

New

Get 50% off your ticket to MongoDB.local NYC on May 2. Use code Web50!

Quick Start: Golang & MongoDB - Modeling Documents with Go Data Structures

[Try MongoDB Atlas Free](#)[Nicolas Raboy](#)

February 6, 2020 | Updated: May 20, 2021

[#go](#)

In the past few getting started tutorials, we explored the various ways to interact with MongoDB data using the [Go programming language](#) (Golang). In particular, we spent time exploring each of the [create](#), [retrieve](#), [update](#), and [delete](#) (CRUD) operations, which are critical when building amazing applications.

In each of the tutorials, we made use of `bson.A`, `bson.D`, and `bson.M`, which represent arrays, documents, and maps. However, these are not the only primitive data structures that are part of the MongoDB Go driver, and they are not necessarily the best way for interacting with data, both within the application and the database.

We're going to look at an alternative way to interact with data through the MongoDB Go driver operations. This time we're going to map MongoDB document fields to native Go data structures.

[Contact Us](#)

The Requirements



To be successful with this tutorial, the following should be met:

- A MongoDB Atlas cluster
- Go 1.10+

You should already have Go installed and configured as this tutorial won't explain how to do this. You should also have your [MongoDB Atlas](#) cluster properly configured with the correct whitelisting. If you need help doing this, check out my [previous tutorial](#) on the subject.

While MongoDB Atlas has a forever FREE tier, apply promotional code [NICRABOY200](#) to receive credit towards a more powerful cluster.

If you haven't experienced the previous tutorials in the series, viewing them would be beneficial, but not a requirement.

Creating a Go Data Structure with BSON Annotations

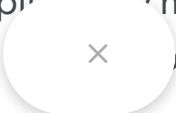
The premise of this tutorial is to use a native Go data structure when working with MongoDB, but let's take a step back and look at what we were working with in the past few tutorials.

You might remember something like the following:

```
bson.M{
  "title": "The Polyglot Developer Podcast",
  "author": "Nic Raboy",
  "tags": bson.A{"development", "programming", "coding"}
}
```

[Contact Us](#)

Working with a `bson.M` will leave you with a `map[string]interface{}` which isn't the most complicated in the world, but isn't necessarily the best for all scenarios in my opinion. Things get more challenging as you start working with `bson.D` as well.

A lot of these scenarios can be simplified by mapping the fields of the document to fields of a data structure, similar to how  does it with JSON and XML.

Take the following Go data structure:

```
type Podcast struct {
    ID      primitive.ObjectID `bson:"_id,omitempty"`
    Title   string             `bson:"title,omitempty"`
    Author  string             `bson:"author,omitempty"`
    Tags    []string           `bson:"tags,omitempty"`
}
```

The above data structure is nearly identical to the `bson.M` that was used previously, with the exception that it has an `ID` field. You'll notice the BSON annotations. These annotations are the actual fields that would appear in the document within MongoDB. The `omitempty` means that if there is no data in the particular field, when saved to MongoDB the field will not exist on the document rather than existing with an empty value.

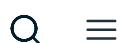
Some of the benefits to using native Go data structures to represent our data is having autocomplete available, error handling, and being able to write methods specific to the data structure.

So how can we work with our documents? Take the following:

```
podcast := Podcast{
    Title: "The Polyglot Developer",
    Author: "Nic Raboy",
    Tags: []string{"development", "programming", "coding"},
}
```

[Contact Us](#)

While `Podcast` is not more difficult to create than a `bson.M`, interacting with documents will be a different story depending on the complexity of your document.



Go Data Structure

Now that we have a general idea on how to create native Go data structures with BSON annotations, let's convert some of our previous examples to use them

with Find and Insert operations.

