

```
mongo
```

On Windows systems, add `.exe` as follows:

```
mongo.exe
```

You may need to specify the path as appropriate.

When you run `mongo` without any arguments, the `mongo` shell will attempt to connect to the MongoDB instance running on the `localhost` interface on port `27017`. To specify a different host or port number, as well as other options, see mongo Shell Reference Page.

Help in `mongo` Shell

Type `help` in the `mongo` shell for a list of available commands and their descriptions:

```
help
```

The `mongo` shell also provides `<tab>` key completion as well as keyboard shortcuts similar to those found in the bash shell or in Emacs. For example, you can use the `<up-arrow>` and the `<down-arrow>` to retrieve operations from its history.

Insert Data with the `mongo` Shell

Overview

You can use the `insert()` method to add documents to a collection in MongoDB. If you attempt to add documents to a collection that does not exist, MongoDB will create the collection for you.

Prerequisites

In the `mongo` shell connected to a running `mongod` instance, switch to the `test` database.

```
use test
```

Insert a Document

Insert a document into a collection named `restaurants`. The operation will create the collection if the collection does not currently exist.

```
db.restaurants.insert(
   {
      "address" : {
         "street" : "2 Avenue",
         "zipcode" : "10075",
         "building" : "1480",
         "coord" : [ -73.9557413, 40.7720266 ],
      },
      "borough" : "Manhattan",
      "cuisine" : "Italian",
      "grades" : [
         {
            "date" : ISODate("2014-10-01T00:00:00Z"),
            "grade" : "A",
            "score" : 11
         },
         {
            "date" : ISODate("2014-01-16T00:00:00Z"),
            "grade" : "B",
            "score" : 17
         }
      ],
      "name" : "Vella",
      "restaurant_id" : "41704620"
   }
)
```

The method returns a `WriteResult` object with the status of the operation.

```
WriteResult({ "nInserted" : 1 })
```

If the document passed to the `insert()` method does not contain the `_id` field, the `mongo` shell automatically adds the field to the document and sets the field's value to a generated ObjectId.

Find or Query Data with the `mongo` Shell

Overview

You can use the `find()` method to issue a query to retrieve data from a collection in MongoDB. All queries in MongoDB have the scope of a single collection.

Queries can return all documents in a collection or only the documents that match a specified filter or criteria. You can specify the filter or criteria in a document and pass as a parameter to the `find()` method.

The `find()` method returns query results in a cursor, which is an iterable object that yields documents.

Prerequisites

The examples in this section use the `restaurants` collection in the `test` database. For instructions on populating the collection with the sample dataset, see Import Example Dataset.

In the `mongo` shell connected to a running `mongod` instance, switch to the `test` database.

```
use test
```

Query for All Documents in a Collection

To return all documents in a collection, call the `find()` method *without* a criteria document. For example, the following operation queries for all documents in the `restaurants` collection.

```
db.restaurants.find()
```

The result set contains all documents in the `restaurants` collection.

Specify Equality Conditions

The query condition for an equality match on a field has the following form:

```
{ <field1>: <value1>, <field2>: <value2>, ... }
```

If the `<field>` is a top-level field and not a field in an embedded document or an array, you can either enclose the field name in quotes or omit the quotes.

If the `<field>` is in an embedded document or an array, use dot notation to access the field. With dot notation, you must enclose the dotted name in quotes.

Query by a Top Level Field

The following operation finds documents whose `borough` field equals `"Manhattan"`.

```
db.restaurants.find( { "borough": "Manhattan" } )
```

The result set includes only the matching documents.

Query by a Field in an Embedded Document

To specify a condition on a field within an embedded document, use the dot notation. Dot notation *requires*quotes around the whole dotted field name. The following operation specifies an equality condition on the`zipcode` field in the `address` embedded document.

```
db.restaurants.find( { "address.zipcode": "10075" } )
```

The result set includes only the matching documents.

For more information on querying on fields within an embedded document, see Embedded Documents.

Query by a Field in an Array

The `grades` array contains embedded documents as its elements. To specify a condition on a field in these documents, use the dot notation. Dot notation *requires* quotes around the whole dotted field name. The following queries for documents whose grades array contains an embedded document with a field gradeequal to "B".

```
db.restaurants.find( { "grades.grade": "B" } )
```

The result set includes only the matching documents.

For more information on querying on arrays, such as specifying multiple conditions on array elements, seeArrays and $elemMatch.

Specify Conditions with Operators

MongoDB provides operators to specify query conditions, such as comparison operators. Although there are some exceptions, such as the $or and $and conditional operators, query conditions using operators generally have the following form:

```
{ <field1>: { <operator1>: <value1> } }
```

For a complete list of the operators, see query operators.

Greater Than Operator ($gt)
Query for documents whose grades array contains an embedded document with a field score greater than 30.

```
db.restaurants.find( { "grades.score": { $gt: 30 } } )
```

The result set includes only the matching documents.

