NoSQL

Introduction

By Eman Fathi

CREATE READ UPDATE DELETE

C R U D

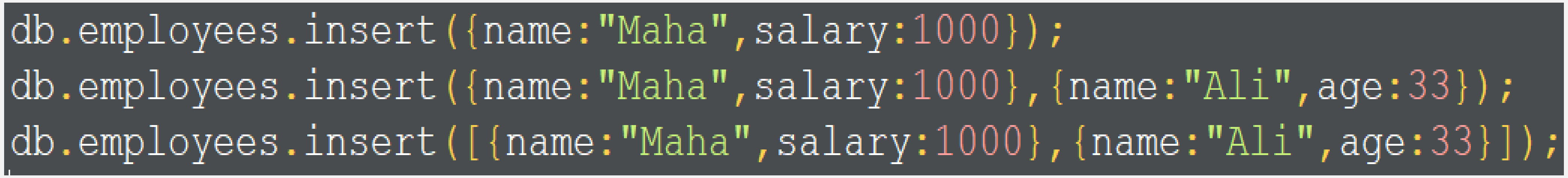
C CREATE

db.collection.insert();

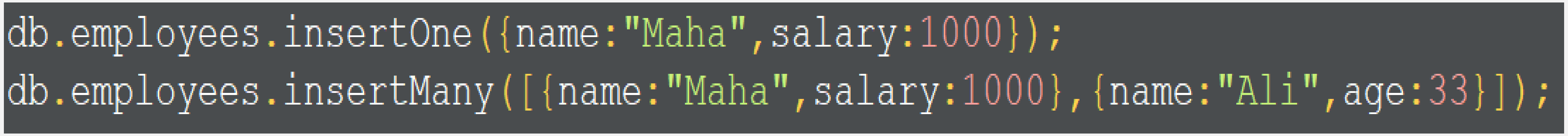
insert(): insert document or documents into a collection

insertOne(): insert only one document into a collection

insertMany(): insert multiple documents into a collection



```
db.employees.insert({name:"Maha",salary:1000});
db.employees.insert({name:"Maha",salary:1000},{name:"Ali",age:33});
db.employees.insert([{name:"Maha",salary:1000},{name:"Ali",age:33}]);
```



```
db.employees.insertOne({name:"Maha",salary:1000});
db.employees.insertMany([{name:"Maha",salary:1000},{name:"Ali",age:33}]);
```

ObjectId

The ObjectId Class is the default primary key for a MongoDB document and is found in _id field in an inserted document.

```
{
    "_id": ObjectId("54759eb3c090d83494e2d804")
}
```

✓Immutable

✓Unique

✓BSON DataType

✓12 byte Value

D DELETE

db.collection.remove();

Remove all documents

```
db.customers.remove({});
```

Remove a document according to specified condition

```
db.customers.remove({salary:{$gt:1000}});
```

R READ

db.collection.find(query,projection);

find(): returns all matched documents , if there is no condition returns all collections documents

findOne(): return only one matching document , the first matching document

```
db.employees.find();
```

All collection documents

```
db.employees.find({name:"eman"})
```

all documents with name =‘eman’

```
db.employees.find({age:{$lt:20}})
```

all documents with age<20

```
db.employees.findOne({age:{$lt:20}})
```

first document with age<20

```
db.employees.find({}).pretty()
```

result as readable json objects

Operators

Comparisons operators

Array operators

logical operators

element operators

Update operators

Comparisons Query operators

Name	Description
$eq	Matches values that are equal to a specified value.
$gt	Matches values that are greater than a specified value.
$gte	Matches values that are greater than or equal to a specified value.
$in	Matches any of the values specified in an array.
$lt	Matches values that are less than a specified value.
$lte	Matches values that are less than or equal to a specified value.
$ne	Matches all values that are not equal to a specified value.
$nin	Matches none of the values specified in an array.

