Welcome to mongoDB

Mongo DB is a highly scalable, NoSQL, schema free data store.

- mongoDB is a document storage and retrieval engine

To begin using MongoDB locally on your **personal computer**, download and Install the MongoDB Community Edition from the following link:

https://www.mongodb.com/products/self-managed/community-edition

The above installation should also install the **MongoDB Compass**, the **GUI tool** for exploring data in MongoDB. If not installed by default, you can download and install it from the following link:

https://www.mongodb.com/products/tools/compass

You can explore MongoDB without a local installation by using the online platform, **MongoDB Atlas**. Simply sign up for MongoDB Atlas through the following link. **This online version will allow you to access data from anywhere**.

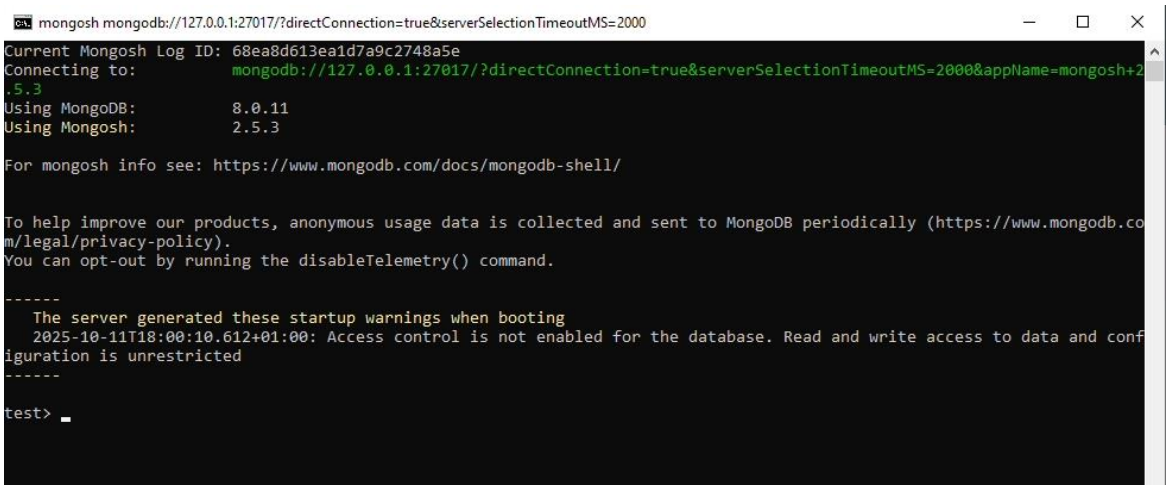
https://www.mongodb.com/products/platform

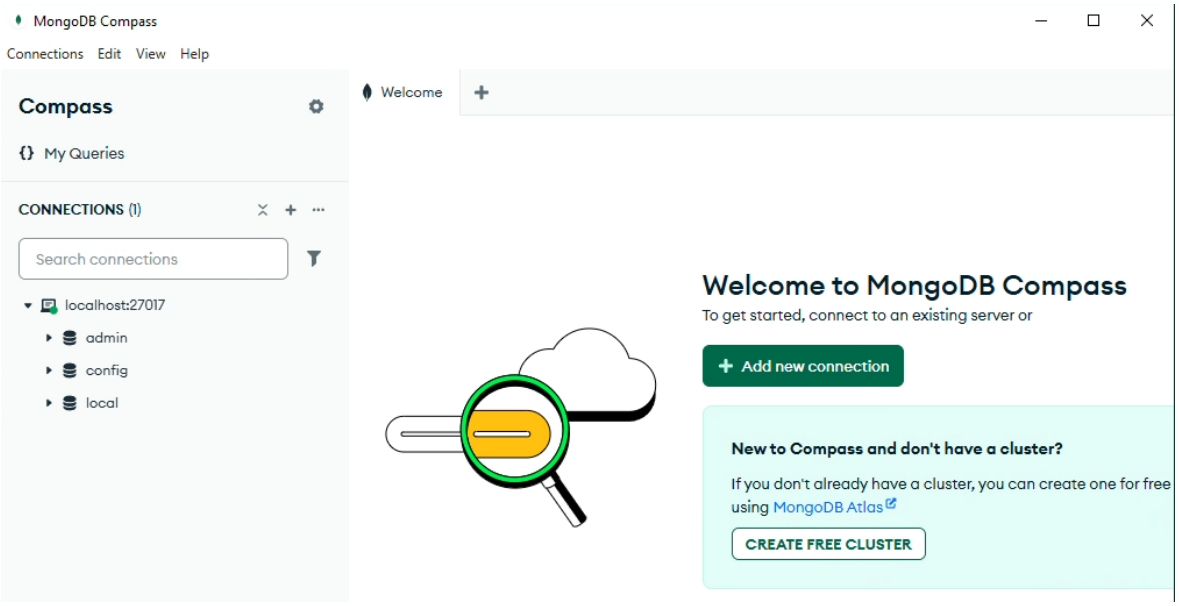
In this lab, you will be working with the pre-installed MongoDB on the Horizon platform. To begin, simply launch the Horizon platform and start the MongoDB software from there.

