Creating, Inserting and Finding in MongoDB
Start up a new database by switching to it.

NOTE: The db does not exist until you create a collection:

use lessondb
Show the current db by running db:
db
Insert data into the lessondb database with this command.

NOTE: This will create the collection automatically.

ALSO, TAKE NOTE: the contents of the insert are basically a JS object, and include an array:

db.places.insert({"continent": "Africa", "country":"Morocco", "majorcities": ["Casablanca", "Fez", "Marrakech"]})

As a class, come up with 3-5 more countries and insert them into the db using the same syntax as above.

Observe where the data was entered in the MongoDB instance (in mongod).

Find all data in a Collection with db.[COLLECTION_NAME].find().

NOTE: the MongoDB _id was created automatically.

This id is specific for each doc in the collection:

db.places.find()

Adding .pretty() makes the data more readable:
db.places.find().pretty()

Find specific data by matching a field:
db.places.find({"continent": "Africa"})
db.places.find({"country": "Morocco"})

Try a few queries with the examples we came up with as a class.

Also, pick something that will find more than one entry so we can see how it will return all matches.

Find specific data by matching an _id:

db.places.find({_id:[COPY AN OBJECTID FROM THE PREVIOUS FIND RESULTS]})

Example: db.places.find({_id: ObjectId("5416fe1d94bcf86cd785439036")})

MongoDB Create/Insert/Find

```
C:\Users\const\Documents\UNCC BootCamp\Class\Week 17\17-NoSQL\01-Activities>mongo
MongoDB shell version v5.0.2
connecting to: mongodb://127.0.0.1:27017/?compressors=disabled&gssapiServiceName=mongodb
Implicit session: session { "id" : UUID("cded9b1b-2f81-4734-9611-93d275762a84") }
MongoDB server version: 5.0.2
================
Warning: the "mongo" shell has been superseded by "mongosh",
which delivers improved usability and compatibility.The "mongo" shell has been deprecated and will be removed in
an upcoming release.
We recommend you begin using "mongosh".
For installation instructions, see
https://docs.mongodb.com/mongodb-shell/install/
================
---
The server generated these startup warnings when booting:
        2021-08-14T10:14:59.618-04:00: Access control is not enabled for the database. Read and write access to data
 and configuration is unrestricted
---
---
        Enable MongoDB's free cloud-based monitoring service, which will then receive and display
        metrics about your deployment (disk utilization, CPU, operation statistics, etc).

        The monitoring data will be available on a MongoDB website with a unique URL accessible to you
        and anyone you share the URL with. MongoDB may use this information to make product
        improvements and to suggest MongoDB products and deployment options to you.

        To enable free monitoring, run the following command: db.enableFreeMonitoring()
        To permanently disable this reminder, run the following command: db.disableFreeMonitoring()
---
>
```

```
> use exampledb
switched to db exampledb
> db
exampledb
> db.places.insert({"country": "USA", "state": "North Carolina", "majorcities": ["Charlotte",
"Wilmington", "Asheville"]})
WriteResult({ "nInserted" : 1 })
> db.places.find()
{ "_id" : ObjectId("612cbcacd1ce45449d51fe83"), "country" : "USA", "state" : "North Carolina",
 "majorcities" : [ "Charlotte", "Wilmington", "Asheville" ] }
> db.places.find().pretty()
{
        "_id" : ObjectId("612cbcacd1ce45449d51fe83"),
        "country" : "USA",
        "state" : "North Carolina",
        "majorcities" : [
                "Charlotte",
                "Wilmington",
                "Asheville"
        ]
}
```

Updating, Deleting and Dropping in MongoDB

Make sure you are using the database, lessondb, that we created earlier.

db
use lessondb

Updating
We update data using db.[COLLECTION_NAME].update()
db.places.update({"country": "Morocco"}, {$set: {"continent": "Antarctica"}})

Note that the above will only update the first entry it matches.