Comparisons Query operators

All documents that have salaries less than 2000

```
db.employees.find({salary:{$lt:2000}});
```

All documents that have salaries greater than or equal 2000

```
db.employees.find({salary:{$gte:2810}});
```

All documents that have salaries between 1000 and 3000

```
db.employees.find({salary:{$lt:3000,$gt:1000}});
```

Any document has age value inside $in array

```
db.employees.find({age:{$in:[22,35,25,40]}});
```

All documents that have salaries equal 2810

```
db.employees.find({salary:{$eq:2810}});
```

Logical Query operators

Name	Description
$and	Joins query clauses with a logical AND returns all documents that match the conditions of both clauses.
$not	Inverts the effect of a query expression and returns documents that do *not* match the query expression.
$nor	Joins query clauses with a logical NOR returns all documents that fail to match both clauses.
$or	Joins query clauses with a logical OR returns all documents that match the conditions of either clause.

Logical Query operators

```
db.employees.find({$or:[{name:"hassan"},{salary:3030}]});
```

```
db.employees.find({$and:[{name:"hassan"},{salary:{$gte:30}}]});
```

```
db.employees.find({$nor:[{salary:2810},{name:"hassan"}]});
```

```
db.employees.find({salary:{$not:{$gt:3000}}});
```

Element Query operators

Name	Description
$exists	Matches documents that have the specified field.
$type	Selects documents if a field is of the specified type.

```
db.employees.find({count:{$type:"string"}});
```

```
db.employees.find({count:{$exists:""}});
```

Array Query operators

Name	Description
$all	Matches arrays that contain all elements specified in the query.
$elemMatch	Selects documents if element in the array field matches all the specified $elemMatchconditions.
$size	Selects documents if the array field is a specified size.

```
db.employees.find({dresses:{$all:["black","red"]}});
```

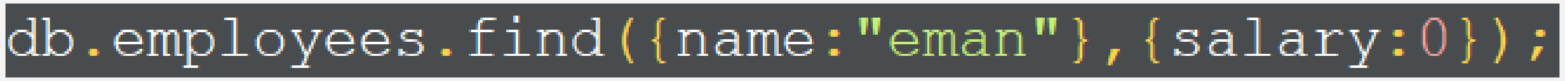
```
db.employees.find({data:{$elemMatch:{$gt:3,$lt:6}}});
```

```
db.employees.find({dresses:{$size:5}})
```

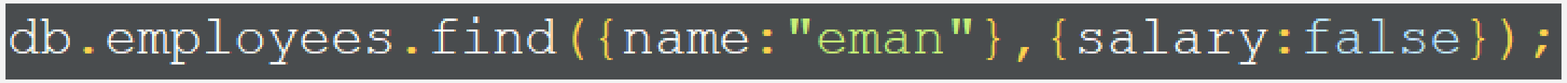
READ

Projection

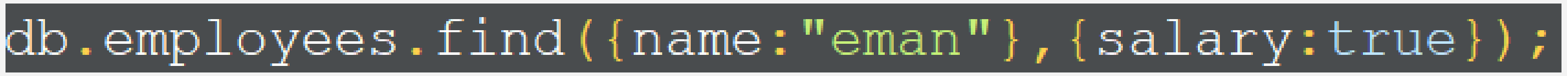
projection means selecting only the necessary data rather than selecting whole of the data of a document.



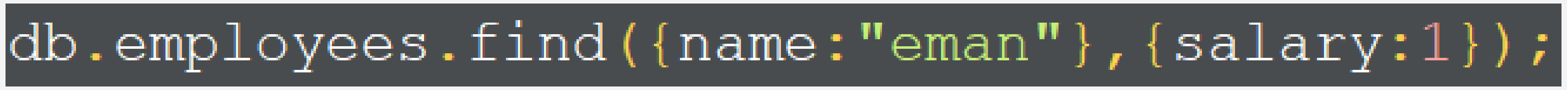
```
db.employees.find({name:"eman"},{salary:0});
```



```
db.employees.find({name:"eman"},{salary:false});
```



```
db.employees.find({name:"eman"},{salary:true});
```



```
db.employees.find({name:"eman"},{salary:1});
```

UPDATE

db.collection.update(query,update,option);

update(): updates one documents or documents in a collection

updateOne(): updates single document in a collection

updateMany(): updates multiple documents in a collection

```
db.employees.update({name:"eman"},{$set:{salary:5600}});
```

```
db.employees.update({name:"eman"},{$set:{salary:5600,age:20}});
```

```
db.employees.update({name:"eman"},{$set:{address:"Mansoura"}},{upsert:true});
```

```
db.employees.update({age:{$gt:20}},{$set:{count:1}},{multi:true});
```

UPDATE

operators

Field Operators

Name	Description
$inc	Increments the value of the field by the specified amount.
$min	Only updates the field if the specified value is less than the existing field value.
$max	Only updates the field if the specified value is greater than the existing field value.
$mul	Multiplies the value of the field by the specified amount.
$set	Sets the value of a field in a document.
$unset	Removes the specified field from a document.

