MongoDB Mongo shell and Data base creation

MongoDB Mongo shell is an interactive JavaScript interface that allows you to interact with MongoDB instances through the command line. The shell can be used for:

- Data manipulation
- Administrative operations such as maintenance of database instances

MongoDB Mongo shell features

MongoDB Mongo shell is the default client for the MongoDB database server. It's a command-line interface (CLI), where the input and output are all console-based. The Mongo shell is a good tool to manipulate small sets of data.

Here are the top features that Mongo shell offers:

- Run all MongoDB queries from the Mongo shell.
- Manipulate data and perform administration operations.
- Mongo shell uses JavaScript and a related API to issue commands.
- See previous commands in the mongo shell with up and down arrow keys.
- View possible command completions using the tab button after partially entering a command.

- Print error messages, so you know what went wrong with your commands.

MongoDB has recently introduced a new mongo shell known as **mongosh**. It has some additional features, such as extensibility and embeddability—that is, the ability to use it inside other products such as VS Code.

Installing the mongo shell

The mongo shell gets installed when you install the MongoDB server. It is installed in the same location as the MongoDB server binary.

If you want to install it separately, you can visit the MongoDB download center, from there select the version and package you need, download the archive, and copy it to a location in your file system.
Mongo shell is available for all main operating systems, including:

- Windows
- Linux
- Mac OS

Connect to MongoDB database

Once you've downloaded and installed MongoDB, you can use the mongo shell to connect with a MongoDB server that is up and running.
Note: It is required that your server is already running before you connect with it through the shell. You can start the server in CMD using the following command.

Copy

```
net start MongoDB
```

```
C:\windows\system32>net start mongodb
The MongoDB Server (MongoDB) service is starting.....
The MongoDB Server (MongoDB) service was started successfully.
```

Basic commands for Mongo shell

Now it's time to work with the Mongo shell. First, we will learn some basic commands that will help you to get started with using MongoDB.

Run the **db** command to see the database you are currently working with

```
db
```

Run the **use** command to switch to a different database. If you don't have a database,

```
use company
```

Run the **use** command to switch to a different database. If you don't have a database, learn how to create a new database.

```
use company
```

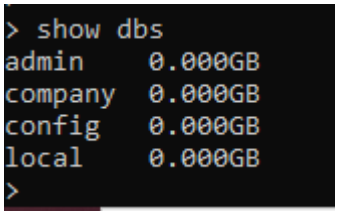
Use the **find** method to fetch data in a collection. The **forEach(printjson)** method will print them with JSON formatting

```
db.employee.find().forEach(printjson)
```

```
> db.employee.find().forEach(printjson)
{ "_id" : ObjectId("5f736ec9cb9ff210daae2271"), "name" : "mark" }
>
```

Use the show dbs command to Show all databases

```
show dbs
```



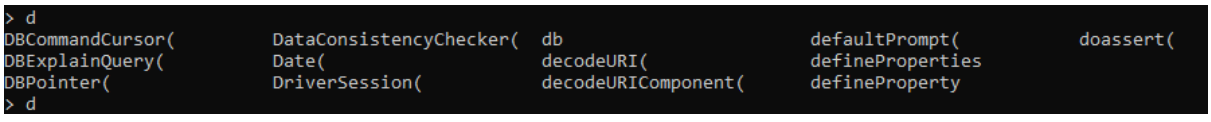
One important command will help you work with the Mongo shell easily: the **help** command. Run the help command to get a list of help options available in the mongo shell.

To get a full list of commands that you can execute on the current database, type **db.help()**

Mongo shell keyboard shortcuts

There are two important keyboard shortcuts that you should know:

1. **Use up and down arrows** to go back and forth in the commands history.
2. **Press the tab key** to get a full list of possible commands. For example, type **d** and press tab twice. You will get the following output.



Disadvantages of the mongo shell

Although the Mongo shell is an excellent tool for learning and testing the MongoDB server, it is difficult to be used in a production environment. Being a shell inherently carries certain disadvantages. Let's see what they are:

- The Mongo shell is strictly a console centric method of data manipulation. While some find it easy and quick, others might not find those characteristics appealing.
- If you are working on multiple sessions, you need multiple terminals.
- If the results are too long, they scroll away.
- Repetitive commands or debugging a function need the programmer to traverse the long command line history manually.

Alternatives to MongoDB mongo shell

So now you know the mongo shell has some disadvantages. At this point, you may want to know what other options are available. MongoDB developers have introduced drivers specific to each programming language to connect with the MongoDB databases when using MongoDB in your applications. You can find them here.

Additionally, many people prefer to use GUIs to work with databases nowadays. One of the best GUI tools for MongoDB is the MongoDB Compass. Some other useful GUI tools are:

- NoSQLBooster
- Mongo Management Studio
- Robo 3T

We connect to the Mongo terminal using the mongo command

```
mongo
```

By default Mongo will connect to localhost.

