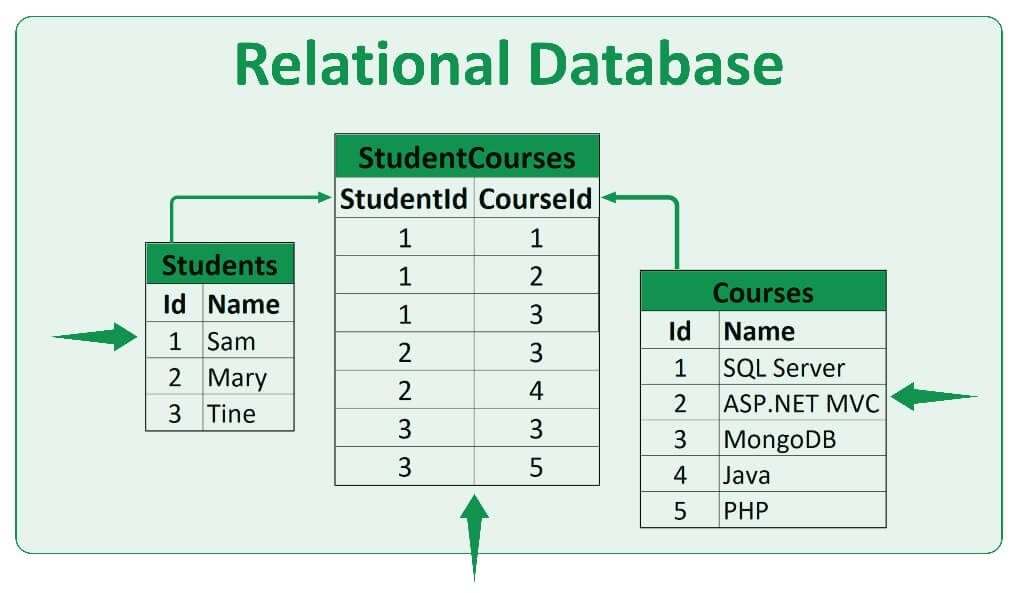
# Relational and non relational databases

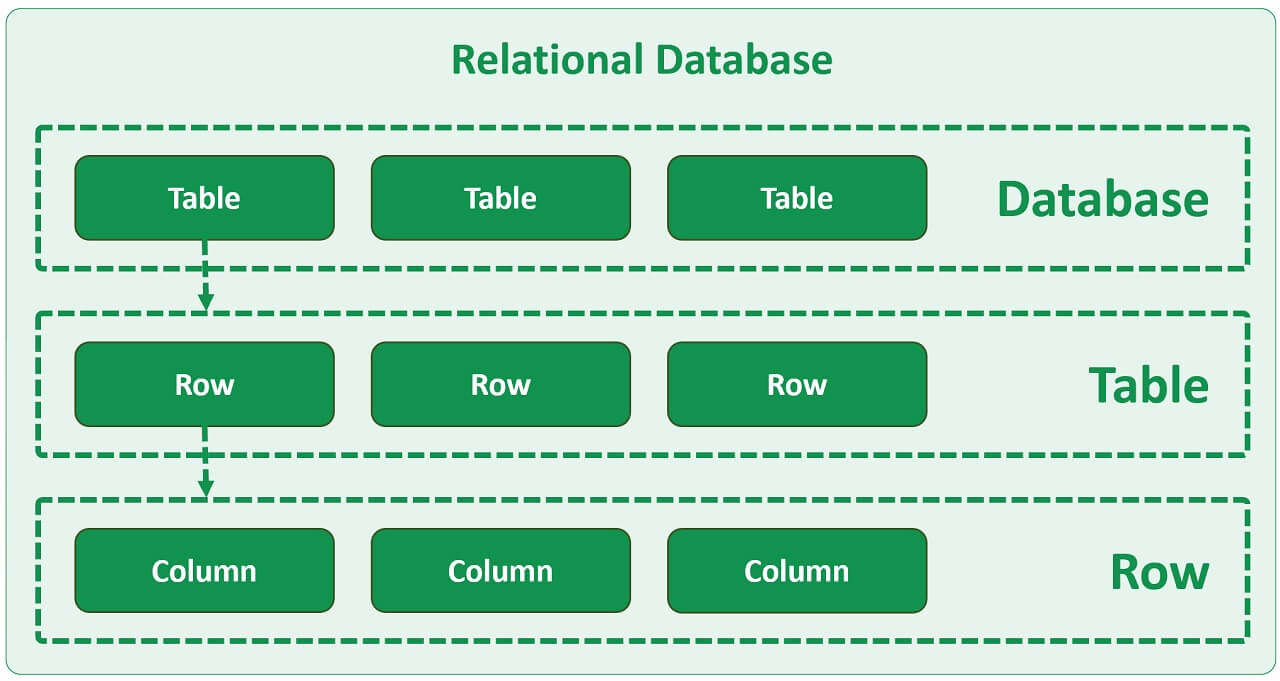
In this short MongoDB video series we will discuss everything you need to know to get strated with MongoDB, a cross-platform non-relational database.

