



Data Management with Python

SQL and NoSQL databases



NoSQL

A red rounded square icon with a folded corner, containing the text 'NoSQL' in white. A thin brown line extends from the left side of the icon towards the center of the slide.

Workshop Details

- Instructor: *Tanya Khanna*
- Email: tk759@scarletmail.rutgers.edu
- Course Materials: Github Link -
https://github.com/Tanya-Khanna/DataScienceWorkshop_Fall-2024_NBL
- Workshop Recordings: <https://libguides.rutgers.edu/datascience/python>
- Spring 2024 Workshops: [Link](#)
- Workshop Feedback Form: https://rutgers.libwizard.com/f/graduate_specialist_feedback

Workshops Schedule

Fall 2024 workshops calendar

Introduction to Python Programming	September 9, 2024; 4 – 5:30 PM
Advanced Python Programming	September 16, 2024; 4 – 5:30 PM
Web Scraping with Python	September 23, 2024; 4 – 5:30 PM
Mastering Data Analysis: Pandas and Numpy	September 30, 2024; 4 – 5:30 PM
Data Management with Python: SQL and NoSQL	October 7, 2024; 4 – 5:30 PM
Python for Visualization and Exploration	October 14, 2024; 4 – 5:30 PM
Introduction to Machine Learning: Supervised Learning	October 21, 2024; 4 – 5:30 PM
Introduction to Machine Learning: Unsupervised Learning	October 28, 2024; 4 – 5:30 PM
Deploying Machine Learning Models	November 4, 2024; 4 – 5:30 PM
Ethical AI and Responsible Data Science	November 11, 2024; 4 – 5:30 PM
Large Language Models (LLMs) and ChatGPT	November 18, 2024; 4 – 5:30 PM

Table of Contents

- A. What is Data Management?
- B. Databases in Data Management: Their uses, types.
- C. Relational Database
- D. SQL and SQLite with Python (sqlite3)
- E. Non-Relational Database
- F. Comparison between Relational and Non-Relational Databases
- G. NoSQL and MongoDB with Python (pymongo)
- H. Performance Tuning Tips for SQL and NoSQL Databases

What is Data Management?

“Imagine you’re running a bakery. You’ve got lots of orders, ingredients to track, employees to manage, and customers to keep happy.”



- **Data management** is like organizing all of that information in a way that makes your life easier.
- It's the process of **collecting, storing, organizing, and using data** effectively so you can make better decisions.
- In the digital world, data management is critical because we deal with **tons of data**—customer details, transactions, social media interactions, website visits, inventory, and much more.
- Managing all this data properly ensures you can find the information you need, keep it safe, and make smart choices based on it.

Organizing the Chaos with Databases!

Databases are like the heart of data management. If data management is the bakery, the database is like one's organized pantry.

One wouldn't just dump flour, sugar, and spices into one big container, right? We need **order** to find what we need quickly, and that's exactly what a database does with data.



Here's how databases come into the picture:

- **Storing Data:** Just like a pantry stores your ingredients, a database stores all your data—whether it's customer orders, inventory, or employee details—in a structured way.
- **Accessing Data:** Need to know how much flour is left? In the same way, a database lets you quickly search for information—like finding how many orders a customer has placed or checking stock levels.
- **Updating Data:** Got a new shipment of sugar? You update your pantry. Similarly, databases let you easily update or delete information, like changing customer details or adding new products to your inventory.
- **Securing Data:** You lock your pantry to keep it safe from anyone who shouldn't have access, and databases do the same with data security, controlling who can access or modify the data.

In the business world, databases are the secret ingredient to managing data. They keep everything structured, accessible, and safe so that businesses (or bakers!) can focus on **doing what they do best**. Without databases, data management would be like trying to bake without knowing where anything is—messy, inefficient, and prone to mistakes!

In short: Data management is how we make sense of all the information we deal with, and databases are the tools that make it possible to store, find, and use that data effectively.