We can connect to a remote server by passing arguments, like so:

```
mongo connection.mongolab.com:45352 -u username -p passw0rd
```

Once we connect to a Mongo instance we can type JavaScript directly into the console. We can create variables, do maths, write JSON.

Exercise - connect to a console

Connect to the console at localhost. Try typing some JavaScript expressions.

- Tell me how many seconds there are in a week
- Tell me how many weeks there are in a human lifetime of 80 years.

Creating a database

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

```
use petshop
```

This will switch to writing to the petshop database. It doesn't matter if the database doesn't exist yet. It will be brought into existence when you first write a document to it.

You can find which database you are using simply by typing db. You can drop the current database and everything in it using db.dropDatabase.

```
db
```

```
> petshop

db.dropDatabase()
```

Exercise - Create a database

- Use the use command to connect to a new database (If it doesn't exist, Mongo will create it when you write to it).

That was easy wasn't it. Don't worry, it gets a bit harder.

Collections

Collections are sets of (usually) related documents. Your database can have as many collections as you like.

Because Mongo has no joins, a Mongo query can pull data from only one collection at a time. You will need to take this into account when deciding how to arrange your data.

You can create a collection using the createCollection command.

```
use petshop

db.createCollection('mammals')
```

Collections will also be created automatically. If you write a document to a collection that doesn't exist that collection will be brought into being for you.

View your databases and collections using the show command, like this:

```
show dbs
```

```
show collections
```

Exercise - Create a collection

- Use db.createCollection to create a collection. I'll leave the subject up to you.
- Run show dbs and show collections to view your database and collections.

Documents

Documents are JSON objects that live inside a collection. They can be any valid JSON format, with the caveat that they can't contain functions.

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

Creating a document

You can create a document by inserting it into a collection

```
db.mammals.insert({name: "Polar Bear"})
```

```
db.mammals.insert({name: "Star Nosed Mole"})
```

Exercise - Create some documents

- Insert a couple of documents into your collection. I'll leave the subject matter up to you, perhaps cars or hats.

Finding a document

You can find a document or documents matching a particular pattern using the find function.

If you want to find all the mammals in the mammals collection, you can do this easily.

```
db.mammals.find()
```

What is REST API?

A REST API, or Representational State Transfer Application Programming Interface, is a set of rules and conventions for building and interacting with web services. It is a widely used architectural style for designing networked applications. In a RESTful system, resources (such as data or services) are identified by unique URLs, and interactions with these resources are performed using standard HTTP (Hypertext Transfer Protocol) methods (popularly known as CRUD operations) like GET, POST, PUT, and DELETE.

REST APIs operate on a stateless communication model, meaning each request from a client to a server contains all the information needed to understand and fulfil that request, without relying on previous interactions.

Real-life analogy

Imagine a REST API as a library where each book represents a resource, and the librarian (server) provides a catalog with unique identifiers (URLs) for each book. You can borrow (GET), return (PUT), add (POST), or remove (DELETE)

books based on the library's rules, all by interacting with the librarian using a standard set of instructions.

Industry applications

REST has become the dominant choice for API implementations due to its simplicity, flexibility, and ease of use. RESTful APIs are widely adopted across industries and are commonly preferred for building web services. SOAP, while still used in certain enterprise scenarios, has seen a decline in popularity due to its complexity. As of a study conducted in 2017, the REST architecture is employed by 83% of APIs, whereas 15% opt for the older SOAP protocol.

Core principles

RESTful APIs adhere to six core principles, five of which are mandatory and one optional. Here's a breakdown:

- **Client-server architecture:** The API separates concerns into a client (requester) and a server (responder). They interact over a network, typically the internet.
- **Statelessness:** Each request-response interaction is independent of past interactions. The server doesn't store any context about the client between requests.
- **Cacheable:** Intermediaries (like browsers or proxies) can cache responses to improve performance and reduce server load.
- **Uniform interface:** Clients access resources using a consistent set of methods (typically GET, POST, PUT, DELETE) that have predictable semantics across different resource types.
- **Layered system:** Intermediaries can be placed between clients and servers without affecting the communication. This enables functionalities like security, load balancing, and caching.

- **Code-on-demand (optional):** Servers can dynamically deliver executable code (e.g., scripts) to clients to extend their functionality.

Pre-requisites

Creating an optimal environment streamlines coding, testing, and collaboration, improving efficiency, and ensuring a solid foundation for robust REST API development. Setting up this development environment entails multiple steps for a smooth and efficient development process.

