Denys Savytskyi

MongoDB

Introduction:

MongoDB is an open-source cross-platform document-oriented database system developed and supported by MongoDB Inc and initially released in 2009. MongoDB stores data as JSON-like documents and is written in C++, Go, JavaScript and Python. Since its release, MongoDB has become one of the most popularly used NoSQL dataBase systems due to its ease of use and efficiency. It uses binary wire protocol as its primary mode of interaction with the server. MongoDB is available for all the major operation systems like Windows, MacOS, Linux and support drivers for nearly every programming language.

Install:

MongoDB can be installed locally, which will allow you to host your own MongoDB server. You can download and use the MongoDB open source Community Server on your hardware for free. To avoid unnecessary moves you can docker. Official docker image available. You can also prefer cloud solution and use a free MongoDB Atlas Shared Cluster. Unlike SQL databases, where you must determine and declare a table's schema before inserting data, MongoDB's collections, by default, do not require their documents to have the same schema.

Overview:

MongoDB stores data records as documents in BSON (binary json) - a JSON-like format. The documents are grouped into collections. Each database stores one or more collections of documents and each collection includes documents. Unlike SQL databases, where you must determine and declare a table's schema before inserting data, MongoDB's collections, by default, do not require their documents to have the same schema.

```
{
    "_id": <ObjectId>,
    "name": "Sue",
    "age": "25",
    "status": "A",
    "groups": [ "news", "sport" ]
}
```

In practice, however, the documents in a collection share a similar structure, and you can enforce validation rules for a collection during update and insert operations. Besides this, every document stored in a collection requires a unique **_id** field that acts as a primary key. If

an inserted document omits the **_id** field, the MongoDB driver automatically generates an ObjectId for the **_id** field.

Relationships:

Just like RDBMS MongoDB has a supported main types of relations:

- **One-to-One Relationships** are implemented as embedded documents

```
// order document
Order={
    "_id": "2934f",
    "salesDate": "2022-05-02",
    "customer"
}
// customer document
{
    "name": "Jack Beanstalk",
    "gender": "M",
    "rewardsMember": "True"
    "order_ip": zxc""
}
```

```
// ONE-TO-ONE
Order={
    "_id": "2934f",
    "salesDate": "2022-05-02",
    "customer":{
        "name": "Jack Beanstalk",
        "gender": "M",
        "rewardsMember": "True"
    }
}
```

- **One-to-One Relationships** are implemented either as array of embedded documents

```
// ONE-TO-MANY
Order={
"_id": "2934f",
"salesDate": "2022-05-02",
"customer":{
    "name": "Jack Beanstalk",
    "gender": "M",
    "rewardsMember": "True"
},
"items": [ {
        "name": "Monstera",
        "price":{
            "$numberdecimal": "8.00"
        },
        "quantity":{
            "$numberInt": "1"
        }
    },
    {
        "name": "Pothos",
        "price":{
            "$numberdecimal": "8.00"
        },
        "quantity":{
            "$numberInt": "2"
        }
    } ]
}
```

or as array of embedded links

```
Order={
"_id": "2934f",
"salesDate": "2022-05-02",
"customer": {
        "name": "Jack Beanstalk",
        "gender": "M",
        "rewardsMember": "True"
    },
"items": [ "12", "35", "86"]
}
```

- **Many-to-Many Relationships** are implemented as array of embedded links with arrays on either side

```
Order={
    "_id": "2934f",
    "salesDate": "2022-05-02",
    "customer":{
        "name": "Jack Beanstalk",
        "gender": "M",
        "rewardsMember": "True"
    },
    "items": [ "12", "35", "86"]
}

Item={
    "_id": "12",
    "name": "Pothos",
    "price":{
        "$numberDecimal": "8.00"
    },
    "orders": [ "2934f", "1b2df", "43de9"]
    21
}
```

- **Tree Structure** with Child References

```
[
    { "_id": "MongoDB", "children": [] },
    { "_id": "dbm", "children": [] },
    { "_id": "Databases", "children": [
         "MongoDB",
         "dbm"
        ] },
    { "_id": "Languages", "children": [] },
    { "_id": "Programming", "children": [
        "Databases",
        "Languages"
        ] },
    { "_id": "Books", "children": [ "Programming" ] }
 ]
```

CRUD Operations:

The MongoDB Shell **mongosh** could be used for o testing queries and operations directly with your database. More about **mongosh** here.

- Create or insert operations add new documents to a collection. If the collection does not currently exist, insert operations will create the collection.MongoDB provides the following methods to insert documents into a collection:

  db.collection.insertOne()
  db.collection.insertMany()

```
db.collection.insertOne({    // collection
    "name": "Sue",           // field: value
    "age": "25",             // field: value
    "status": "A"            // field: value
})
```

- Read operations retrieve documents from a collection; i.e. query a collection for documents. MongoDB provides the following methods to read documents from a collection:

  db.collection.find()

```
db.users.find(                    // collection
    {"age": { $gt: 18 } },        // queue filter
    { "name": 1, "adress": 1 }    // projection
).limit(5)                        // cursor modifier
```

- Update operations modify existing documents in a collection. MongoDB provides the following methods to update documents of a collection:

  db.collection.updateOne()
  db.collection.updateMany()
  db.collection.replaceOne()

```
db.users.updateMany(                      // collection
    {"age": { $gt: 18 } },                // update filter
    { "$set": { "status": "reject" } }    // update action
)
```

- Delete operations remove documents from a collection. MongoDB provides the following methods to delete documents of a collection:

  db.collection.deleteOne()
  db.collection.deleteMany()

```
db.users.deleteMany(        // collection
    {"status": "reject" }   // delete filter
)
```

Conclusion

Thanks to the flexibility of document-oriented databases, determining the best way to model relationships in a document database is less of a strict science than it is in a relational database. Although concepts are completely different, there still are some similarities.

RDBMS	Table	Row	Column	Joins
MongoDB	Collection	Document	Field	Embedded documents

Learn More:

- Mongo Docs
- W3S Mongo
- Mongo Manual
- BSON
- Mongosh Docs