UPDATE

Field Operators

```
db.employees.update({name:"eman"},{$set:{salary:5600},$inc:{count:1}});
```

```
db.employees.update({name:"eman"},{$max:{count:3}});
```

```
db.employees.update({name:"eman"},{$min:{count:3}});
```

```
db.employees.update({name:"eman"},{$mul:{count:3}});
```

```
db.employees.update({},{$unset:{color:"red"}},{multi:true});
```

```
db.employees.update({},{$rename:{"count":"counter"}},{multi:true});
```

UPDATE operators

Array Operators

Name	Description
$	Acts as a placeholder to update the first element that matches the query condition.
$[]	Acts as a placeholder to update all elements in an array for the documents that match the query condition.
$pop	Removes the first or last item of an array.
$pull	Removes all array elements that match a specified query.
$push	Adds an item to an array.

UPDATE

operators

Array Operators

```
db.employees.update({name:"eman"},{$set:{"books.2":"JQuery"}});
```

```
db.employees.update({name:"eman"},{$set:{"books.$":"ES"}})
```

```
db.employees.update({name:"eman"},{$set:{"books.$[]":"ES"}})
```

```
db.employees.update({name:"eman"},{$push:{books:"nodejs"}});
```

Document Validations version 3.2

jsonSchema Validation version3.6

Document Validation

- ✓ Introduced in MongoDB3.2
- ✓ MongoDB provides the capability to validate documents during updates and insertions. Validation rules are specified on a per-collection basis using the validator option, which takes a document that specifies the validation rules or expressions.

To specify document validation on a new collection, use the new validator option in the db.createCollection() method.

To add document validation to an existing collection, use the new validator option in db.runComm() method

Validation occurs during updates and inserts. Existing documents do not undergo validation checks until modification.

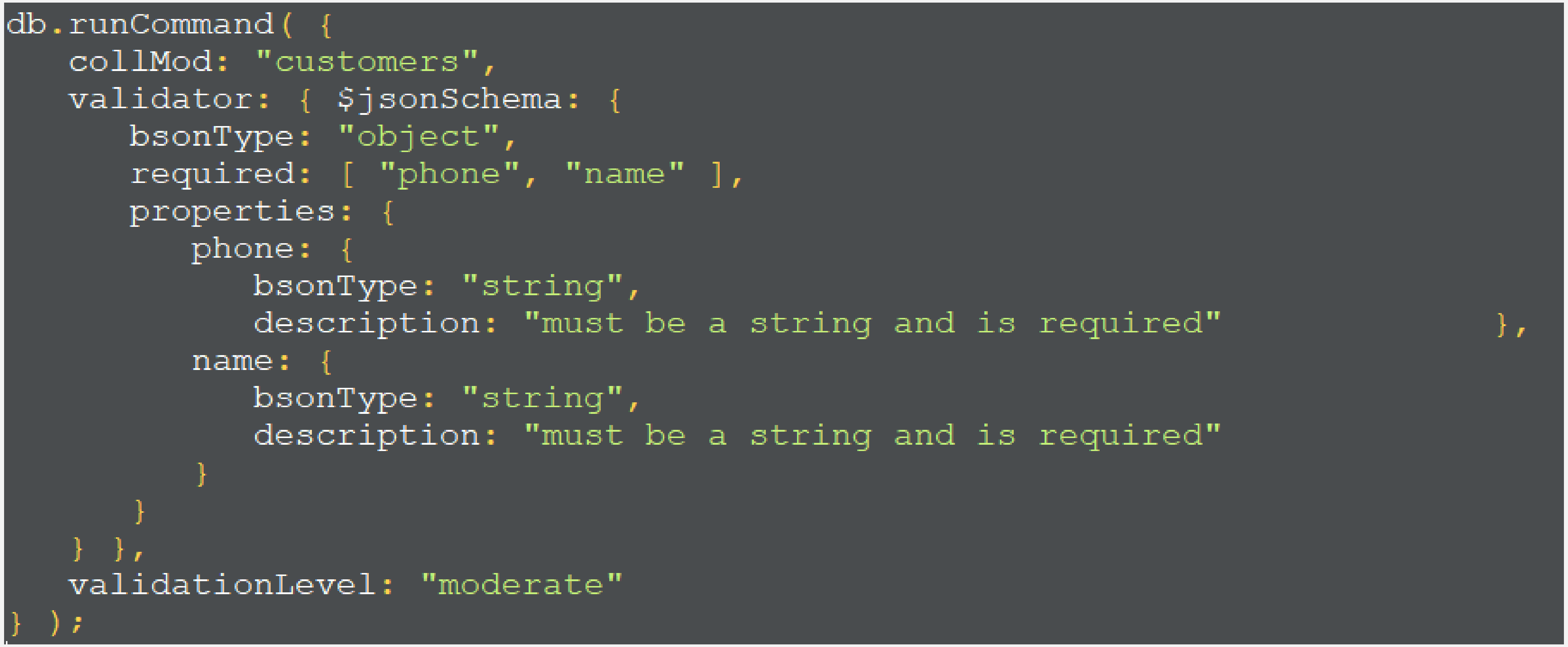
To view the validation specifications for a collection, use the db.getCollectionInfos() method.