If you are unsure how to load and access MongoDB via Horizon, follow the

Step-by-Step Guide

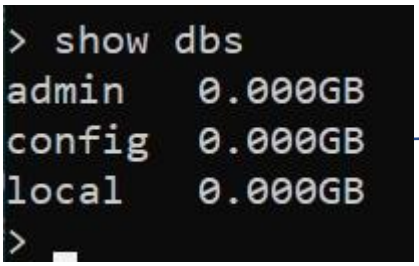
If you followed the process correctly, you should see the screens similar to the ones shown below. By default, MongoDB connects to the localhost. Once connected to a MongoDB instance, you can enter JavaScript directly into the console. This allows you to create databases, define variables, perform calculations, and work with JSON data.





To start experimenting with MongoDB, you'll first need to create a new database. However, before doing so, run the following command to view the existing databases:

```
show dbs
```



There are three databases: admin, config, and local

Creating a database using Mongo shell command

We can switch to a database in Mongo with the **use** command.

```
use Bookshop
```

This command switches to the 'Bookshop' database. If the database doesn't already exist, MongoDB will **automatically create it when you first insert a document into it.**

To check which database you're currently using, simply run the **db** command as shown below. If you'd like to delete the database, use the **dropDatabase()** method as demonstrated.

```
db
```

```
db.dropDatabase()
```

Exercise- Create a database

- Use the **use** command to create a new database called '**University**'

Collections

Collections in MongoDB are groups of related documents, **similar to tables** in a relational database. Your database can contain as many collections as needed. However, since MongoDB does not support joins, queries can only retrieve data from one collection at a time. This is an important consideration when planning how to structure your data.

You can create a collection using the **`createCollection()`** method. The following command will create a collection called **`Students`**.

```
db.createCollection('Students')
```

Similar to databases, collections in MongoDB are created automatically. If you insert a document into a collection that does not yet exist, MongoDB will create the collection for you.

View collections in a database using the show command, like this:

```
show collections
```

Documents

Documents in MongoDB are **JSON objects** stored within a collection, **akin to rows in a table**. They can take any valid JSON format, with the exception that they cannot contain functions. Additionally, documents can include embedded documents.

Documents have **a unique ID (the _id field)**

The size limit for a document is **16Mb** which is more than ample for most use cases.

Simple document

```
{
  _id : ObjectId("4cb4ab6d7addf98506010001"),
  name: "John Doe",
  age: 45
 }
```

Document with **Embedded documents**

```
{
 _id : "derickr",
 name: "Derick Rethans",
 articles:[
           {
            title : "Profiling PHP Applications",
             year : 2021
            },
           {
            title : "Xdebug",
            year : 2005
           }
        ]

     }
```

Creating a document

You can create a document by inserting it into a collection using command line

```
db.Students.insertOne({UoB:20213425, name: 'Alice Johnson', age:25})
```

Function structure

`insertOne(document)`

This command will insert only a single document. But what if you want to insert multiple documents at once?

```
db.Students.insertMany([{UoB:20203425, name: 'Bob Smith', age:29},
{UoB:20191425, name: 'Alice Brown', age:35}, {UoB:20182425, name:
'Mohammed Ali', age:28}])
```

Function structure

`insertMany([d1, d2,…,dn])`

This command will insert multiple documents at once!

Finding a document

You can find a document or documents matching a particular pattern using the **`find()`** method.

The **`find()`** method has the following form:

```
db.Students.find(<query>, <projection>, <options>)
```

`query`: is a document type **optional** parameter. It specifies selection filter using query operators. To return all documents in a collection, omit this parameter or pass an empty document ({}).

`projection`: is also a document type **optional** parameter. It specifies the fields to return in the documents that match the query filter. To return all fields in the matching documents, omit this parameter.

`options`: is a document type **optional** parameter. It specifies additional options for the query. These options modify query behaviour and how results are returned.