To update multiple entries, you need to add {multi: true}

db.places.update({"country": "Morocco"}, {$set: {"continent": "Antarctica"}}, {multi: true})

Recall from the earlier demo the structure of our document:
db.places.insert({"continent": "Africa", "country": "Morocco", "majorcities": ["Casablanca", "Fez", "Marrakech"]})

What do you think will happen when you run the following command, even though there is not a capital field in the document?
db.places.update({"country": "Morocco"}, {$set: {"capital": "Rabat"}})

Answer: $set will create the field capital

The newly created field can now be updated with the same command:

db.places.update({"country": "Morocco"}, {$set: {"capital": "RABAT"}})

We can update the values in an array with $push:
db.places.update({"country": "Morocco"}, {$push: {"majorcities": "Agadir"}})

Deleting
We delete an entry with db.[COLLECTION_NAME].remove()
db.places.remove({"country": "Morocco"})

We can also empty a collection with db.[COLLECTION_NAME].remove()
db.places.remove({})

Dropping
We drop a collection with db.[COLLECTION_NAME].drop()
db.places.drop()

To drop a database:
db.dropDatabase()

MongoDB Update/Delete

```
> db.places.update({"state": "North Carolina"}, {$set: {"country": "England"}})
WriteResult({ "nMatched" : 1, "nUpserted" : 0, "nModified" : 1 })
> db.places.find().pretty()
{
        "_id" : ObjectId("612cbcacd1ce45449d51fe83"),
        "country" : "England",
        "state" : "North Carolina",
        "majorcities" : [
                "Charlotte",
                "Wilmington",
                "Asheville"
        ]
}
> db.places.update({"state": "North Carolina"}, {$push: {"majorcities": "Greensboro"}})
WriteResult({ "nMatched" : 1, "nUpserted" : 0, "nModified" : 1 })
> db.places.find().pretty()
{
        "_id" : ObjectId("612cbcacd1ce45449d51fe83"),
        "country" : "England",
        "state" : "North Carolina",
        "majorcities" : [
                "Charlotte",
                "Wilmington",
                "Asheville",
                "Greensboro"
        ]
}
```

Sorting in MongoDB
The format of a sort follows db.COLLECTION_NAME.find().sort ({FIELD:1})

A value of 1 is for ascending order.

A value of -1 is for descending order.

NOTE: Remember to add .pretty() afterwards so the results are readable!

Create a new db named zoo and insert at least 5 animals with the following attributes:
numLegs - a integer that points to the number of legs.

class - a string that points to the animal's class ("reptile", "mammal" etc).

weight - a integer that points to the animals weight.

name - a string that points to the animal's name.

Example:
```
{
  "name": "Panda",
  "numLegs": 4,
  "class": "mammal",
  "weight": 254
}
```

Using the animals collection that you created:

Sort by id:
The id contains a timestamp, so sorting by id will sort by when they were entered to the database.
```
db.animals.find().sort({ _id:1 });
db.animals.find().sort({ _id:-1 });
```

Sort by an integer - numLegs:
```
db.animals.find().sort({ numLegs:1 });
db.animals.find().sort({ numLegs:-1 });
```

Sort by a string - class:
```
db.animals.find().sort({ class:1 });
db.animals.find().sort({ class:-1 });
```

MongoDB Sort

```
const express = require("express");
const mongojs = require("mongojs");

const app = express();

const databaseUrl = "zoo";
const collections = ["animals"];

const db = mongojs(databaseUrl, collections);

db.on("error", (error) => {
  console.log("Database Error:", error);
});

app.get("/", (req, res) => {
  res.send("Hello world");
});

app.get("/all", (req, res) => {
  db.animals.find({}, (err, data) => {
    if (err) {
      console.log(err);
    } else {
      res.json(data);
    }
  });
});

// 1: Name: Send JSON response sorted by name in ascending order, e.g. GET "/name"
app.get("/name", (req, res) => {
  db.animals.find().sort({ name: 1 }, (err, found) => {
    if (err) {
      console.log(err);
    } else {
      res.json(found);
    }
  });
});
// 2: Weight: Send JSON response sorted by weight in descending order, , e.g. GET "/weight"
app.get("/weight", (req, res) => {
  db.animals.find().sort({ weight: -1 }, (err, found) => {
    if (err) {
      console.log(err);
    } else {
      res.json(found);
    }
  });
});
// Set the app to listen on port 3000
app.listen(3000, () => {
  console.log("App running on port 3000!");
});
```