Use cases of Databases



E-Commerce Websites: Databases store product information, customer details, and order histories, allowing users to browse products, make purchases, and track their orders.



Social Media Platforms: Platforms like Facebook or Instagram use databases to store user profiles, posts, comments, and interactions, so users can easily access their personal feeds.



Banking Systems: Banks use databases to store account information, transaction histories, and manage the secure transfer of funds.

Healthcare Systems:

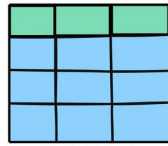
Databases are used to store patient records, medical histories, and treatment information, enabling doctors and hospitals to access and update records efficiently.



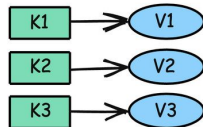
Inventory Management:

Businesses use databases to track stock levels, supplier information, and sales to ensure they have the right products available when needed.

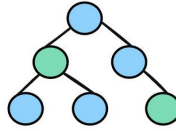
Types of Databases



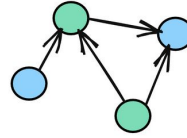
Relational



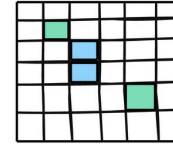
Key-Value



Document



Graph



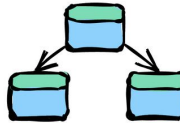
Wide-Column



In-Memory



Time-Series



Object-Oriented



Text-Search



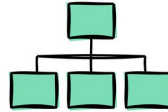
Spatial



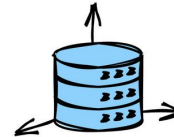
Blob



Ledger



Hierarchical



Vector



Embedded

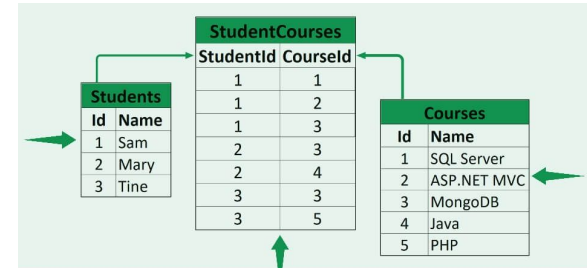
Relational Databases



They are like a well-organized digital filing system!

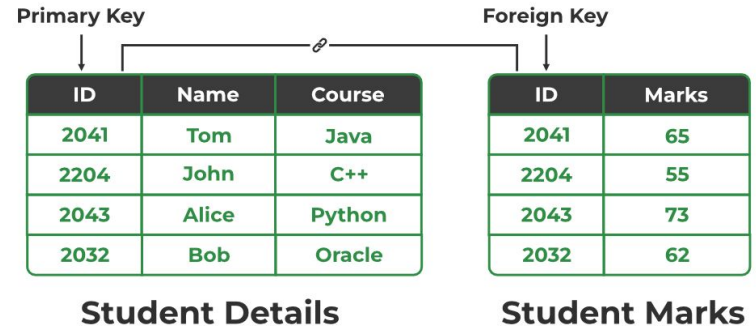
- It stores data in **tables** (similar to grids or spreadsheets) where each piece of information is linked to another.
- These tables can “communicate” through relationships, making it simple to store, retrieve, and manage large amounts of structured data.

Imagine a **giant spreadsheet** where each **table** focuses on a specific topic (such as customers, orders, or products). Data is arranged in **rows** (records) and **columns** (attributes). What makes it powerful is how the tables relate to one another—allowing you to perform quick, efficient queries across different types of data. The database manages these relationships and ensures the information is always accurate and easily accessible.



Elements of a Relational Database

Primary key	Foreign key
It must contain unique values.	It can contain duplicate values.
It cannot contain null values.	It can contain null values.
A database can have only one primary key.	A database can have more than one foreign key.
It is used to identify the records in a table uniquely.	It is used to make a relation between two tables.