## What is a relational database

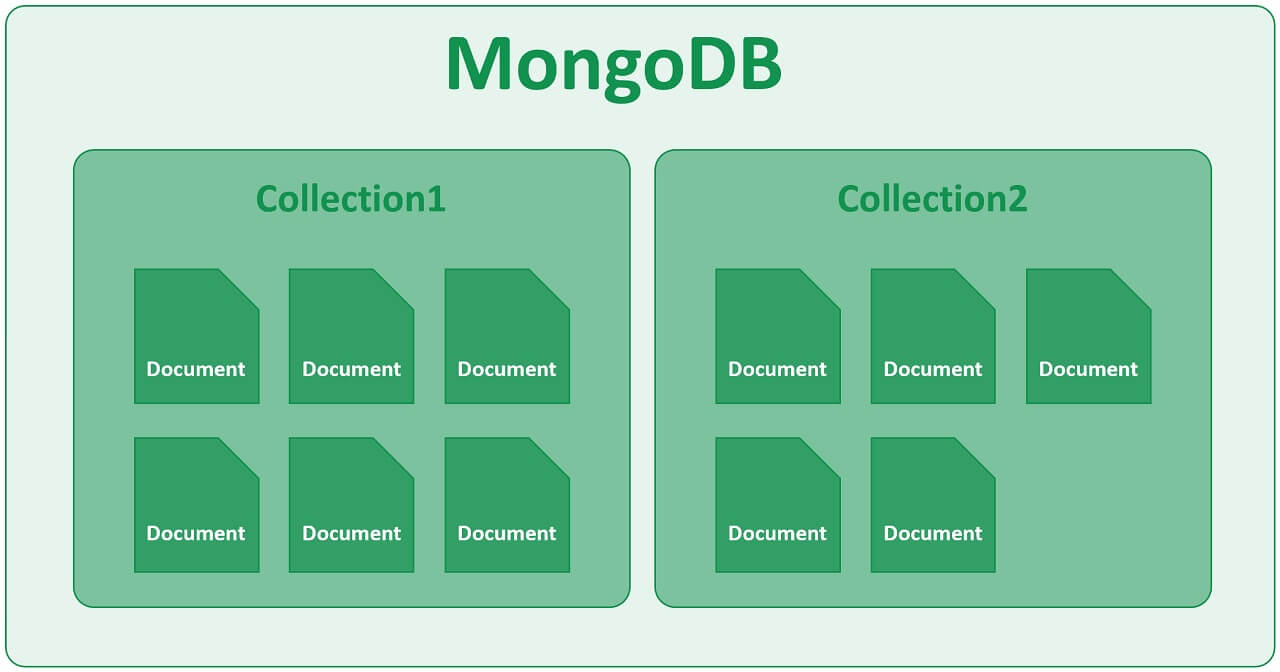
In a relational database like SQL Server or Oracle, a database is collection of tables i.e it contains one or more tables and it is these tables that store data. Simply put, a relational database is a collection of one or more tables.



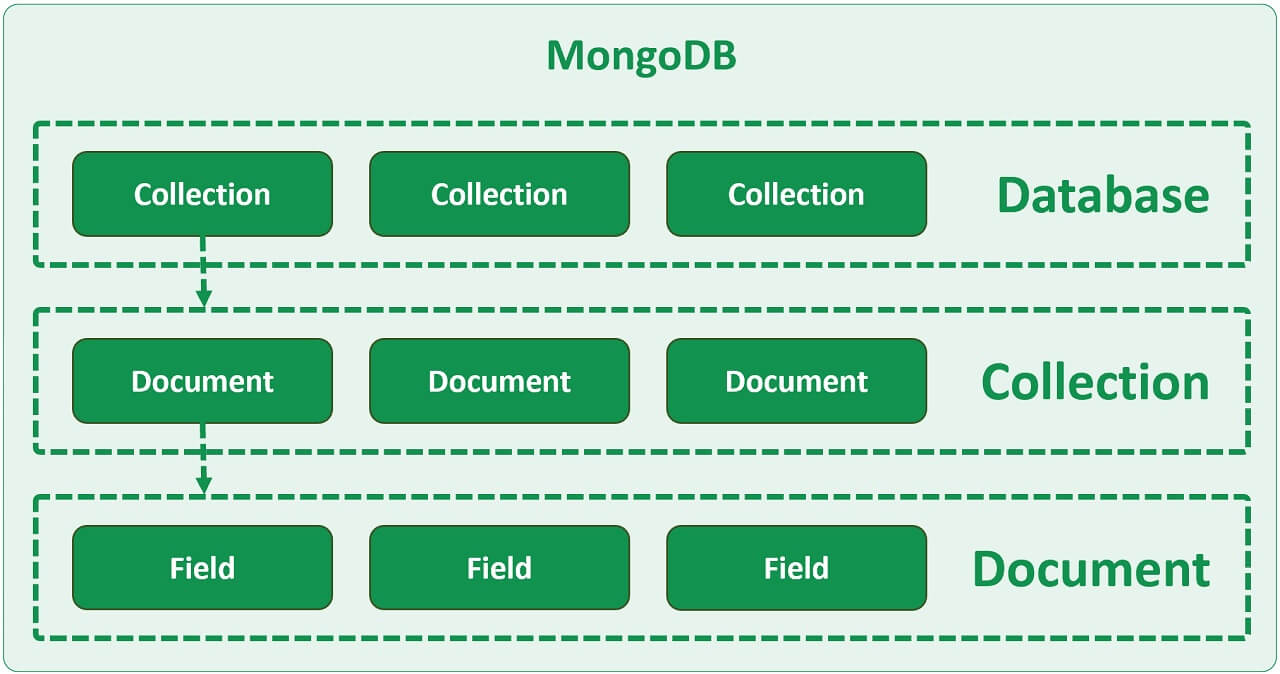
A database table contains one or more rows i.e it is a collection of rows. A table row contains one or more columns.