Json Schema Validation

- Introduced in MongoDB 3.6
- a “Validation Schema” was already introduced in 3.2 but the new “JSON Schema Validator” introduced in the 3.6 release is by far the best and a friendly way to manage validations in MongoDB.
- JSON Schema Validation extends Document Validation in many different ways, including the ability to enforce schemas inside arrays and prevent unapproved attributes from being added.
- To specify document validation on a new collection, use the new validator option in the db.createCollection() method.
- To add document validation to an existing collection, use the new validator option in db.runCommand() method
- To view the validation specifications for a collection, use the db.getCollectionInfos() method.

Json Schema Validation

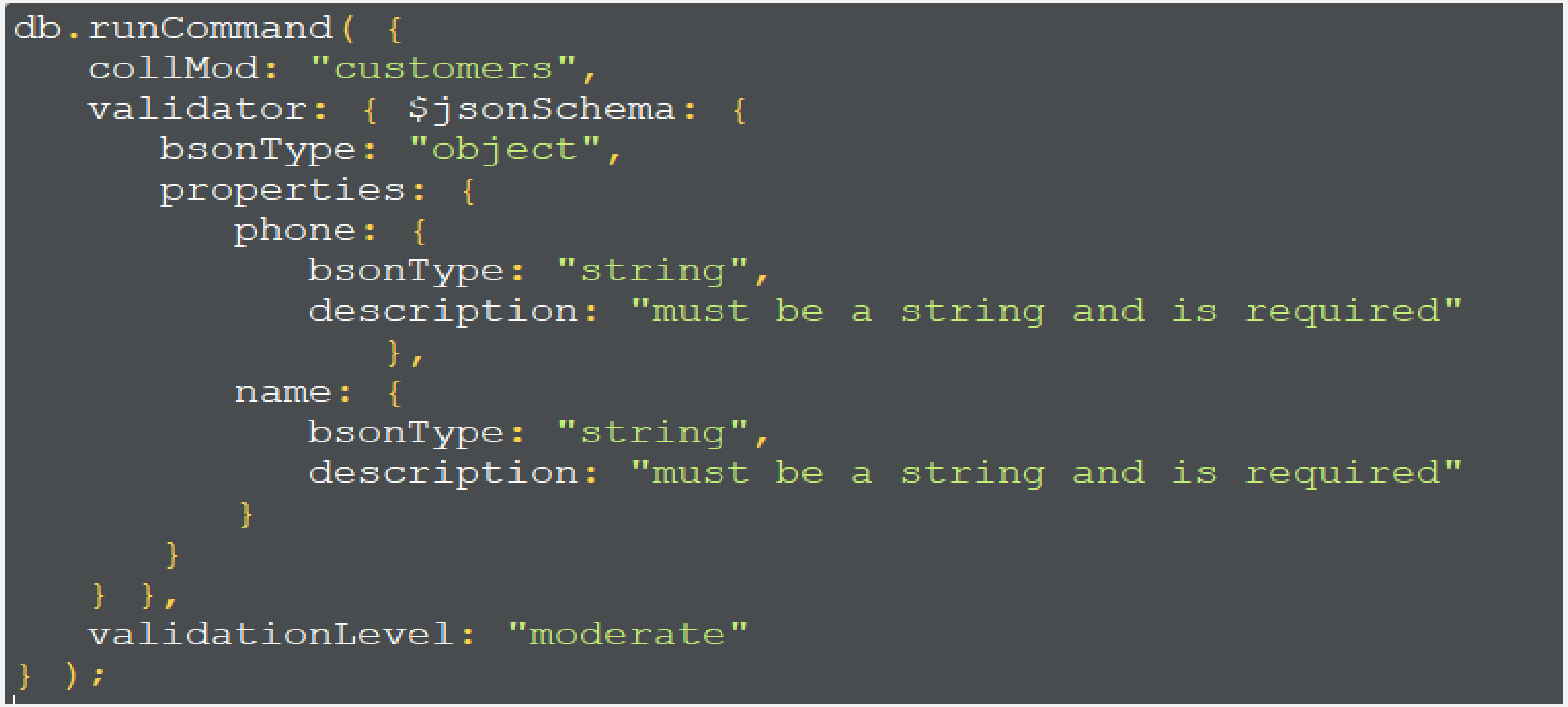
To specify certain fields to be exists and can have extra filed



```
db.runCommand( {
   collMod: "customers",
   validator: { $jsonSchema: {
      bsonType: "object",
      required: [ "phone", "name" ],
      properties: {
         phone: {
            bsonType: "string",
            description: "must be a string and is required"              },
         name: {
            bsonType: "string",
            description: "must be a string and is required"
         }
      }
   } },
   validationLevel: "moderate"
} );
```