Keys are like the **connectors and organizers** of your database. They keep data unique, help tables work together, maintain data accuracy, and make your database run smoothly!

Understanding Queries

*Queries are like the **conversations** between you and your database.*

- It is simply a way to ask questions or request information from a database.
- It's a command that tells the database what data you want to see, update, delete, or add.

"Show me all customers from New York."

"Increase the salary of all employees in the HR department by \$2,000."

"Give me a list of all customers who ordered pizza and the dates they placed their orders."

"Add a new employee to the table. Their name is John Doe, they work in the HR department, and their salary is \$60,000."

What is SQL?

SQL (Structured Query Language) is like the **universal language of databases**. If your database were a country, SQL would be the language everyone speaks there! It's how we tell databases to **show us data, add new information, update existing records**, or **remove** things we don't need.

Ordering food? That's like a **SELECT** query in SQL!

```
SELECT * FROM Menu;
```

Adding a new dish? You'd use an **INSERT** query!

```
INSERT INTO Menu (DishName, Price) VALUES ('Tacos', 10.99);
```

What is SQLite?

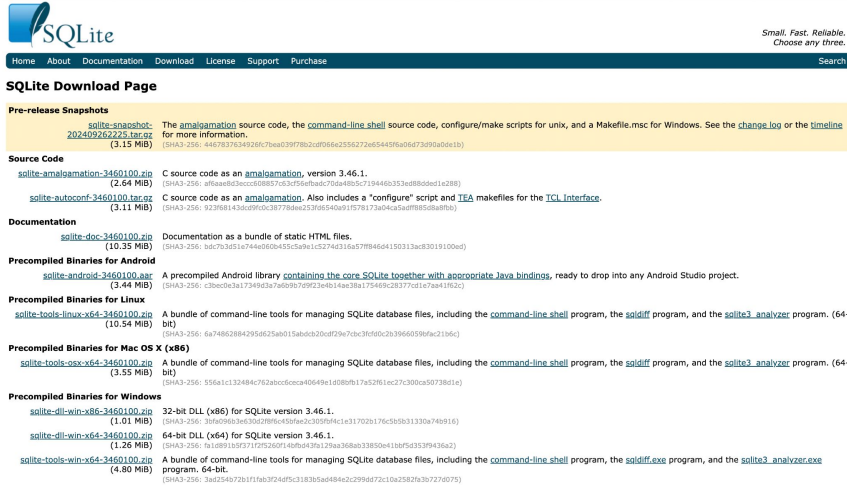
SQLite is like a lightweight, self-contained database system. It doesn't need a fancy setup or a separate server like other databases (e.g., MySQL or PostgreSQL). Instead, everything runs from a single file on your computer.

Why is SQLite Popular?

1. **Simple Setup:** There's no need to install a database server or manage complex configurations. Just download it, and you're ready to go.
2. **File-Based:** All your data is stored in a single file (usually ending in `.db`), making it super portable.
3. **Great for Local Use:** Ideal for smaller projects, mobile apps, or quick testing because it's fast and easy to use without extra software.

Downloading SQLite and Getting Started

Visit <https://www.sqlite.org/download.html>



The screenshot shows the SQLite Download Page. At the top is the SQLite logo and a navigation bar with links: Home, About, Documentation, Download, License, Support, Purchase. Below the navigation bar is the title "SQLite Download Page". The main content area is divided into sections: "Pre-release Snapshots" (with a link to a snapshot), "Source Code" (with links to source code for different platforms), "Documentation" (with a link to documentation), "Precompiled Binaries for Android" (with a link to a precompiled binary), "Precompiled Binaries for Linux" (with a link to a precompiled binary), "Precompiled Binaries for Mac OS X (x86)" (with a link to a precompiled binary), and "Precompiled Binaries for Windows" (with links to precompiled binaries for different architectures). Each section includes a brief description and a link to the download.

