COURSEWARE

Documents

Contents

Overview

Documents are how MongoDB stores data.
Documents use a key-value pair format very similar to JSON files.
Documents do not have to follow a strict schema *unless* one has been set when the collection was created.

Create

Insert a document by calling the **insertOne()** function on the collection you wish to add to and pass in the JSON object you want to store.

```
db.collectionName.insertOne({
    "firstName" : "Tadas",
    "lastName" : "Vaidotas",
    "age" : 33,
    "occupation" : "Trainer",
    "specialisation": "DevOps",
    "subjects": [
        "Docker",
        "AWS",
        "Scala"
    ]
})
```

It is possible to insert more than one document at a time by passing a JSON array into the **insertMany()** function.

```
db.collectionName.insertMany([
    {
        "firstName" : "Tadas",
        "lastName" : "Vaidotas",
        "age" : 33,
        "occupation" : "Trainer",
        "specialisation": "DevOps",
        "subjects": [
            "Docker",
            "AWS",
            "Scala"
        ]
    },
    {
        "firstName" : "Jordan",
        "lastName" : "Harrison",
        "age" : 25,
        "occupation" : "Trainer",
        "subjects": [
            "Java",
            "API dev",
            "Spring"
        ]
    }
])
```

When documents are inserted into a Mongo database they are each given a **primary key** which acts as a unique identifier for each document.
In Mongo this primary key is the **_id** field.

Read

Pull all documents from a collection using the **find()** function.

```
db.collectionName.find()
```

It is possible to only pull certain documents by passing a **query** object into the **find()** function.

Projection

If you only want to find certain fields rather than the whole document then you can pass a **projection** object into the **find()** function, this will **project** out the fields we want into the query results.

```
db.practice.find(
    {},
    {
        "firstName": true,
        "lastName": true
    }
)
```

This query will find *all* documents (because we passed in a blank filter) but the only fields that will be displayed are **firstName**, **lastName** and **_id**.
This is because the **id** field will always be displayed unless specifically excluded, like so:

```
db.collectionName.find(
    {},
    {
        "_id": false
        "firstName": true,
        "lastName": true
    }
)
```

Note that normally it is only possible to include *or* exclude fields in a single projection but the **_id** field is an exception to this rule and can always be excluded.

Queries

It is possible to create very simple queries in Mongo by passing in partial objects.
For example, if you wanted to find someone with a first name of 'Tadas'

```
db.collectionName.find(
    {
        "firstName": "Tadas"
    }
)
```

For more complicated queries you can use *query operators*.

Equals

The **$eq** operator basically works in the same way as using no operator at all.
i.e. We could have written the previous example as

```
db.collectionName.find(
    {
        "firstName": { "$eq": "Tadas" }
    }
)
```

Obviously not that much point in using this operator so let's look at the not equals operator **$ne**.

```
db.collectionName.find(
    {
        "firstName": { "$ne": "Tadas" }
    }
)
```

This will flip the previous query and instead find every document where the firstName is *not* 'Tadas'.

Greater/Less than

For numerical fields it is possible to find all docs where a value is *greater than* a particular number using **$gt**.
For example, finding all trainers over the age of thirty:

```
db.collectionName.find(
    {
        "age": { "$gt": 30 }
    }
)
```

You can specify *greater than or equal to* conditions using **$gte**.
This will find any trainers that are *at least* 30:

```
db.collectionName.find(
    {
        "age": { "$gte": 30 }
    }
)
```

Like wise we can do *less than* with **$lt**:

```
db.collectionName.find(
    {
        "age": { "$lt": 30 }
    }
)
```

And *less than or equal to* with **$lte**:

```
db.collectionName.find(
    {
        "age": { "$lte": 30 }
    }
)
```

In/Nin

```
db.collectionName.find(
    {
        "specialisation": {
                        "$in": [
                            "Software Dev",
                            "DevOps"
                            ]
                        }
    }
)
```

```
db.collectionName.find(
    {
        "specialisation": {
                        "$nin": [
                            "Software Dev",
                            "DevOps"
                            ]
                        }
    }
)
```