Mongo CRUD pt 1

```
const actionBtn = document.getElementById("action-button");
// new item
const makeNote = document.getElementById("make-new");
// clear all items
const clear = document.getElementById("clear-all");
// delete an item
const results = document.getElementById("results");
const status = document.getElementById("status");

function getResults() {
  // Empty any results currently on the page
  $("#results").empty();
  // Grab all of the current notes
  $.getJSON("/all", function(data) {
    // For each note...
    for (var i = 0; i < data.length; i++) {
      // ...populate #results with a p-tag that includes
              the note's title and object id
      $("#results").prepend("<p class='data-entry' data-id=" +
              data[i]._id + "><span class='dataTitle' data-id=" +
        data[i]._id + ">" + data[i].title + "</span><span class='delete'>
              X</span></p>");
    }
  });
}

// Runs the getResults function as soon as the script is executed
getResults();

// When the #make-new button is clicked
$(document).on("click", "#make-new", function() {
  // AJAX POST call to the submit route on the server
  // This will take the data from the form and send it to the server
  $.ajax({
    type: "POST",
    dataType: "json",
    url: "/submit",
    data: {
      title: $("#title").val(),
      note: $("#note").val(),
      created: Date.now()
    }
  })
  // If that API call succeeds, add the title and a delete button
              for the note to the page
    .then(function(data) {
    // Add the title and delete button to the #results section
      $("#results").prepend("<p class='data-entry' data-id=" +
              data._id + "><span class='dataTitle' data-id=" +
      data._id + ">" + data.title + "</span><span class='delete'>
              X</span></p>");
      // Clear the note and title inputs on the page
      $("#note").val("");
      $("#title").val("");
    });
});

// When the #clear-all button is pressed
$("#clear-all").on("click", function() {
  // Make an AJAX GET request to delete the notes from the db
  $.ajax({
    type: "DELETE",
    dataType: "json",
    url: "/clearall",
    // On a successful call, clear the #results section
    success: function(response) {
      $("#results").empty();
    }
  });
});

// When user clicks the delete button for a note
$(document).on("click", ".delete", function() {
  // Save the p tag that encloses the button
  var selected = $(this).parent();
  // Make an AJAX GET request to delete the specific note
  // this uses the data-id of the p-tag, which is linked to the specific note
  $.ajax({
    type: "DELETE",
    url: "/delete/" + selected.attr("data-id"),
    // On successful call
    success: function(response) {
      // Remove the p-tag from the DOM
      selected.remove();
      // Clear the note and title inputs
      $("#note").val("");
      $("#title").val("");
      // Make sure the #action-button is submit (in case it's update)
      $("#action-button").html("<button id='make-new'>Submit</
button>");
    }
  });
});
// When user click's on note title, show the note, and allow for updates
$(document).on("click", ".dataTitle", function() {
  // Grab the element
  var selected = $(this);
  // Make an ajax call to find the note
  // This uses the data-id of the p-tag, which is linked to the specific note
  $.ajax({
    type: "GET",
    url: "/find/" + selected.attr("data-id"),
    success: function(data) {
      // Fill the inputs with the data that the ajax call collected
      $("#note").val(data.note);
      $("#title").val(data.title);
      // Make the #action-button an update button, so user can
      // Update the note s/he chooses
      $("#action-button").html("<button id='updater' data-id='" +
              data._id + "'>Update</button>");
    }
  });
});

// When user click's update button, update the specific note
$(document).on("click", "#updater", function() {
  // Save the selected element
  var selected = $(this);
  // Make an AJAX POST request
  // This uses the data-id of the update button,
  // which is linked to the specific note title
  // that the user clicked before
  $.ajax({
    type: "POST",
    url: "/update/" + selected.attr("data-id"),
    dataType: "json",
    data: {
      title: $("#title").val(),
      note: $("#note").val()
    },
    // On successful call
    success: function(data) {
      // Clear the inputs
      $("#note").val("");
      $("#title").val("");
      // Revert action button to submit
      $("#action-button").html("<button id='make-new'>
              Submit</button>");
      // Grab the results from the db again, to populate the DOM
      getResults();
    }
  });
});
```