## What is a non-relational database



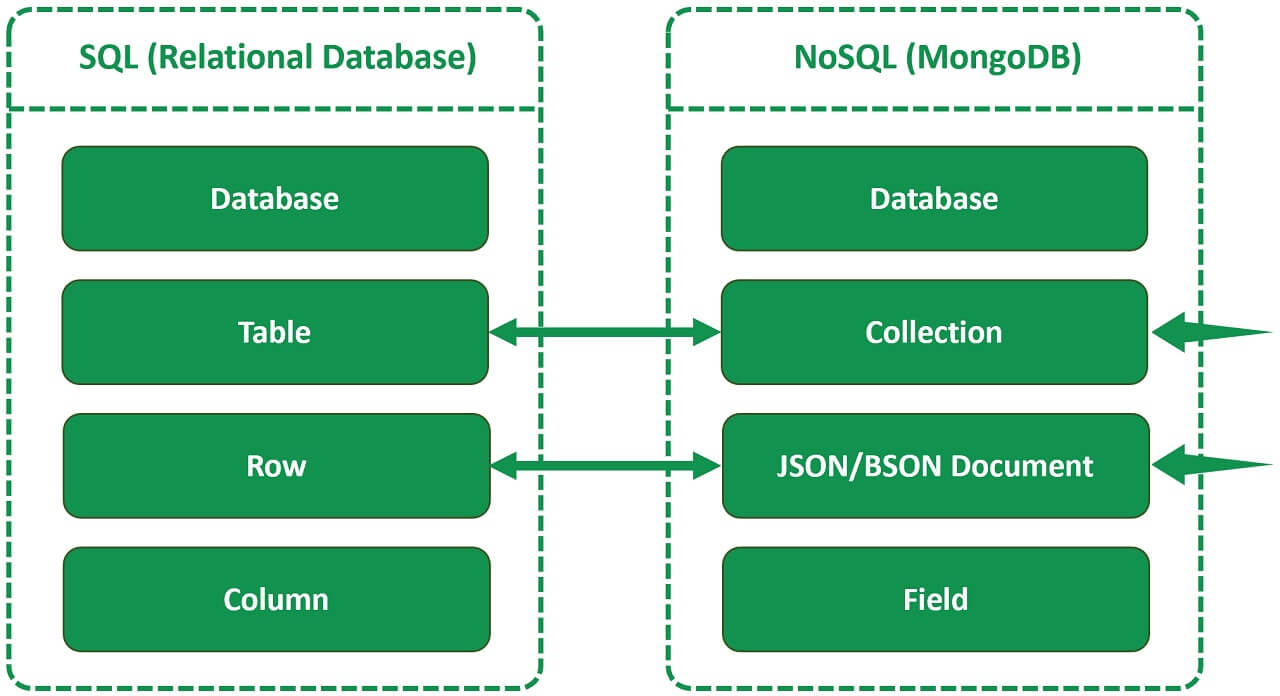
In a non-relational database like MongoDB for example, we do not have tables, instead we have Collections and Documents. A collection, as the name implies, is a collection of one or more documents. If you are from a relational database background, you can think of a collection as a table and documents as table rows.



So in MongoDB, a database contains one or more collections. A collection is made up of one or more documents and each document is made up of one or more fields.

## Relational vs non-relational databases (technical terms)

1. Now, both in SQL and non-relational world, we have Database at the top. In SQL we have tables and in the non-relational world (like MongoDB for example) we have Collections.
2. Table rows in SQL world correspond to BSON documents in MongoDB. BSON is binary JSON. We will discuss JSON and BSON and the difference between them in just a bit.
3. A column in the SQL world corresponds to a field in a JSON document.



It gets a bit easier if we understand these 4 technical terms in MongoDB.

1. Collection
2. Document
3. JSON
4. BSON

So, in a nutshell, think of a collection as a table and a document as a table row.

## Difference between JSON and BSON in MongoDB

It's important to understand the difference between JSON and BSON. At the end of the day, MongoDB is a database and we use it to store data. In a relational database like SQL server data is stored in a table, obviously in rows and columns. In MongoDB we do not have tables, instead we have documents. So data in MongoDB is stored in BSON documents. BSON is Binary JSON, so first let's understand, what is JSON?

What is JSON : The acronym JSON stands for JavaScript Object Notation. In the following JSON document we have student data. JSON format is text-based and both humans and machines can very easily read it. This is the reason it became a very popular format for storing and exchanging data between software systems.

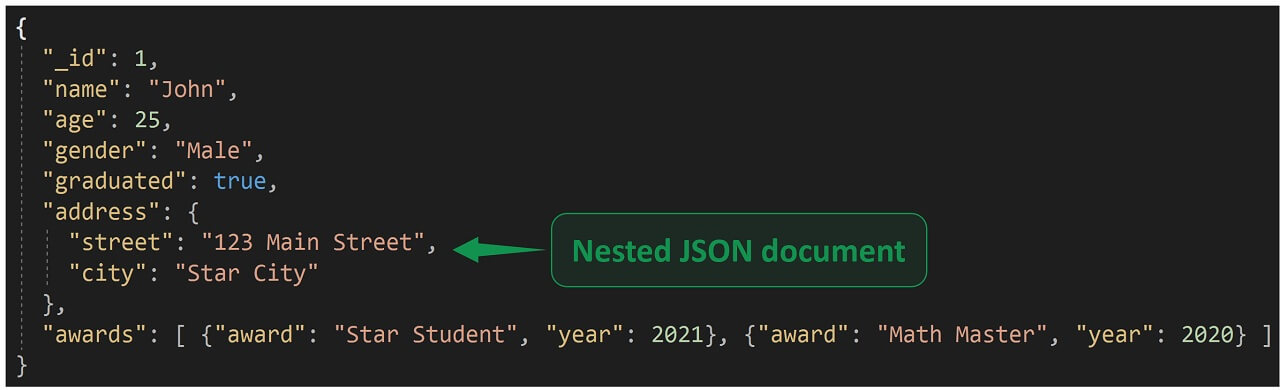


* Data in a JSON document is stored in key/value pairs. Example "name": "John"
* A key/value pair consists of a field name (in double quotes), followed by a colon, followed by a value
* Value can be a string, number, boolean, or an array

In the above example,

* name field value is a string
* age is a number
* graduated is a boolean value (true or false)
* awards is an array

## What is a Nested JSON document



A field value in a JSON document can also be another nested JSON document. In the above example "address" field value is another JSON document with it's own set of fields and values. So the address is an embedded JSON document. There is a 1 to 1 relationship between student and address.

To model something like this in a relational database, we may have 2 tables.

1. Students - Stores students data
2. Addresses - Stores address data

To retrieve data, we have to join and query both the tables. In Mongo we have the entire data in one document. There is obviously no need for expensive join operations.

## Arrays in JSON

"awards": [ {"award": "Star Student", "year": 2021}, {"award": "Math Master", "year": 2020} ]

"awards" field value is an array. A student may have received ZERO or more awards. So there is 1 to Many relationship between students and awards.

Again, to model something like this in a relational database, we may have 2 tables.

