MonGolang V0.2.2 - Demo

Code to demo MonGolang Version 0.2.2 Changes. These include:

1. Bug fix of `FindOne()` with no parameters
2. Support for additonal Mongo Shell functions:
   - `InsertOne()`
   - `InsertMany()`
   - `DeleteOne()`
   - `DeleteMany()`
3. New `PrintBSON()` function
4. Support for MongoDB Extended JSON

Setup

In [1]:
```
import (
    "fmt"

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

    "github.com/ddgarrett/mongolang"
)
```

In [2]:
```
db := mongolang.DB{}
db.InitMonGolang("mongodb://localhost:27017").Use("quickstart")
db.ShowCollections()
```

Out[2]:
```
[zips amazon testCollection episodes podcasts]
```

FindOne Bug and `PrintBSON()` Function

Previously the `FindOne()` method would not return any results when not passed any parameters. This now works correctly.

The example below also illustrates the use of the new `PrintBSON()` function. The new function prints the contents and type of any type of bson structure.

In [3]:
```
mongolang.PrintBSON(*db.Coll("zips").FindOne())
```

```
{
   _id (string): 01001
   city (string): AGAWAM
   loc (A): [-72.622739, 42.070206]
   pop (int32): 15338
   state (string): MA
 }
```

`InsertOne()`

MonGolang now supports Mongo Shell functions `InsertOne()` and `InsertMany()`.

In [4]:
```
insertDocJSON := `{
    "title": "The Polyglot Developer Podcast",
    "author": "Nic Raboy",
    "tags": ["development", "programming", "coding"] }`

insertOneResult := db.Coll("testCollection").InsertOne(insertDocJSON)
```

Extended JSON

To verify that we did insert a new document and also to illustrate the use of extended JSON, we'll next read the document we just inserted using the generated Object ID in `insertOneResult`.

In [5]:
```
oid := insertOneResult.InsertedID.(primitive.ObjectID)
filter := fmt.Sprintf(`{"_id": { "$oid": "%s" }}`,oid.Hex())
filter
```

Out[5]:
```
{"_id": { "$oid": "606b39d55b0bbc28b967764f" }}
```

Extended JSON defines an `$oid` field type which we've used to define the filter.

In the code above we first cast the `insertOneResult.InsertedID` as a `primitive.ObjectID` from the `go.mongodb.org/mongo-driver/bson/primitive"` import library.

We then use that `primitive.ObjectID` to obtain the hex string generated for the `InsertOne()` and place it in an extended JSON string to define the `$oid` value.

Now let's read the just inserted document using the extended JSON string filter.

In [6]:
```
db.Coll("testCollection").Find(filter).Pretty()
```

Out[6]:
```
{
    _id : ObjectID("606b39d55b0bbc28b967764f")
    title : The Polyglot Developer Podcast
    author : Nic Raboy
    tags : [development programming coding]
}
```

You can read more about the JSON extensions available along with examples at https://docs.mongodb.com/manual/reference/mongodb-extended-json/ (https://docs.mongodb.com/manual/reference/mongodb-extended-json/)

Note that the above code, converting the generated Object ID to an extended JSON string was just done for illustrative purposes. In this particular case it actually would have been easier as well as much more efficient to skip the JSON string and just generate a `bson.M` filter directly as shown below.

In [7]:
```
bsonFilter := bson.M{"_id":insertOneResult.InsertedID}
bsonFilter
```

Out[7]:
```
map[_id:ObjectID("606b39d55b0bbc28b967764f")]
```

In [8]:
```
db.Coll("testCollection").Find(bsonFilter).Pretty()
```

Out[8]:
```
{
    _id : ObjectID("606b39d55b0bbc28b967764f")
    title : The Polyglot Developer Podcast
    author : Nic Raboy
    tags : [development programming coding]
}
```

InsertMany()

In [9]:
```
multipleJSONDoc := `[
    { "title": "The Polyglot Developer Podcast Version 2",
      "author": "Nic Raboy",
      "tags": ["development", "programming", "coding"] },

    { "title": "The Polyglot Developer Podcast Version 3",
        "author": "Nic Raboy",
        "tags": ["development", "programming", "coding"] },

    { "title": "The Polyglot Developer Podcast Version 4",
          "author": "Nic Raboy Jr.",
          "testCase": "Nic Raboy Jr. will not be deleted first time",
          "tags": ["development", "programming", "coding"] }
]`

insertManyResult := db.Coll("testCollection").InsertMany(multipleJSON
Doc)
```

Doing a `Find()` using the results of the `InsertMany()` is a bit more complicated. Looking at the results returned we see:

In [10]:
```
mongolang.PrintStruct(insertManyResult)
```

```
{
  "InsertedIDs": [
    "606b39d55b0bbc28b9677650",
    "606b39d55b0bbc28b9677651",
    "606b39d55b0bbc28b9677652"
  ]
}
```

As before, the most efficient way to create a `Find()` filter from `ObjectID` slice is best done without converting to JSON first, though the code is a bit more complex than the `Find()` filter created after the `InsertOne()`.

In [11]:
```
bsonFilterMany :=
        bson.M{"_id": bson.M{ "$in":insertManyResult.InsertedIDs}}

db.Coll("testCollection").Find(bsonFilterMany,`{"title":1}`).Pretty()
```

Out[11]:
```
{
    _id : ObjectID("606b39d55b0bbc28b9677650")
    title : The Polyglot Developer Podcast Version 2
}
{
    _id : ObjectID("606b39d55b0bbc28b9677651")
    title : The Polyglot Developer Podcast Version 3
}
{
    _id : ObjectID("606b39d55b0bbc28b9677652")
    title : The Polyglot Developer Podcast Version 4
}
```

Delete Methods

MonGolang now also supports Mongo Shell functions `DeleteOne()` and `DeleteMany()`.

We'll use the previously defined filters from the `InsertMany()` to specify which documents to delete.

In [12]:
```
db.Coll("testCollection").DeleteOne(bsonFilterMany)
```

Out[12]:
```
&{1}
```

Note that the `DeleteOne()` as well as the `DeleteMany()` simply return the number of documents deleted.

Now let's rerun the previous `Find()` to see which documents remain from the `InsertMany()`. Note that exactly which document is deleted when the criteria matches multiple documents is unpredictable.

In [13]:
```
db.Coll("testCollection").Find(bsonFilterMany,`{"title":1}`).Pretty()
```

Out[13]:
```
{
    _id : ObjectID("606b39d55b0bbc28b9677651")
    title : The Polyglot Developer Podcast Version 3
}
{
    _id : ObjectID("606b39d55b0bbc28b9677652")
    title : The Polyglot Developer Podcast Version 4
}
```

We now delete the remaining two documents from the `InsertMany()` and then show which documents still remain int he `testCollection` collection.

In [14]:
```
db.Coll("testCollection").DeleteMany(bsonFilterMany)
```

Out[14]:
```
&{2}
```

In [15]:
```
db.Coll("testCollection").Find(`{}`,`{"title":1}`).Pretty()
```

Out[15]:
```
{
    _id : ObjectID("606b39d55b0bbc28b967764f")
    title : The Polyglot Developer Podcast
}
```

As you can see we now have only the document inserted from the previous `InsertOne()` call. Let's delete any remaining documents using the `DeleteMany()` with an empty filter.

In [16]:
```
db.Coll("testCollection").DeleteMany(`{}`)
```

Out[16]:
```
&{1}
```

Finally we close the database.

In [17]:
```
db.Disconnect()
```