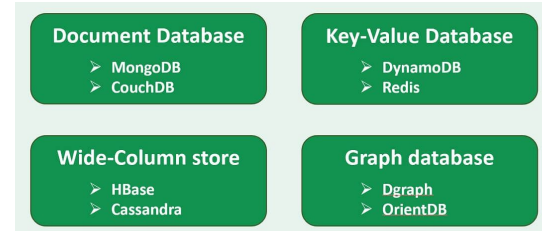
Choose the **precompiled binaries** for your operating system. Extract the downloaded file.



```
(base) tanyakhanna@Tanyas-MBP ~ % cd ~/Downloads/sqlite-tools-osx-x64-3460100
(base) tanyakhanna@Tanyas-MBP sqlite-tools-osx-x64-3460100 % ./sqlite3
zsh: killed      ./sqlite3
(base) tanyakhanna@Tanyas-MBP sqlite-tools-osx-x64-3460100 % ./sqlite3

SQLite version 3.46.1 2024-08-13 09:16:08
Enter ".help" for usage hints.
Connected to a transient in-memory database.
Use ".open FILENAME" to reopen on a persistent database.
sqlite> .open mydatabase.db
sqlite> █
```

What are Non-Relational databases?



Imagine your data is like a collection of **toys**. When you have a **relational database** (like SQL), it's like organizing all your toys into neat, stackable boxes where everything fits perfectly and has a fixed spot. A **non-relational database** (NoSQL), on the other hand, is more like tossing your toys into different bins based on categories—some bins are small, others are large, and you can have some flexibility in how you organize everything. Sometimes, you just need to be flexible!

A **non-relational database** (also known as **NoSQL**) doesn't use the traditional table-based, row-column structure that relational databases like SQL use. Instead, data is stored in a variety of ways—such as **key-value pairs**, **documents**, **graphs**, or **wide-column stores**.

- **Key-Value Stores:** Think of it like a **dictionary**, where each entry has a key (name) and a value (data).
- **Document Stores:** Data is stored as **documents** (usually in JSON or BSON), where each document contains all the information you need about an object.
- **Graph Databases:** Imagine drawing **networks** of how your toys are related—graph databases specialize in showing **relationships** between data points.
- **Wide-Column Stores:** These allow for very **flexible tables** with variable columns.

What is NoSQL?

NoSQL stands for "Not Only SQL." It refers to databases that allow more flexibility in how data is stored and accessed. Unlike relational databases, which follow strict schemas, NoSQL databases can handle **unstructured**, **semi-structured**, or **highly variable** data. This makes them a great fit for fast-changing data, huge datasets, or complex relationships that might not fit into neat tables.

- **Real-Time Data:** Apps that need real-time analysis, like social media feeds, often use NoSQL to handle incoming data quickly.
- **Scalable Applications:** Need to handle millions of users or huge data sets (like online retail or game apps)? NoSQL databases scale horizontally—just add more servers!
- **Flexible Data:** If your data keeps changing (like product catalogs, customer preferences), NoSQL can easily adjust.
- **Complex Relationships:** Graph databases like **Neo4j** are perfect when you're dealing with complex data relationships, such as **social networks** or **recommendation engines**.
- **Unstructured Data:** If your data doesn't fit into a fixed schema (like chat logs, images, or sensor data), NoSQL can manage it easily.

When to Use NoSQL Over SQL?

1. Flexibility in Data Structure:

Use NoSQL when your data doesn't fit into neat, predefined rows and columns. For example:

- **SQL:** Your data is well-structured and follows consistent rules (e.g., customer databases, financial records).
- **NoSQL:** You have semi-structured or rapidly changing data (e.g., product reviews, social media posts).

2. Scalability:

If you expect your app to grow quickly and need to scale out easily, NoSQL databases shine. They're designed to handle **huge loads** of data across multiple servers. Relational databases, while scalable, require more effort and complexity to scale out.