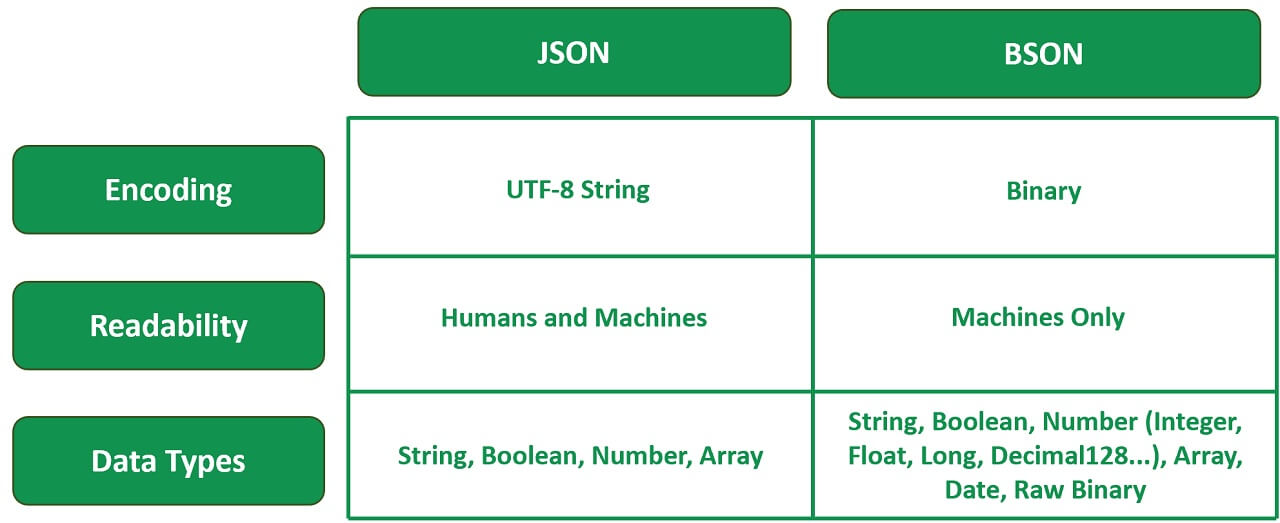
1. Students - Stores students data
2. Awards - Stores awards data

In Mongo we have all the data in one JSON document.

## What is BSON and JSON vs BSON

JSON stands for JavaScript Object Notation and BSON stands for Binary JavaScript Object Notation. Simply put, BSON is Binary JSON and it is in this format MongoDB stores data internally on the disk. So in a nut-shell, BSON is the serialization format used in MongoDB. At this point, you might be wondering, we already have JSON, why did MongoDB invent BSON?

Well, JSON only supports a limited number of basic data types (String, Boolean, Number, Array). Also, JSON is a text-based format, and parsing JSON text is very slow. We want database engines to be fast, we don't want text parsing to slow it down, hence MongoDB invented BSON. The following are the 3 primary differences between JSON and BSON.

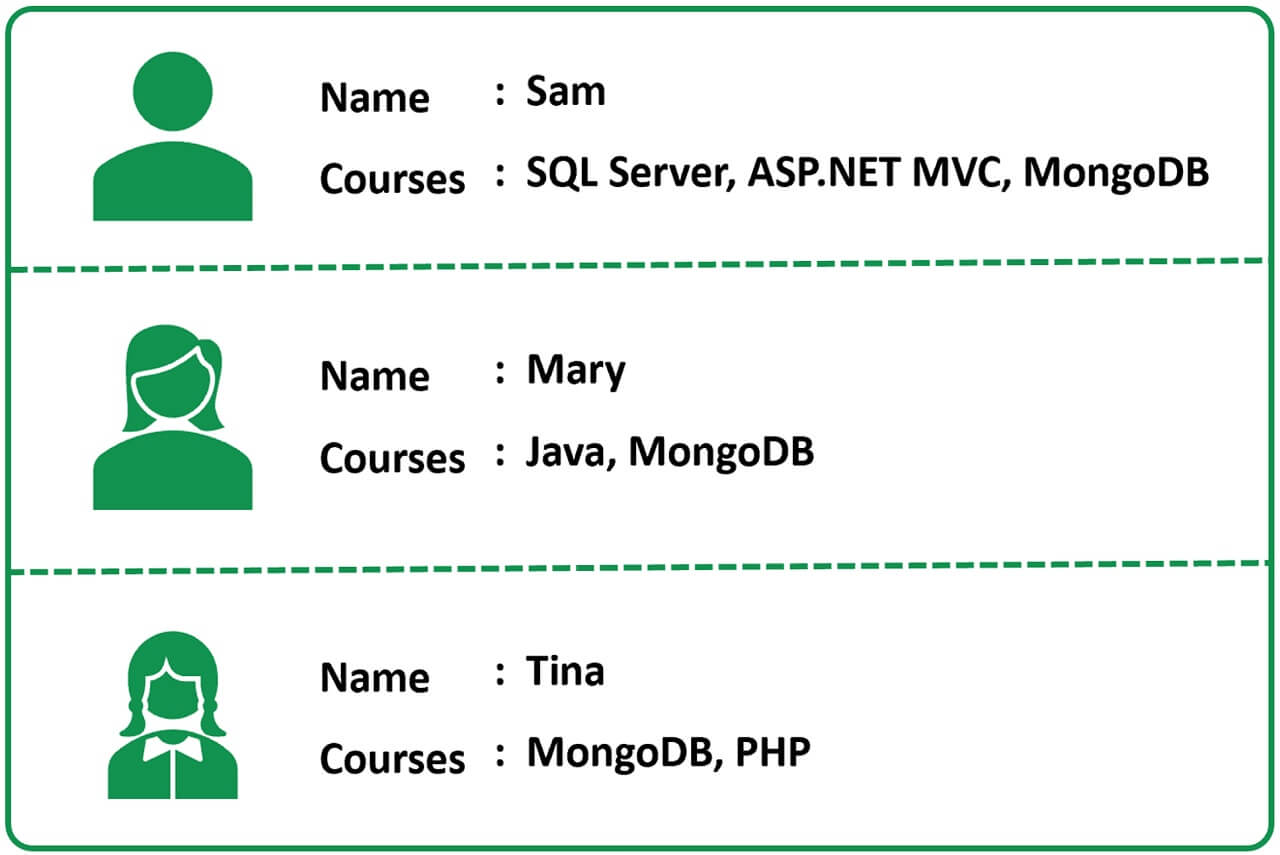


At this point, you might be wondering, should I now learn this new binary BSON format? Well, no. As developers we still use JSON format and MongoDB takes care of converting it to BSON and saving it on the disk. In your JSON documents, if required use the additional data types provided by BSON. We don't have to worry about the internal BSON format, it's all taken care of by MongoDB.