Json Schema Validation

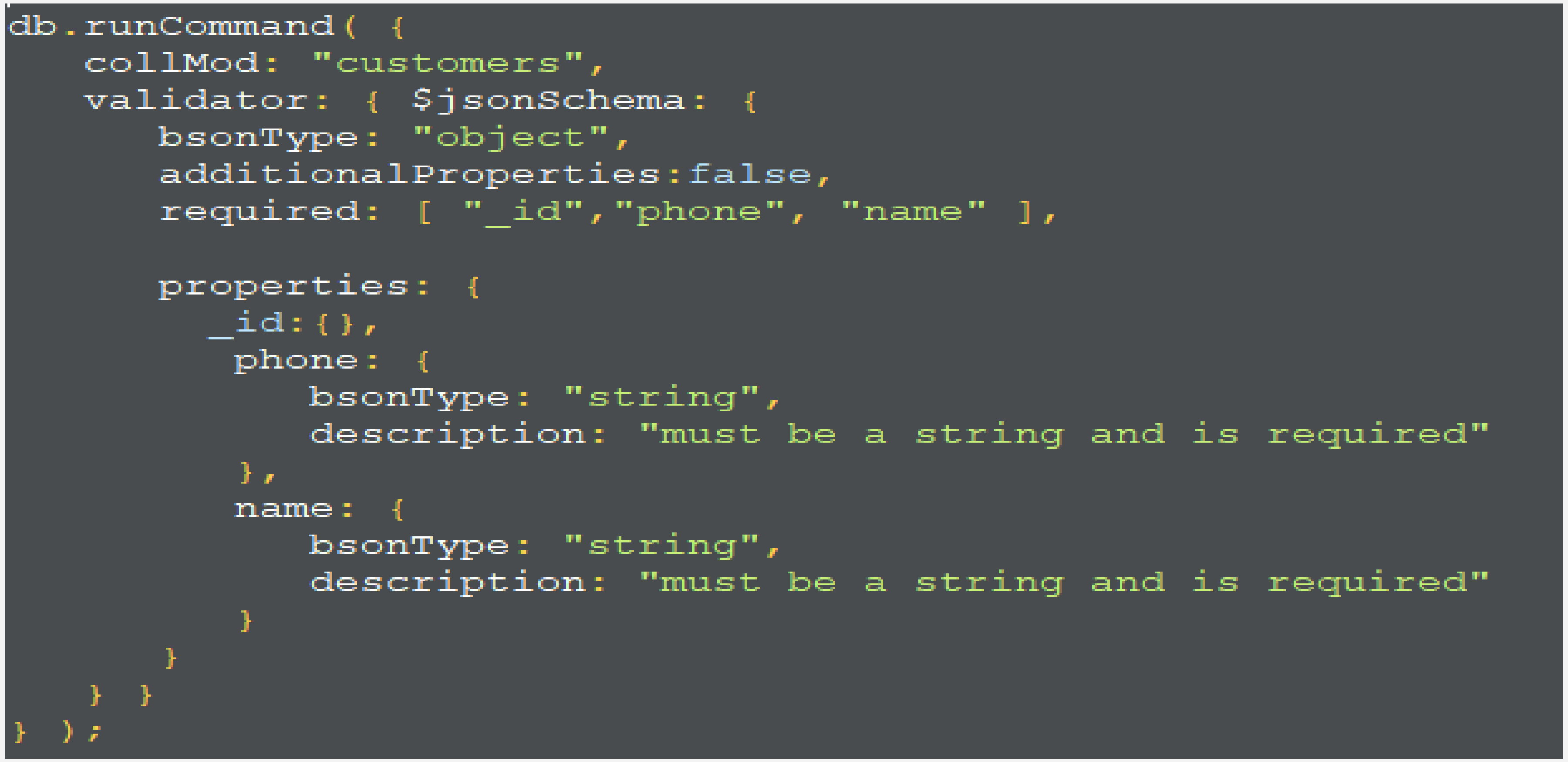
To specify all files needed as optional and can have extra filed



```
db.runCommand( {
   collMod: "customers",
   validator: { $jsonSchema: {
      bsonType: "object",
      properties: {
         phone: {
            bsonType: "string",
            description: "must be a string and is required"
               },
         name: {
            bsonType: "string",
            description: "must be a string and is required"
         }
      }
   } },
   validationLevel: "moderate"
} );
```

Json Schema Validation

To specify certain fields (additionalProperties) without adding additional ones
But _id must be specified