3. Fast Data Retrieval for Big Data:

If you're dealing with huge amounts of data and need to query it fast, especially for **real-time applications**, NoSQL is your best bet. Relational databases are more rigid in their query processes, which can slow things down.

4. Schema Flexibility:

NoSQL lets you change or grow your data structure without breaking things. If you're constantly adding new types of information or modifying existing fields, you don't want to be locked into a rigid schema.

Relational v/s Non-Relational

Feature	Relational (SQL)	Non-Relational (NoSQL)
Structure	Fixed schema (rows & columns)	Flexible schema (documents, key-value pairs)
Scalability	Scales vertically (more power to one server)	Scales horizontally (more servers)
Data Type	Structured data (e.g., customer data)	Unstructured or semi-structured data (e.g., social media posts)
Query Language	Uses SQL (Structured Query Language)	Varies by NoSQL type (some support SQL-like queries)
Performance	Slower for large, dynamic datasets	Faster for big data, high read/write loads
Use Cases	Banking, e-commerce, ERP systems	Social media, real-time apps, product catalogs

What is MongoDB?



MongoDB is a popular **NoSQL** database that stores data in a **flexible, document-based format**. Unlike traditional databases that use tables and rows (like SQL), MongoDB uses collections of **documents** to store data, making it more flexible and easier to scale.

Key Concepts in MongoDB

1. Collections:

A **collection** in MongoDB is like a **folder** that holds multiple documents. It's equivalent to a table in SQL, but much more flexible because documents in the same collection don't have to follow the same structure.

- **Example:** A **collection** called **Users** could hold all user data, like names, addresses, and preferences.

2. Documents:

A **document** is a **single record** in a collection. It's similar to a row in a SQL table but stored in a flexible JSON-like format. Each document can have a different structure, meaning it can store different fields and data types.

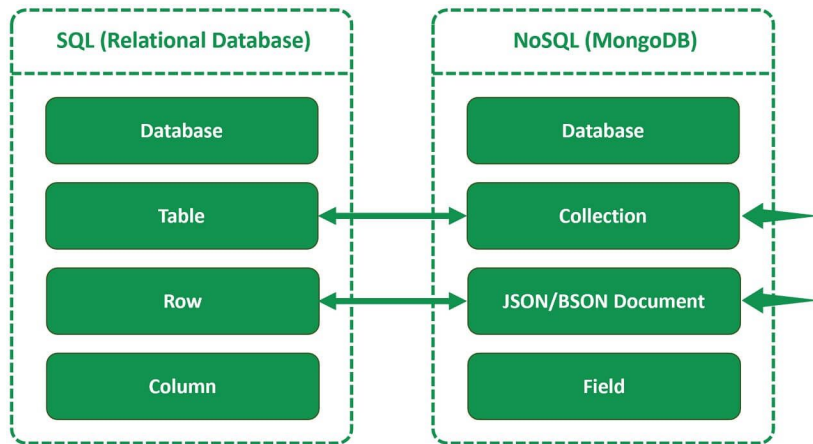
- **Example:** A document in the **Users** collection could look like this:

```
{  
  "name": "Alice",  
  "email": "alice@example.com",  
  "age": 28  
}
```

3. Fields:

A **field** in MongoDB is like a **column** in a relational database. It's a key-value pair inside a document, where the key is the field name, and the value is the actual data.

- **Example:** In the document above, **name**, **email**, and **age** are fields, and "Alice", "alice@example.com", and 28 are their respective values.



MongoDB stores data in **BSON** (Binary JSON) format. While **JSON** is a text-based format that's easy for humans to read and write, **BSON** is optimized for efficiency and performance in MongoDB.