Remember BSON is a binary representation of JSON and it contains more data types than JSON. So, you can think of BSON as a superset of JSON.

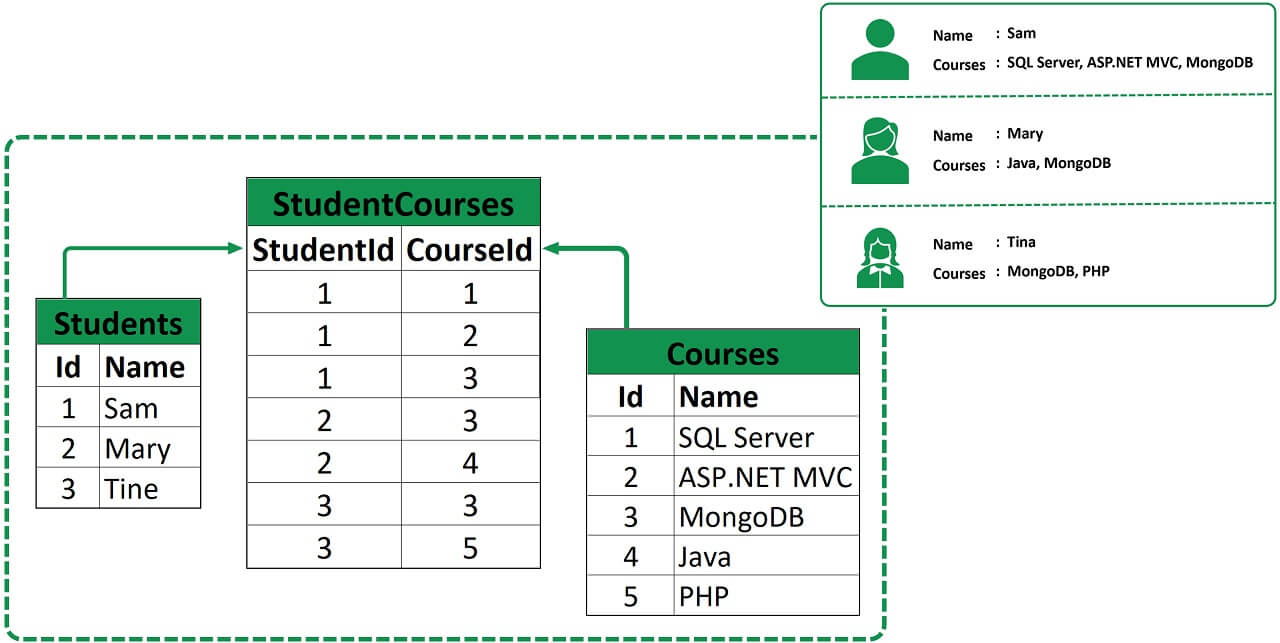
## A very simple but a real world use case

Let's say we are developing an application for a training institute and we have a page that displays student names and the courses that they have enrolled in, on a web page as you can see right here.



To model something like this, in a relational database world like SQL Server or Oracle, we typically create 3 tables.

1. Students - Stores the details of students like Id and Name.
2. Courses  - Stores the course details like course Id and Name.
3. StudentCourses - There is a many to many relationship between Students and Courses tables, meaning, a student can enrol in multiple i.e many courses, similarly a given course can have multiple students enrolled, so this is the table that links students and courses, basically the bridge table for many-to-many relationship between Students and Courses.



On the application side to be able to display this database data we may have a class that looks like the following.