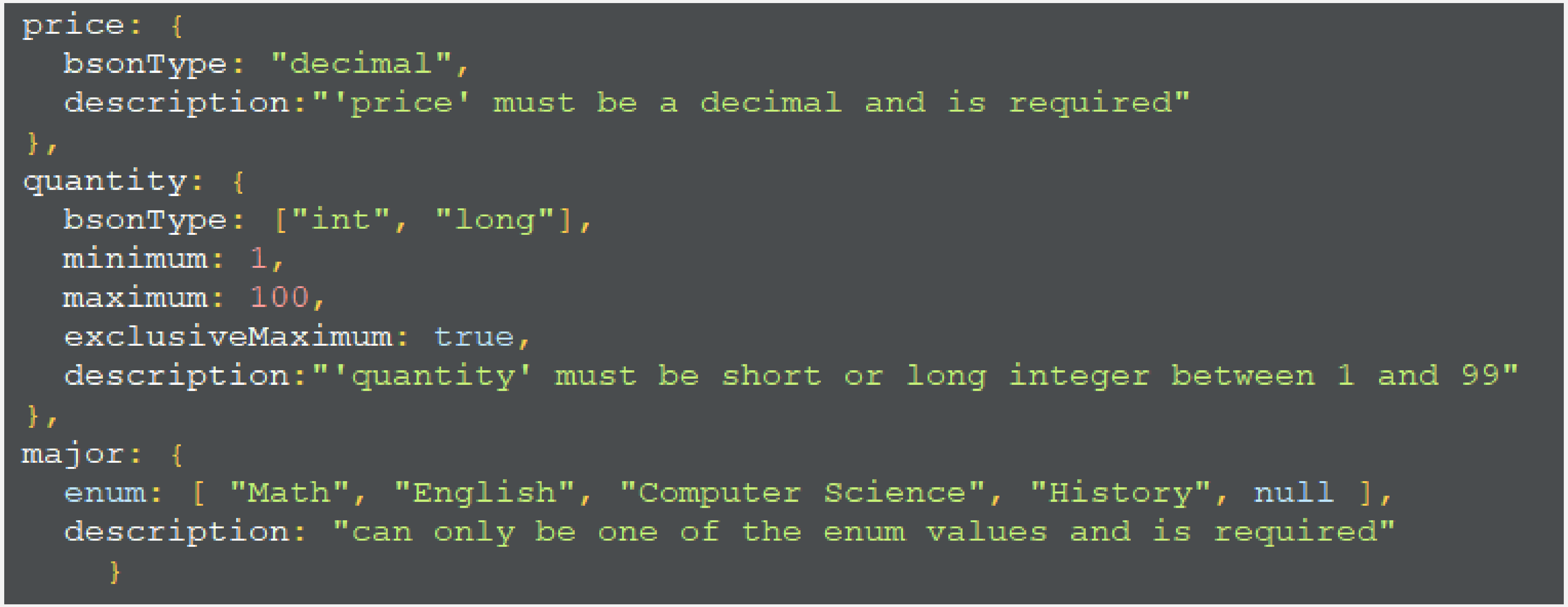


```
db.runCommand( {
   collMod: "customers",
   validator: { $jsonSchema: {
      bsonType: "object",
      additionalProperties:false,
      required: [ "_id","phone", "name" ],

      properties: {
        _id:{},
         phone: {
            bsonType: "string",
            description: "must be a string and is required"
         },
         name: {
            bsonType: "string",
            description: "must be a string and is required"
         }
      }
   } }
} );
```

Json Schema Validation

Specifying more data types



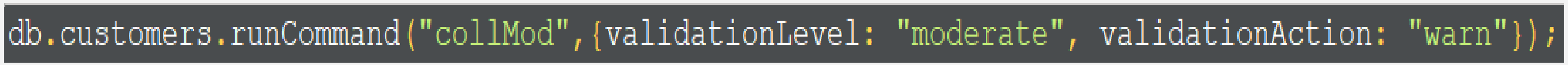
```
price: {
  bsonType: "decimal",
  description:"'price' must be a decimal and is required"
},
quantity: {
  bsonType: ["int", "long"],
  minimum: 1,
  maximum: 100,
  exclusiveMaximum: true,
  description:"'quantity' must be short or long integer between 1 and 99"
},
major: {
  enum: [ "Math", "English", "Computer Science", "History", null ],
  description: "can only be one of the enum values and is required"
    }
```

https://docs.mongodb.com/manual/reference/bson-types/

Document Validation