Mongo CRUD pt 2

```
const express = require("express");
const mongojs = require("mongojs");
const logger = require("morgan");

const app = express();

app.use(logger("dev"));

app.use(express.urlencoded({ extended: true }));
app.use(express.json());

app.use(express.static("public"));

const databaseUrl = "notetaker";
const collections = ["notes"];

const db = mongojs(databaseUrl, collections);

db.on("error", (error) => {
  console.log("Database Error:", error);
});

app.get("/", (req, res) => {
  res.sendFile(path.join(__dirname + "./public/index.html"));
});

// 1. Save a note to the database's collection
// POST: /submit
app.post("/submit", (req, res) => {
  console.log(req.body);
  db.notes.insert(req.body, (error, data) => {
    if (error) {
      res.send(error);
    } else {
      res.send(data);
    }
  });
});

// 2. Retrieve all notes from the database's collection
// GET: /all
app.get("/all", (req, res) => {
  db.notes.find({}, (error, data) => {
    if (error) {
      res.send(error);
    } else {
      res.json(data);
    }
  });
});

// 3. Retrieve one note in the database's collection by it's ObjectId
// (remember, mongojs.ObjectId(IdYouWantToFind)
// GET: /find/:id
app.get("/find/:id", (req, res) => {
  db.notes.findOne(
    {
      _id: mongojs.ObjectId(req.params.id)
    },
    (error, data) => {
      if (error) {
        res.send(error);
      } else {
        res.send(data);
      }
    }
  );
});

// 4. Update one note in the database's collection by it's ObjectId
// (remember, mongojs.ObjectId(IdYouWantToFind)
// POST: /update/:id
app.post("/update/:id", (req, res) => {
  db.notes.update(
    {
      _id: mongojs.ObjectId(req.params.id)
    },
    {
      $set: {
        title: req.body.title,
        note: req.body.note,
        modified: Date.now()
      }
    },
    (error, data) => {
      if (error) {
        res.send(error);
      } else {
        res.send(data);
      }
    }
  );
});

// 5. Delete one note from the database's collection by it's ObjectId
// (remember, mongojs.ObjectId(IdYouWantToFind)
// DELETE: /delete/:id
app.delete("/delete/:id", (req, res) => {
  db.notes.remove(
    {
      _id: mongojs.ObjectID(req.params.id)
    },
    (error, data) => {
      if (error) {
        res.send(error);
      } else {
        res.send(data);
      }
    }
  );
});

// 6. Clear the entire note collection
// DELETE: /clearall
app.delete("/clearall", (req, res) => {
  db.notes.remove({}, (error, response) => {
    if (error) {
      res.send(error);
    } else {
      res.send(response);
    }
  });
});

// Listen on port 3000
app.listen(3000, () => {
  console.log("App running on port 3000!");
});
```

```
<!DOCTYPE html>
<html lang="en-us">
<head>
  <meta charset="UTF-8">
  <title>NoteTaker</title>
  <script src="https://cdnjs.cloudflare.com/ajax/libs/jquery/3.2.1/jquery.min.js"></script>
  <style media="screen">
    body {
      text-align: center;
      background-color: cornflowerblue;
    }

    .primary {
      color: #64edbd;
    }

    h1 {
      font-size: 40px;
      color: white;
    }

    h2, h4 {
      color: white;
    }

    .delete {
      margin-left: 20px;
      color: red;
      cursor: pointer;
      text-align: right;
    }

    input {
      width: 400px;
      height: 50px;
      margin-left: auto;
      margin-right: auto;
      font-size: 42px;
      background-color: antiquewhite;
    }

    textarea {
      width: 400px;
      height: 400px;
      margin-left: auto;
      margin-right: auto;
      font-size: 24px;
      background-color: antiquewhite;
    }

    .data-title {
      cursor: pointer;
      font-size: 24px;
    }

    #results {
      width: 400px;
      height: 200px;
      overflow: auto;
      margin-left: auto;
      margin-right: auto;
      background-color: antiquewhite;
    }

    #action-button {
      display: inline-block;
    }

    #buttons {
      margin: 20px;
    }
  </style>
</head>
<body>
  <h1>Mongo Note App</h1>
  <h2>Status: <span id="status" class="primary">Creating</span></h2>
  <h4>Create a new note by submitting its title and content below</h4>
  <h4>Click on a note to edit its title or content</h4>
  <div id="user-input">
    <p>Title</p>
    <input type="text" id="title" />
    <br>
    <p>Note</p>
    <textarea id="note"></textarea>
    <div id="buttons">
      <div id="action-button">
        <button id="make-new">Submit</button>
      </div>
      <button id="clear-all">Delete All Notes</button>
    </div>
  </div>
  <div id="results"></div>
  <script src="app.js"></script>
</body>
</html>
```