public class StudentCourses

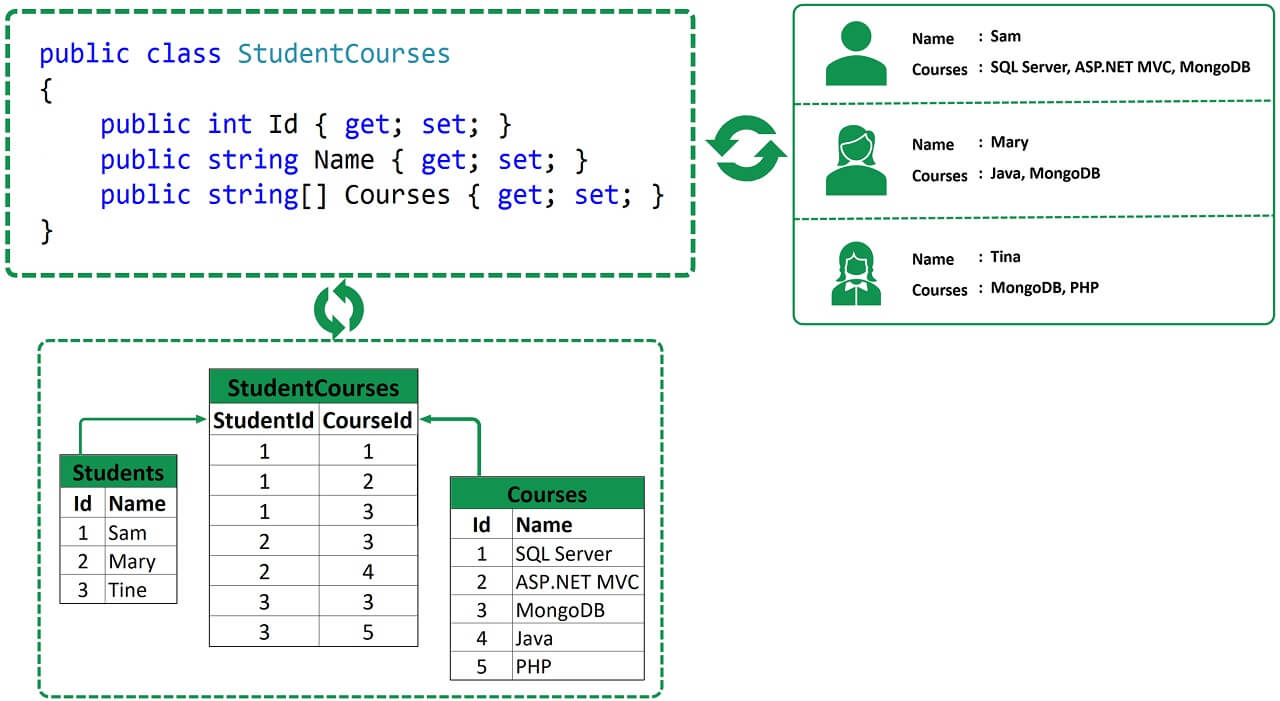
{

public int Id { get; set; }

public string Name { get; set; }

public string[] Courses { get; set; }

}

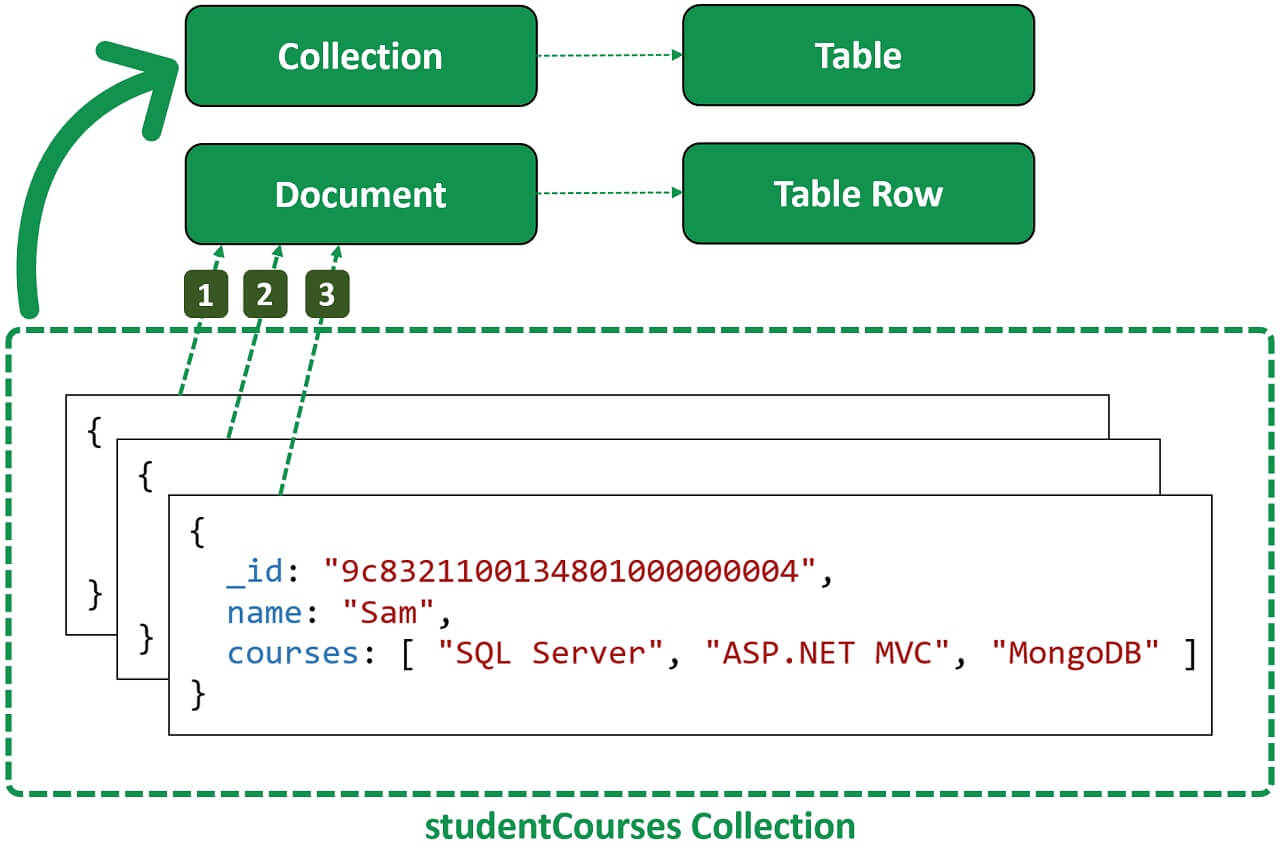


This is a very very simple example here. In the real world in our application layer we may use object oriented programming concepts like inheritance, polymorphism etc between classes which makes it even more difficult and a painful exercise to map our application layer objects to database tables and rows. So the point that I am trying to get across is there is a lot of misalignment between application classes and database tables. Simply put, our application objects are not tables and rows.

Now, to fill this gap and misalignment we may write a custom mapping layer or use ORMs (i.e Object Relational Mappers) like Entity Framework for example. As software engineers we want a database that is easy to use with our application code.

## Non-relational Databases and MongoDB

Now, let's see how this same data may be stored in a non-relational database like MongoDB. As we have already discussed, in Mongo we do not have tables and rows, instead we have collections and documents.



You can think of a collection in MongoDB as a table in a relational database and a document as a table row.

In this example, we have 1 collection. The name of the collection is studentCourses. In this collection we have 3 documents.

So, just like how a table is a container for rows, a collection is a container for documents. It is the document that physically stores data.

The data is in BSON format so the documents are also called BSON documents. If you are wondering what is BSON? Well, it's binary JSON, MongoDB stores data on the disk in BSON format,

The same students and courses data that we have across 3 tables in a relational database can be stored in a single collection in Mongo.

For an application this data is very easy to work with. No complex joins and mapping layers are required. Our application code will be much cleaner, simpler and easier to maintain.

## Relationships between documents in MongoDB

So does this mean we cannot define relationships between documents in a non-relational database like MongoDB?

Well, we can define relationships between documents in MongoDB, but these are usually kept to an absolute minimum. In most cases we embed JSON documents inside another document. The obvious benefit of this approach is we have all the data we need in one place and there is no need to traverse multiple related documents.

Here is the important bit, we can define relationships or references in MongoDB, we can even completely normalize schema in MongoDB just like a relational database, but the question is if you want to normalize and use a non-relational database like a relational database, why use a non-relational database, use a relational database.