Create a `main.go` file within your `$GOPATH` with the following boilerplate code:

```
package main

import (
    "context"
    "fmt"
    "os"
    "time"

    "go.mongodb.org/mongo-driver/bson"
    "go.mongodb.org/mongo-driver/bson/primitive"
    "go.mongodb.org/mongo-driver/mongo"
    "go.mongodb.org/mongo-driver/mongo/options"
)

type Podcast struct {
    ID      primitive.ObjectID `bson:"_id,omitempty"`
    Title   string             `bson:"title,omitempty"`
    Author  string             `bson:"author,omitempty"`
    Tags    []string           `bson:"tags,omitempty"`
}

type Episode struct {
    ID          primitive.ObjectID `bson:"_id,omitempty"`
    Podcast     primitive.ObjectID `bson:"podcast,omitempty"`
    Title       string             `bson:"title,omitempty"`
    Description string             `bson:"description,omitempty"`
    Duration    int32              `bson:"duration,omitempty"`
}

func main() {
    ctx, _ := context.WithTimeout(context.Background(), 10*time.Second)
    client, err := mongo.Connect(ctx, options.Client().ApplyURI(os.Getenv("MONGODB_URI")))
    if err != nil {
        panic(err)
    }
    defer client.Disconnect(ctx)

    database := client.Database("quickstart")
    podcastsCollection := database.Collection("podcasts")
    episodesCollection := database.Collection("episodes")
}
```

[Contact Us](#)

We're going to assume that you've installed the MongoDB Go driver as outlined in the previous tutorials. Take note in the code that my MongoDB Atlas URI is stored in my environment variables to prevent exposing it in my source code.

Let's assume that we have documents in our collections as of now. We can try adding the following code:

```
var episodes []Episode
cursor, err := episodesCollection.Find(ctx, bson.M{"duration": bson.D{{$gt, 25}}
if err != nil {
    panic(err)
}
if err = cursor.All(ctx, &episodes); err != nil {
    panic(err)
}
fmt.Println(episodes)
```

Instead of creating a []bson.D we are creating a []Episode and loading the cursor results into it. From there, if we wanted to, we could interact with each item in the slice and access each field without any manual manipulation.

So now let's try to create some data.

Creating data with a native Go data structure isn't any more difficult than retrieving it. We could add something like the following:

```
podcast := Podcast{
    Title: "The Polyglot Developer",
    Author: "Nic Raboy",
    Tags: []string{"development", "programming", "coding"},
}
insertResult, err := podcastsCollection.InsertOne(ctx, podcast)
if err != nil {
    panic(err)
}
fmt.Println(insertResult.InsertedID)
```

[Contact Us](#)

The same rules can be applied when updating or removing data from a collection as well.

Conclusion



You just saw how to map MongoDB document fields to fields within native Go data structures using the MongoDB Go driver with the Go programming language. Being able to work with data directly how you'd find it in the database is a huge benefit as there aren't any complicated marshalling and unmarshalling that needs to be done manually within your application.

If you need to catch up with your Go and MongoDB skills, take a look at the other tutorials that appeared in this series:

1. [How to Get Connected to Your MongoDB Cluster with Go](#)
2. [Creating MongoDB Documents with Go](#)
3. [Retrieving and Querying MongoDB Documents with Go](#)
4. [Updating MongoDB Documents with Go](#)
5. [Deleting MongoDB Documents with Go](#)

In future tutorials we're going to take a look at the MongoDB aggregation framework, change streams, and transactions, all using the [Go programming language](#).

[Contact Us](#)

[< Previous](#)

Making an Impact as a Solution Architect at MongoDB - Meet Winston Vargo

Enterprise Solutions Architect, Winston Vargo talks about his life as an SA at MongoDB.

February 5, 2020

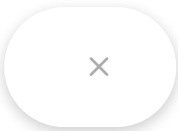
[Next →](#)

Building AI With MongoDB: Integrating Vector Search And Cohere to Build Frontier...

Cohere is the leading enterprise AI platform, building large language models (LLMs) which help businesse...

April 25, 2024

[Contact Us](#)



About

Careers

Investor Relations

Legal Notices

Privacy Notices

Security Information

Trust Center

Support

Contact Us

Customer Portal

Atlas Status

Customer Support

Social

 GitHub

 Stack Overflow

 LinkedIn

 YouTube

 Twitter

 Twitch

 Facebook

Contact Us