Less Than Operator ($lt)
Query for documents whose grades array contains an embedded document with a field score less than10.

```
db.restaurants.find( { "grades.score": { $lt: 10 } } )
```

The result set includes only the matching documents.

Combine Conditions

You can combine multiple query conditions in logical conjunction (AND) and logical disjunctions (OR).

Logical AND

You can specify a logical conjunction (AND) for a list of query conditions by separating the conditions with a comma in the conditions document.

```
db.restaurants.find( { "cuisine": "Italian", "address.zipcode": "10075" }
)
```

The result set includes only the documents that matched all specified criteria.

Logical OR

You can specify a logical disjunction (OR) for a list of query conditions by using the `$or` query operator.

```
db.restaurants.find(
   { $or: [ { "cuisine": "Italian" }, { "address.zipcode": "10075" } ] }
)
```

The result set includes only the documents that match either conditions.

Sort Query Results

To specify an order for the result set, append the `sort()` method to the query. Pass to `sort()` method a document which contains the field(s) to sort by and the corresponding sort type, e.g. `1` for ascending and `-1`for descending.

For example, the following operation returns all documents in the `restaurants` collection, sorted first by the `borough` field in ascending order, and then, within each borough, by the `"address.zipcode"` field in ascending order:

```
db.restaurants.find().sort( { "borough": 1, "address.zipcode": 1 } )
```

The operation returns the results sorted in the specified order.

Update Data with the `mongo` Shell

Overview

You can use the `update()` method to update documents of a collection. The method accepts as its parameters:

- a filter document to match the documents to update,
- an update document to specify the modification to perform, and
- an options parameter (optional).

To specify the filter, use the same structure and syntax as the query conditions. See Find or Query Data with the mongo Shell for an introduction to query conditions.

By default, the `update()` method updates a single document. Use the `multi` option to update all documents that match the criteria.

You cannot update the `_id` field.

Prerequisites

The examples in this section use the `restaurants` collection in the `test` database. For instructions on populating the collection with the sample dataset, see Import Example Dataset.

In the `mongo` shell connected to a running `mongod` instance, switch to the `test` database.

```
use test
```

Update Specific Fields

To change a field value, MongoDB provides update operators, such as `$set` to modify values. Some update operators, such as `$set`, will create the field if the field does not exist. See the individual update operatorsreference.

Update Top-Level Fields

The following operation updates the first document with `name` equal to `"Juni"`, using the `$set` operator to update the `cuisine` field and the `$currentDate` operator to update the `lastModified` field with the current date.

```
db.restaurants.update(
    { "name" : "Juni" },
    {
      $set: { "cuisine": "American (New)" },
      $currentDate: { "lastModified": true }
    }
)
```

The update operation returns a `WriteResult` object which contains the status of the operation.

Update an Embedded Field

To update a field within an embedded document, use the dot notation. When using the dot notation, enclose the whole dotted field name in quotes. The following updates the `street` field in the embedded `address`document.

```
db.restaurants.update(
  { "restaurant_id" : "41156888" },
  { $set: { "address.street": "East 31st Street" } }
)
```

The update operation returns a `WriteResult` object which contains the status of the operation.

Update Multiple Documents

By default, the `update()` method updates a single document. To update multiple documents, use the`multi` option in the `update()` method. The following operation updates *all* documents that have`address.zipcode` field equal to `"10016"` and `cuisine` field equal to `"Other"`, setting the `cuisine`field to `"Category To Be Determined"` and the `lastModified` field to the current date.

```
db.restaurants.update(
  { "address.zipcode": "10016", cuisine: "Other" },
  {
    $set: { cuisine: "Category To Be Determined" },
    $currentDate: { "lastModified": true }
  },
  { multi: true}
)
```

The update operation returns a `WriteResult` object which contains the status of the operation.

Replace a Document

To replace the **entire** document except for the `_id` field, pass an entirely new document as the second argument to the `update()` method. The replacement document can have different fields from the original document. In the replacement document, you can omit the `_id` field since the `_id` field is immutable. If you do include the `_id` field, it must be the same value as the existing value.

IMPORTANT

After the update, the document **only** contains the field or fields in the replacement document.

After the following update, the modified document will **only** contain the `_id` field, `name` field, the `address`field. i.e. the document will *not* contain the `restaurant_id`, `cuisine`, `grades`, and the `borough` fields.

```
db.restaurants.update(
   { "restaurant_id" : "41704620" },
   {
     "name" : "Vella 2",
     "address" : {
              "coord" : [ -73.9557413, 40.7720266 ],
              "building" : "1480",
              "street" : "2 Avenue",
              "zipcode" : "10075"
     }
   }
)
```

The update operation returns a `WriteResult` object which contains the status of the operation.