## When to use SQL database over non-relational database

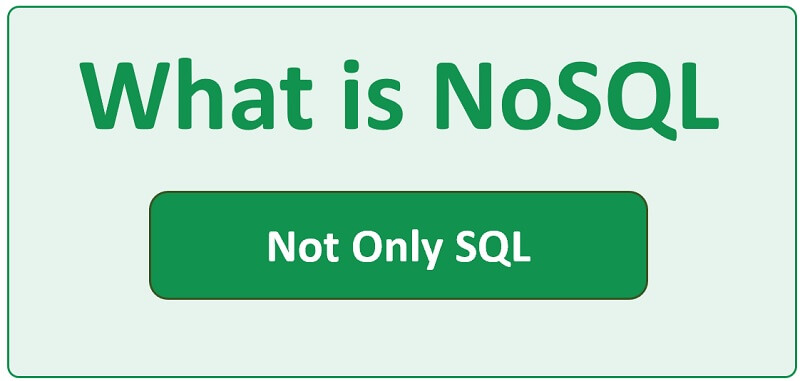
Well, relational databases are still the best choice if the data is predictable, in terms of structure, size, and frequency of access. Also, if relationships between entities are important, again relational databases are the best choice.

Normalization in a relational database reduces the size of the data on disk by limiting duplicate data and anomalies. In a non-relational database documents are embedded within documents, which helps keep data that will be accessed together in the same place, but if this isn’t right for your needs, use a relational database, for example, if you have a large dataset with complex structure and relationships, embedding might not create clear enough relationships.

Finally, relational databases have been around for a very very very long time which means there is wide support available, from tools to integration with data from other systems.

If you want a flexible schema for the data i.e in terms of shape or size, or if it needs to be open to change in the future, then a non-relational database is the answer. non-relational databases have been designed from the groundup for the cloud, which makes them naturally good for horizontal scaling where lots of smaller servers can be spun up to handle increased load.

## What is NoSQL



Lots of people use these two terms - NoSQL database and non-relational database interchangeably. A non-relational database like MongoDB is more than a NoSQL database.

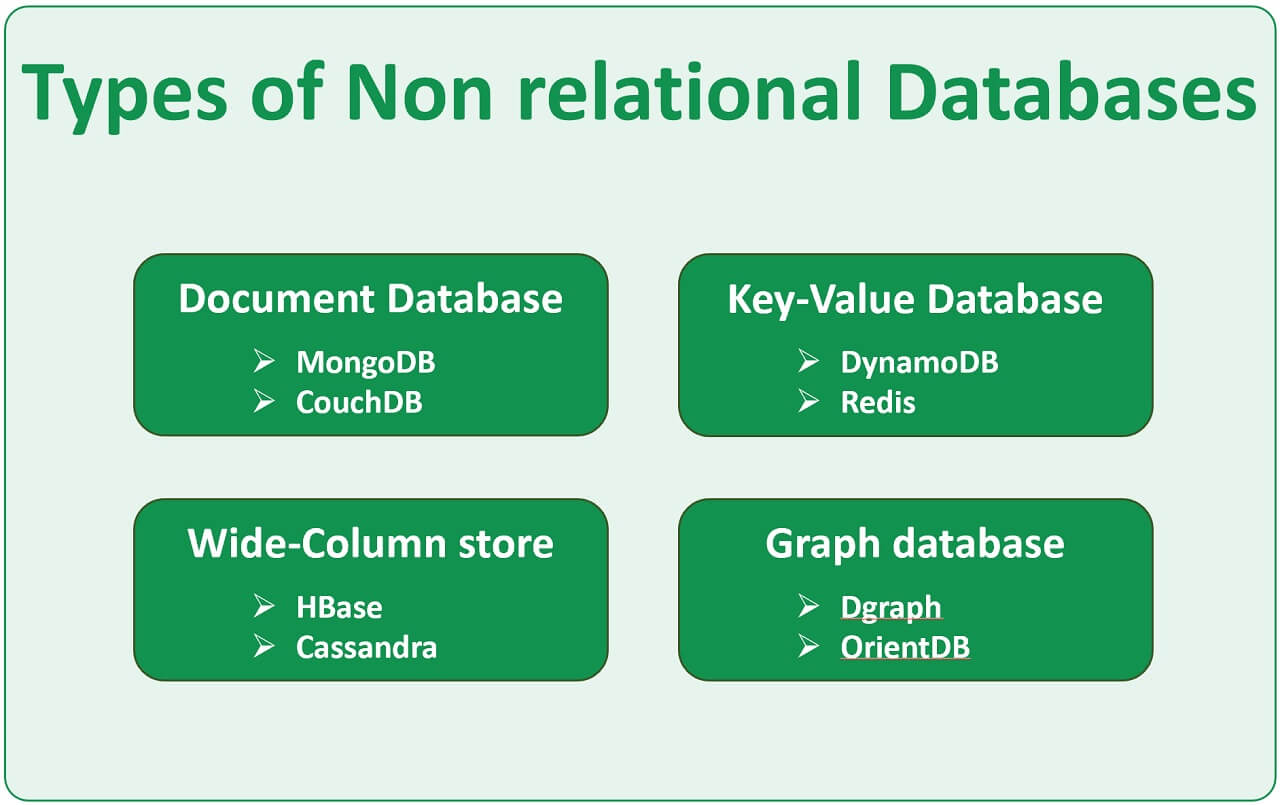
**So, what does NoSQL stand for?**

Well, when we hear this term, we think NoSQL means there is No SQL. This is not true, the term NoSQL stands for Not Only SQL and it means, a non-relational database is a non-tabular database and stores data differently than relational tables.

For example, in a relational database like SQL server or Oracle data is stored in tables whereas in a non-relational database like MongoDB data is stored in JSON documents.

## Types of non-relational databases

There are different types of non-relational databases.



**Document Database** - As the name implies data is stored in documents, for example in JSON documents. Examples of document databases are MongoDB and CouchDB.

**Key-Value Database** - It stores data as a collection of key-value pairs in which a key is used as a unique identifier. Both keys and values can be anything, ranging from simple objects to complex compound objects. Examples are DynamoDB and Redis.