To retrieve all the documents (similar to rows in a table) from the **`Students`** collection, you can easily do so using the **`find()`** method, as shown below.

```
db.Students.find()
```

Can you identify any similarities with a SQL query? Doesn't it resemble this one?

```
SELECT *
FROM Students;
```

Results returned by the **`find()`** method can be sorted

```
db.Students.find().sort({name:1})
```

Sorted based on **name**
1: in alphabetical order
-1: in reverse alphabetical order

```
db.Students.find().sort({age:1})
```

Sorted based on **age**
1: in ascending order
-1: in descending order

We can also limit the information returned by the **`find()`** method by using the **`limit()`** method.

```
db.Students.find().limit(2)
```

Will return 2 documents

We can use the **`sort()`** and the **`limit()`** methods together

```
db.Students.find().sort(age:1).limit(3)
```

You might not always want to retrieve all documents; instead, you may want specific documents based on certain conditions. In the `find()` method, you can add arguments (i.e., define conditions) as query parameters to filter the results accordingly.

```
db.Students.find({name:'Alice Johnson'})
```

Doesn't it resemble this one?

```
SELECT *
FROM Students
WHERE name='Alice Johnson';
```

Similarly, a condition is set based on age

```
db.Students.find({age:35})
```

It is possible to set multiple conditions

```
db.Students.find({name:'John Doe', age:35})
```

We can also find using expressions. We define these using JSON, like this:

```
db.Students.find({age:{$gt: 28}})
```

We can use operators like this:

- $gt - Greater than
- $lt - Less than
- $exists - The field exists

See the full list here:

https://www.mongodb.com/docs/manual/reference/mql/query-predicates/#std-label-query-selector

The **projection** parameter in the `find()` method can be used to specify which fields to include or exclude in the returned documents.

```
db.Students.find({},{name:true})
```

Will return only the name fields of all documents

You may notice that the `_id` field is also displayed alongside the name field. If you don't need the `_id` field, you can exclude it by setting it to false, as shown below.

```
db.Students.find({},{_id:false, name:true})
```

Update Documents

You can update a single document using the **updateOne()** method. The basic syntax of this method is as follows:

```
db.Students.updateOne(<filter>,<update>, <options>)
```

filter: selection criteria for an update

update: the change you want to make

```
db.Students.updateOne(
{_id:ObjectId("642c0fde134b27a2efa35a54")},
{$set:{age: 35}}
)
```

If you want to remove a specific field rather than updating its value, you can use **`$unset`** within the **`updateOne()`** method.

The **`updateMany()`** method allows you to update multiple documents simultaneously.

```
db.Students.updateMany({},{$set:{course: 'CS'}}
```

The above command will update all student records by setting the `course` field to 'CS'. If the `course` field doesn't already exist, it will be created.

Suppose there is a field named **`score`** representing a student's mark in a module. You want to add a new field called **`grade`** and set it to **`Fail`** for all students whose score is below 40%. You can do this using the **`updateMany()`** method as shown below.

```
db.Students.updateMany({score:{$lt:40}},{$set:{grade:
'Fail'}})
```

The **`deleteOne()`** method deletes the first document that matches the specified filter. In the example below, it will remove the first document where the `name` field is 'John Doe'

```
db.Students.deleteOne({name:'John Doe'})
```

The **`deleteMany()`** method removes all documents that match the specified filter. For example, if you want to delete all students who have a score below 40, you can use the following command:

```
db.Students.deleteMany({score:{$lt:40}})
```

Exercise: If you haven't created the Students collection yet, please do so within your University Database.

Next, convert the following table into MongoDB documents and insert them into the Students collection.

UoB	Name	Age	Course	House No	Street	Town	Hobbies
123456	Alice Johnson	22	Computer Science	45	High Street	Birmingham	Traveling
876543	Bob Smith	20	Mathematics	12B	Baker Street	London	Playing Chess, Painting
234567	Clara Lee	23	Physics	78	King Street	Manchester	Photography, Hiking
345678	David Brown	27	Engineering	14A	Queen Road	Leeds	Football, Gaming
456789	Emma Wilson	25	Business Management	34	Church Lane	Liverpool	Singing, Traveling
567890	Frank Adams	31	Economics	56	Bridge Street	Glasgow	Cooking, Reading
678901	Grace Thomas	19	Biology	22	Elm Road	Oxford	Swimming, Painting, Reading
789012	Henry Walker	32	Chemistry	9	Maple Avenue	Cambridge	Cycling, Writing
890123	Ivy Carter	27	Law	50	Park Lane	Leeds	Volunteering, Gardening
901234	Jack White	25	History	31	Oak Street	Manchester	Running, Cooking

Now, please answer the following questions by writing the appropriate queries for your MongoDB database.

1. Find all students who are older than 25 years.
2. Find all students and return only their names and courses, without including their addresses or other details.
3. Retrieve the names of all students and sort them by age in descending order.
4. Find all students who live in "Manchester" and have "Traveling" as one of their hobbies.
5. Sort the students by course in alphabetical order and, if there are ties, sort by age in ascending order.
6. List the details of the top 3 youngest students.
7. Find the top 2 oldest students who have "Reading" as one of their hobbies.
8. Find all students from "Leeds" and update their course to "Data Science".
9. Find the student named "Ivy Carter" and change her course to "Corporate Law" if her age is 27.
10. Remove the Hobbies field from all students who are studying "Biology"
11. Increase the age of all students studying “Chemistry" by 1 year.
12. Delete the student named "Bob Smith" from the collection.
13. Delete all students who are older than 30 years.