Quick Start

On this page

Prerequisites

- A running MongoDB instance on localhost using the default port, 27017.
- The Ruby MongoDB driver. See installation for instructions on how to install the MongoDB driver.
- The following statement at the top of your code:

```
require 'mongo'
```

Make a Connection

Use `Mongo::Client` to establish a connection to a running MongoDB instance.

```
client = Mongo::Client.new([ '127.0.0.1:27017' ], :database => 'test')
```

You can also use a URI connection string:

```
client = Mongo::Client.new('mongodb://127.0.0.1:27017/test')
```

SEE ALSO:

Connect to a replica set, Connect to a sharded cluster, Client options

Access a Database and a Collection

The following examples demonstrate how to access a particular database and show its collections:

```
client = Mongo::Client.new([ '127.0.0.1:27017' ], :database => 'test')
db = client.database

db.collections # returns a list of collection objects
db.collection_names # returns a list of collection names
```

To access a collection, refer to it by name.

```
collection = client[:restaurants]
```

If the collection does not exist, the server will create it the first time you put data into it.

Insert a Document

To insert a single document into a collection, use the `insert_one` method.

```
client = Mongo::Client.new('mongodb://127.0.0.1:27017/test')

collection = client[:people]

doc = { name: 'Steve', hobbies: [ 'hiking', 'tennis', 'fly fishing' ] }

result = collection.insert_one(doc)
result.n # returns 1, because one document was inserted
```

To insert multiple documents into a collection, use the `insert_many` method.

```
docs = [ { _id: 1, name: 'Steve', hobbies: [ 'hiking', 'tennis', 'fly fishing' ] },
         { _id: 2, name: 'Sally', hobbies: ['skiing', 'stamp collecting' ] } ]

result = collection.insert_many(docs)
result.inserted_count # returns 2 because two documents were inserted
```

Query the Collection

Use the `find` method to create collection queries.

An empty query filter returns all documents in the collection.

```
client = Mongo::Client.new('mongodb://127.0.0.1:27017/test')
collection = client[:people]

collection.find.each do |document|
  #=> Yields a BSON::Document.
end
```

Use a query filter to find only matching documents.

```
client = Mongo::Client.new('mongodb://127.0.0.1:27017/test')
collection = client[:people]

puts collection.find( { name: 'Sally' } ).first
```

The example should print the following:

```
{"_id" => 2, "name" => "Sally", "hobbies" => ["skiing", "stamp collecting"]}
```

SEE ALSO:
Query Options, Read Preference

Update Documents

There are several update methods, including `update_one` and `update_many`. `update_one` updates a single document, while `update_many` updates multiple documents at once.

Both methods take as arguments a query filter document and a second document with the update data. Use `$set` to add or update a particular field or fields. Without `$set`, the entire existing document is replaced with the update data.

```
client = Mongo::Client.new('mongodb://127.0.0.1:27017/test')
collection = client[:people]

result = collection.update_one( { 'name' => 'Sally' }, { '$set' => { 'phone_number' => "555-5

puts collection.find( { 'name' => 'Sally' } ).first
```

The example should print the following:

```
{"_id" => 2, "name" => "Sally", "hobbies" => ["skiing", "stamp collecting"], "phone_number" =
```

The following example uses `update_many` with a blank query filter to update all the documents in the collection.

```
client = Mongo::Client.new('mongodb://127.0.0.1:27017/test')
collection = client[:people]

result = collection.update_many( {}, { '$set' => { 'age' => 36 } } )

puts result.modified_count # returns 2 because 2 documents were updated
```

SEE ALSO:

Other update options

Delete Documents

Use the `delete_one` or `delete_many` methods to delete documents from a collection (either singly or several at once).

```
client = Mongo::Client.new('mongodb://127.0.0.1:27017/test')
collection = client[:people]

result = collection.delete_one( { name: 'Steve' } )

puts result.deleted_count # returns 1 because one document was deleted
```

The following example inserts two more records into the collection, then deletes all the documents with a `name` field which matches a regular expression to find a string which begins with “S”.

```
client = Mongo::Client.new('mongodb://127.0.0.1:27017/test')
collection = client[:people]

collection.insert_many([ { _id: 3, name: "Arnold" }, { _id: 4, name: "Susan" } ])

puts collection.count # counts all documents in collection

result = collection.delete_many({ name: /$S*/ })

puts result.deleted_count # returns the number of documents deleted
```

Create Indexes

Use the `create_one` or `create_many` methods to create indexes singly or several at once.

```
client = Mongo::Client.new('mongodb://127.0.0.1:27017/test')
collection = client[:people]

collection.indexes.create_one({ name: 1 }, unique: true)
```

Use the `create_many` method to create several indexes with one statement. Note that when using `create_many`, the syntax is different from `create_one`.

```
client = Mongo::Client.new('mongodb://127.0.0.1:27017/test')
collection = client[:people]

collection.indexes.create_many([
    { key: { name: 1 } , unique: true },
    { key:  { hobbies: 1 } },
  ])
```

SEE ALSO:

Index options

Complete Sample App

A sample app using the Ruby driver for several common use cases is available for download from GitHub.