Update

To change existing documents you will need to use **update query operators**, for example if I wanted to give Jordan a specialisation I could use the **updateOne()** function, passing in a filter and an update.

```
db.collectionName.updateOne(
    {
    "firstName": "Jordan",
    "lastName": "Harrison"
    },
    {
        "$set" : {
            "specialisation": "Software Development"
            }
    }
)
```

Note the use of the **$set** operator.
In MongoDB **$** is used to signify system fields or operators - in this case it is used to differentiate between the set **operator** and just a field called set.
It is possible to update multiple documents in a similar manner using the **updateMany()** function.
The only real difference between **updateMany** and **updateOne** is that **updateOne** stops looking after it finds any document that matches the filter, whereas **updateMany** will search the whole collection looking for any matching documents.

```
db.collectionName.updateMany(
    {},
    {
        "$set": {
            "reportsTo": "John Gordon"
            }
    }
)
```

By passing in an empty filter the update operation will target *every* document in the collection and add the new field "reportsTo" with a value of "John Gordon".

If you want to replace an existing document with a brand-new one rather than updating an existing doc you can use the **replaceOne()** function.

```
db.collectionName.replaceOne()
```

Arrays

Values can be added into arrays using the **$push** operator

```
db.collectionName.updateOne(
    {
        "firstName": "Jordan",
        "lastName": "Harrison"
    },
    {
        "$push": {
            "subjects": "MongoDB"
        }
    }
)
```

And similarly removed using the **$pull** operator.

```
db.collectionName.updateOne(
    {
        "firstName": "Tadas",
        "lastName": "Vaidotas"
    },
    {
        "$pull": {
            "subjects": { "$nin": "scala"}
        }
    }
)
```

Delete

The delete functions work in much the same way as the update functions except without the need for update operators.
For example, if you want to delete one document:

```
db.collectionName.deleteOne(
    {
        "firstName": "Jordan"
    }
)
```

This will delete the first document found where the firstName field has a value of Jordan.
Similarly you can delete *all* the documents that meet a certain criteria:

```
db.collectionName.deleteMany(
    {
        "specialisation": "DevOps"
    }
)
```

This command will delete any trainer with a DevOps specialisation.

Embedded Documents

Through the power of JavaScript it is possible to put a document *inside* of another document.
This is typically used to represent an entity that belongs to another entity - for

example, we might represent a person's job as a separate entity *embedded* inside the person entity.

```
{
    "firstName": "Jordan",
    "surname": "Harrison",
    "age": 25,
    "height": 182,
    "hobbies":  [
        "Gaming",
        "Reading",
        "Writing course-ware"
    ],
    "job": {
        "title": "Learning specialist",
        "salary": 1000000000,
        "startDate": new Date("2018-09-24"),
        "manager": "Christopher Perrins"
    }
}
```

Tutorial

Exercises

1. Create a new database called **document_practice**.

▶ Show solution

2. Create an object to represent yourself - this object should contain information like; first name, surname, age, height and hobbies.

▶ Show solution

3. Insert the previous object into a new collection called **people**.

▶ Show solution

4. Do the same for everyone in your row in *one query*.

▶ Show solution

From now on we will be using the answer for the previous question as the practice dataset.

5. Ben starts playing roulette.

▶ Show solution

6. Sally stops playing air guitar.

▶ Show solution

7. Find everyone over 50.

▶ Show solution

8. Find everyone under 30 *but only show their first and last name*.

▶ Show solution

9. Everyone over 30 now wears glasses.

▶ Show solution

10. Delete everyone not wearing glasses.

▶ Show solution

11. Ben decides to buy a dog, update his document in the collection to have an *embedded* document representing this dog (name this field 'pet'). Make sure to give the embedded doc sufficient information (e.g. name, age, species)

▶ Show solution