Here's a general guide to help you set up a REST API development environment:

1. **Planning:** Define the resources your API will expose, their representations (e.g., JSON, XML), and the operations you want to support (e.g., CRUD).
2. **Choose a programming language and framework:** Popular options include Node.js (Express.js), Python (Django, Flask), Java (Spring Boot), and Ruby (Rails).
3. **Set up the development environment:** Install the necessary tools and libraries for your chosen language and framework.
4. **Design the API endpoints:** Define the URIs for accessing resources and the HTTP methods (GET, POST, PUT, DELETE) used for different operations.
5. **Implement the API logic:** Write code to handle requests, process data, and generate responses.
6. **Test and debug:** Ensure your API functions as expected and address any errors.
7. **Document your API:** Provide clear and concise documentation for developers to understand and use your API effectively.

Additional tips for setting up a REST API development environment include:

- Using a **version control system** (e.g., github.com) to track changes and collaborate with others.
- Implementing **security** measures like authentication and authorization to protect your API.
- Choosing a **framework** that provides features like routing, middleware, and error handling.
- Adopting **API testing tools** to automate testing and ensure quality.

Organizations can efficiently build REST APIs by leveraging existing frameworks and tools that cater to specific programming languages. Here are some noteworthy frameworks and tools for building REST APIs:

- **CData API Server:** A lightweight, self-managed application that provides a point-and-click interface for building professional API for your enterprise data.
- **Python Flask:** A web framework in Python, Flask comes equipped with the Flask-RESTful extension, facilitating swift and straightforward REST API development.
- **js:** Known for its scalability, Node.js utilizes the restify framework, which has been adopted by industry giants like Netflix and Pinterest for the development of scalable REST APIs.
- **Ruby on Rails:** With the introduction of Rails 5, Ruby on Rails now incorporates an API mode that streamlines the creation of web APIs, providing a convenient option for REST API development.
- **Spring (Java):** Java's Spring framework is a robust choice for crafting REST APIs. It offers an in-depth tutorial that guides developers through the intricacies of creating RESTful interfaces.

Choosing the right programming language for building a REST API can seem daunting, but it's an important decision that impacts development speed, functionality, and future maintainability. Let's explore some key factors to consider:

- **Project complexity:** Are you building a simple API for internal use or a complex one for public consumption? A simpler language like Python might be fine for the former, while a robust language like Java might be better for the latter.
- **Performance requirements:** Does your API need to manage high traffic or complex calculations? Languages like Go or Node.js excel in performance, while Python might sacrifice speed for ease of development.
- **Existing skills and preferences:** Choose a language you're already familiar with to accelerate development and ensure you enjoy the process. If you're open to learning something new, consider factors like community size and learning curve.

How to build a REST API

1. Planning your API

By meticulously planning your API, you establish a foundation for successful development to ensure it meets the needs of your target users and provides a structured and intuitive interface for seamless integration into diverse applications.

- **Define the purpose and scope of an API**

    - **Approach:** Clearly articulate the specific goals and limitations of your API and understand the problem it solves and its intended outcomes.
    - **Example:** Consider the purpose of a weather API designed to provide real-time weather data for mobile applications, limiting the scope to current conditions, forecasts, and historical data within a certain timeframe.

- **Identify target users and use cases**
    - **Approach:** Define the primary users and their needs, and identify common use cases and scenarios to tailor the API to user requirements.
    - **Example:** For a financial services API, target users could include app developers, traders, and analysts who might need to view real-time stock prices, analyze market trends, and execute trades.

- **Design a clear and logical API structure**
    - **Approach:** Organize endpoints and data structures in a way that aligns with user expectations. Make sure to follow RESTful principles for simplicity and consistency.
    - **Example:** In a social media API, structure endpoints logically— **/users** for user information, **/posts** for user posts. Use clear naming conventions, such as **/users/{userID}** for individual user details.

2. Creating RESTful endpoints

RESTful endpoints are URLs that represent resources in a web service. Each endpoint corresponds to a specific functionality or data entity in the system, providing a way for clients to interact with the server.

HTTP methods for CRUD

- **GET /patients**: Retrieve a list of all patients
- **GET /patients/{patientID}**: Retrieve details of a specific patient
- **POST /patients**: Create a new patient record
- **PUT /patients/{patientID}**: Update information for a specific patient
- **DELETE /patients/{patientID}**: Delete a patient record

By defining endpoints in this manner, the example of Healthcare API discussed above provides a structured and intuitive interface for managing patient and doctor information, appointments, and medical records.

3. Handling your data

Efficient data handling in a REST API involves thoughtful data modelling, database considerations, and streamlined storage and retrieval processes.

Data modeling and database considerations

Data modeling involves defining the structure of your data and relationships between entities. In the context of RESTful APIs, understanding how data is represented and stored is crucial for efficient operations.