MongoJS Review pt.1

```
const express = require("express");
const mongojs = require("mongojs");
const logger = require("morgan");

const app = express();
app.use(logger("dev"));

app.use(express.urlencoded({ extended: true }));
app.use(express.json());

app.use(express.static("public"));

const databaseUrl = "warmup";
const collections = ["books"];

const db = mongojs(databaseUrl, collections);

db.on("error", (error) => {
  console.log("Database Error:", error);
});

// Routes
// ======

// Post a book to the mongoose database
app.post("/submit", ({ body }, res) => {
  // Save the request body as an object called book
  const book = body;
  book.read = false;
  db.books.insert(book, (err, data) => {
    res.json(data);
  });
  db.books.save(book, (error, saved) => {
    if (error) {
      console.log(error);
    } else {
      res.send(saved);
    }
  });
});

// Find all books marked as read
app.get("/read", (req, res) => {
  db.books.find({ read: true }, (error, found) => {
    if (error) {
      console.log(error);
    } else {
      res.json(found);
    }
  });
});

// Find all books marked as unread
app.get("/unread", (req, res) => {
  db.books.find({ read: false }, (error, found) => {
    if (error) {
      console.log(error);
    } else {
      res.json(found);
    }
  });
});

// Mark a book as having been read
  // Remember: when searching by an id,
            the id needs to be passed in
  // as (mongojs.ObjectId(IdYouWantToFind))
app.put("/markread/:id", ({ params }, res) => {
  db.books.update(
    {
      _id: mongojs.ObjectId(params.id),
    },
    {
      $set: {
        read: true,
      },
    },
    (error, edited) => {
      if (error) {
        console.log(error);
        res.send(error);
      } else {
        console.log(edited);
        res.send(edited);
      }
    }
  );
});

// Mark a book as having been not read
app.put("/markunread/:id", ({ params }, res) => {
  db.books.update(
    {
      _id: mongojs.ObjectId(params.id),
    },
    {
      $set: {
        read: false,
      },
    },
    (error, edited) => {
      if (error) {
        console.log(error);
        res.send(error);
      } else {
        console.log(edited);
        res.send(edited);
      }
    }
  );
});

// Listen on port 3000
app.listen(3000, () => {
  console.log("App running on port 3000!");
});
```