JSON (JavaScript Object Notation):

- **JSON** is a lightweight data format that is easy to read and write for humans.
- It is commonly used to exchange data between servers and web applications.
- **Example of a JSON Document:**

```
{  
  "name": "Bob",  
  "age": 30,  
  "is_student": false  
}
```

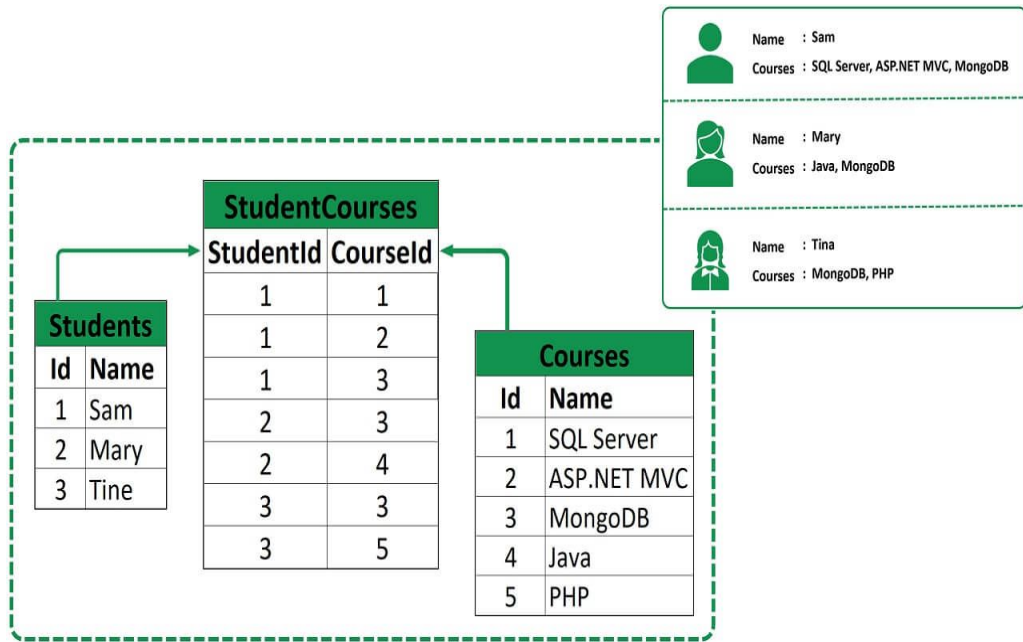
BSON (Binary JSON):

- **BSON** is a binary version of JSON, optimized for MongoDB. It allows for faster reading/writing of data and supports more data types (e.g., dates, binary data).
- While **BSON** is not as human-readable as JSON, it's more efficient for storage and performance.
- **Example of a BSON Document** (conceptual, since BSON is binary):
 - {
 - "name": "Bob",
 - "age": int32(30),
 - "is_student": false
 - }
 -
- In BSON, **age** would be stored as an **int32** type, optimizing storage and access.

Simplifying Data with MongoDB: From Relational to Non-Relational

Imagine we are building an application for a training institute, and we need to show the names of students along with the courses they have enrolled in on a webpage. To model this scenario using a relational database like SQL Server or Oracle, we would typically create three tables:

- **Students:** This table stores student details such as their ID and name.
- **Courses:** This table holds information about courses, like course IDs and names.
- **StudentCourses:** Since there is a many-to-many relationship between students and courses—meaning a student can enroll in multiple courses and a course can have many students enrolled—this table serves as a link between the two, acting as a bridge to manage this relationship.