validationAction : Determines whether to error on invalid documents or just warn about the violations but allow invalid documents to be inserted

.

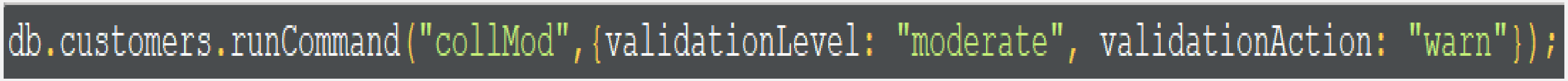


```
db.customers.runCommand("collMod",{validationLevel: "moderate", validationAction: "warn"});
```

validationAction	**Description**
"error"	Default Documents must pass validation before the write occurs. Otherwise, the write operation fails.
"warn"	Documents do not have to pass validation. If the document fails validation, the write operation logs the validation failure.

Document Validation

validationLevel : Determines how strictly MongoDB applies the validation rules to existing documents during an update.



```
db.customers.runCommand("collMod",{validationLevel: "moderate", validationAction: "warn"});
```

validationLevel	**Description**
"off"	No validation for inserts or updates.
"strict"	Default Apply validation rules to all inserts and all updates.
"moderate"	Apply validation rules to inserts and to updates on existing valid documents. Do not apply rules to updates on existing invalid documents.