MongoJS Review pt.2

```
// Click Events

// Click event to add a book to the db
$("#addbook").on("click", function() {
  $.ajax({
    type: "POST",
    url: "/submit",
    dataType: "json",
    data: {
      title: $("#title").val(),
      author: $("#author").val(),
      created: Date.now()
    }
  })
    .then(function(data) {
      console.log(data);
      getUnread();
      $("#author").val("");
      $("#title").val("");
    }
    );
  return false;
});

// Click event to mark a book as read
$(document).on("click", ".markread", function() {
  var thisId = $(this).attr("data-id");
  $.ajax({
    type: "PUT",
    url: "/markread/" + thisId
  });
  $(this).parents("tr").remove();
  getRead();
});

// Click event to mark a book as not read
$(document).on("click", ".markunread", function() {
  var thisId = $(this).attr("data-id");
  $.ajax({
    type: "PUT",
    url: "/markunread/" + thisId
  });
  $(this).parents("tr").remove();
  getUnread();
});
```

```
// Load unread books and render them to the screen
function getUnread() {
  $("#unread").empty();
  $.getJSON("/unread", function(data) {
    for (var i = 0; i < data.length; i++) {
      $("#unread").prepend("<tr><td>" + data[i].title +
            "</td><td>" + data[i].author +
        "</td><td><button class='markread' data-id='" +
            data[i]._id + "'>Mark Read</button>
            </td></tr>");
    }
    $("#unread").prepend("<tr><th>Title</th>
            <th>Author</th>
            <th>Read/Unread</th></tr>");
  });
}

// Load read books and render them to the screen
function getRead() {
  $("#read").empty();
  $.getJSON("/read", function(data) {
    for (var i = 0; i < data.length; i++) {
      $("#read").prepend("<tr><td>" + data[i].title +
            "</td><td>" + data[i].author +
        "</td><td><button class='markunread' data-id='" +
            data[i]._id + "'>Mark Unread</button>
            </td></tr>");
    }
    $("#read").prepend("<tr><th>Title</th>
            <th>Author</th>
            <th>Read/Unread</th></tr>");
  });
}

// Calling our functions
getUnread();
getRead();
```

```
<!DOCTYPE html>
<html lang="en-us">
  <head>
    <meta charset="UTF-8">
    <title>Book Tracker</title>
    <style>
      #unread, #read {
        width: 40%;
        background-color: antiquewhite;
      }
      #unread { float: left; }
      #read { float: right; }
      table, td {
        border-width: 1px;
        border-style: solid;
        border-color: black;
      }
      #wrapper {
        width: 960px;
        padding-top: 10%;
        margin-left: auto;
        margin-right: auto;
        text-align: center;
      }
      body {
        font-family: sans-serif;
        background-color: #C0FFEE;
      }
      #results { margin-top: 50px; }
      #userinput { font-size: 24px; }
      input {
        width: 50%;
        height: 36px;
        font-size: 24px;
      }
    </style>
  </head>
  <body>
    <div id="wrapper">
      <h1>Book Tracker 4000</h1>
      <h2><i>The latest and greatest in book tracking technology!</i></h2>
      <form id="userinput">
        <input type="text" id="title" placeholder="Book Title"/>
        <br />
        <input type="text" id="author" placeholder="Book Author"/>
        <br />
        <button type="submit" id="addbook">Add New</button>
      </form>
      <div id="results">
        <table id="unread">
          <tr><th>Title</th><th>Author</th><th>Read/Unread</th></tr>
        </table>
        <table id="read">
          <tr><th>Title</th><th>Author</th><th>Read/Unread</th></tr>
        </table>
      </div>
    </div>
    <script src="https://cdnjs.cloudflare.com/ajax/libs/jquery/3.2.1/jquery.min.js"></script>
    <script src="app.js"></script>
  </body>
</html>
```

Code 1

title

Code 2

Code 3

Code 1

title

Code 2

Code 3

Code 1

title

Code 2

Code 3

Code 1

title

Code 2

Code 3

Code 1

title

Code 2

Code 3