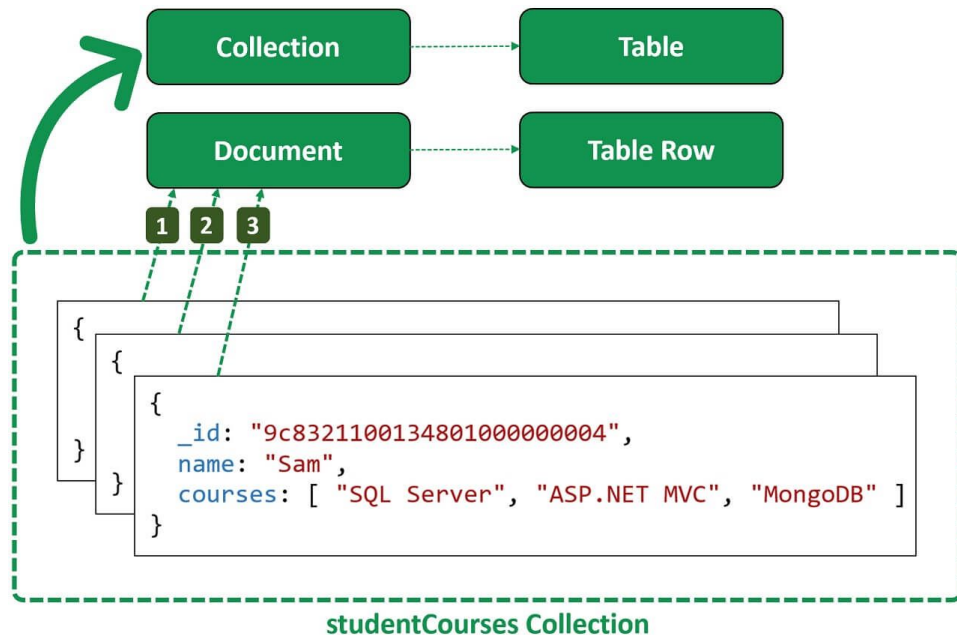
Simplifying Data with MongoDB: From Relational to Non-Relational

Now, let's see how this data might be modeled using a non-relational database like **MongoDB**. Unlike relational databases that use tables and rows, MongoDB uses **collections** and **documents**. You can think of a collection in MongoDB as similar to a table, and a document as equivalent to a row in that table.

In this case, we have just **one collection**, named `studentCourses`, which contains three documents. Just like a table holds rows, a collection holds documents, and the document is what actually stores the data.

MongoDB stores data in **BSON** (Binary JSON) format, so the documents are referred to as **BSON documents**. If you're wondering what BSON is, it's essentially a binary version of JSON that MongoDB uses to store data on disk.

The same data we spread across three tables in a relational database (Students, Courses, and StudentCourses) can be stored in a **single collection** in MongoDB. This makes the data much easier to work with in our application, as we avoid the need for complex joins or mapping layers. The result is cleaner, simpler code that is easier to maintain.



Installing and Setting Up MongoDB Atlas

MongoDB Atlas, is a free cloud-based MongoDB service. It's a fully-managed database that allows you to create, manage, and scale MongoDB clusters in the cloud, and it's free for small applications.

Step 1: Create a Cluster

1. **Click on the "Create" button** under "Create a cluster".
2. **Choose a cloud provider and region:**
 - Pick a **cloud provider** (AWS, Google Cloud, or Azure). You can stick with the default if you're not sure.
 - Choose a **region** close to you for better performance.
 - MongoDB offers a **free tier**, so you won't be charged for using the cluster.
3. **Cluster Tier:** Make sure to select the **Mo (Free Tier)** to ensure you don't incur any costs.
4. **Cluster Name:** You can give the cluster a name, or leave it as the default (**Cluster0**).
5. **Click "Create Cluster":** This process might take a few minutes, so wait while your cluster is being provisioned.

Installing and Setting Up MongoDB Atlas

Click on **"Drivers"** (under "Connect to your application"). This option will give you the connection string needed to connect to MongoDB from Python. After clicking on "Drivers," you'll be asked to select your programming language.

- **Choose Python** from the dropdown.
- It should now provide you with a connection string that looks something like this:

mongodb+srv://<username>:<password>@cluster0.mongodb.net/?retryWrites=true&w=majority

Copy the connection string. This is what you'll use to connect your Python code to MongoDB Atlas.