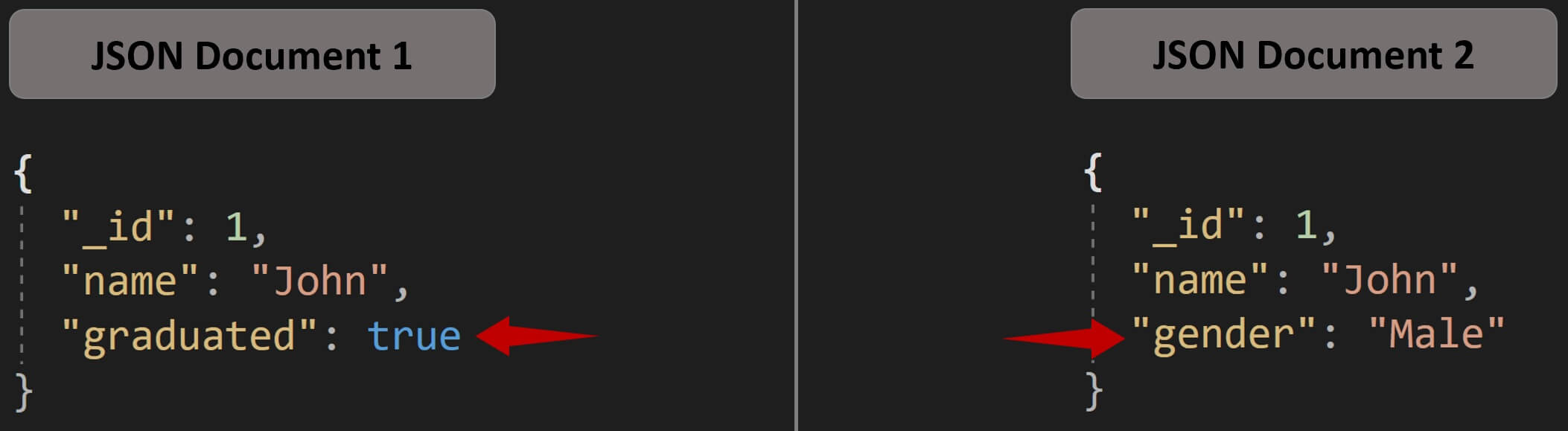
**Wide-Column store** - It is similar to a relational database in the sense it stores data in rows and columns, however the columns are dynamic and each row does not have to have the same columns. Examples are HBase and Cassandra.

**Graph database** - It is used to store and navigate relationships. There is no limit to the kind and number of relationships. Heavily used in fraud detection and social networking. It is easy to create relationships between data entities and quickly query these relationships, for example you can find out who the "friends of friends" of a particular person are. Examples - Dgraph and OrientDB.

## Benefits of a document database

**Easy to use data model :** In a document database like MongoDB we do not have tables and rows, instead we have collections and documents. The data that we typically store across many many relational database tables can be stored in a single collection. For an application this data is very easy to work with. No complex joins and mapping layers are required. Our application code will be much cleaner, simpler and easier to maintain.

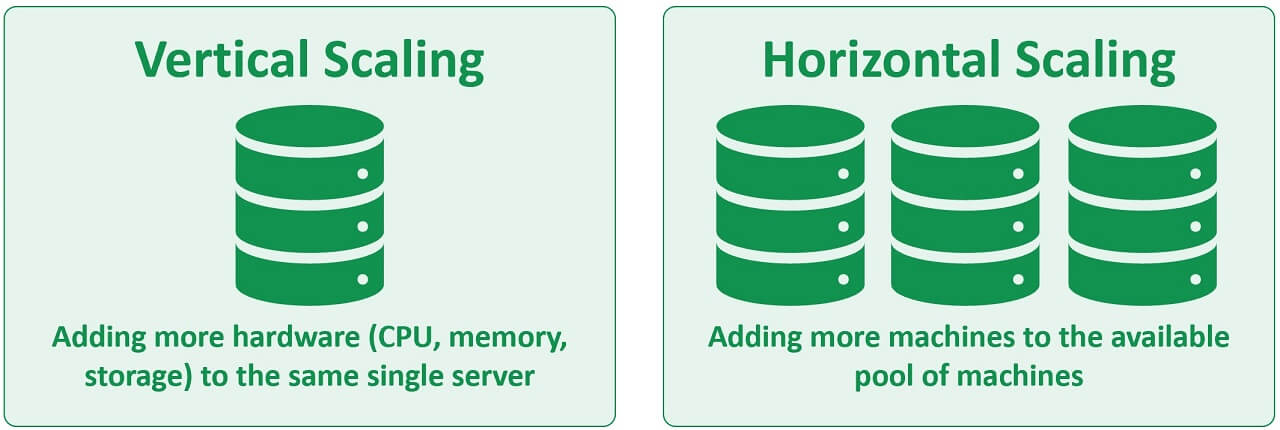
**Flexible schema :** JSON documents that actually store data have a flexible schema which means it is perfectly okay and normal for 2 documents to have different fields. In the following example, we have 2 documents with student data.



Both the documents have id and name fields. In the first document we have graduated boolean field, whereas in the second document instead of graduated we have a string gender field. So the point is, document schema in Mongo is dynamic and self-describing. Every document we save in Mongo can be as flat and simple or as complex as our application requires.

**Feature rich query language :** MongoDB Query Language (MQL) is based on JavaScript and very easy to learn. When querying data, you have a wide range of options, operators, expressions and filters. Data is very easy to query and you rarely need joins or transactions. The MongoDB Query Language (MQL) is a full-featured, powerful language that allows you to query deep into documents, and even use the aggregation framework to perform complex analytics with just a few lines. Indexing is also supported which means documents can be retrieved quickly.

**Easy horizontal scaling :** Most SQL databases allow only vertical scaling which is also commonly called scaling up. Vertical scaling involves adding more resources (like memory, storage etc) to the existing database server. Vertical scaling has an upper limit beyond which you cannot just add more CPU, memory etc. There is also downtime involved. You have to shut the database server down to add more hardware.



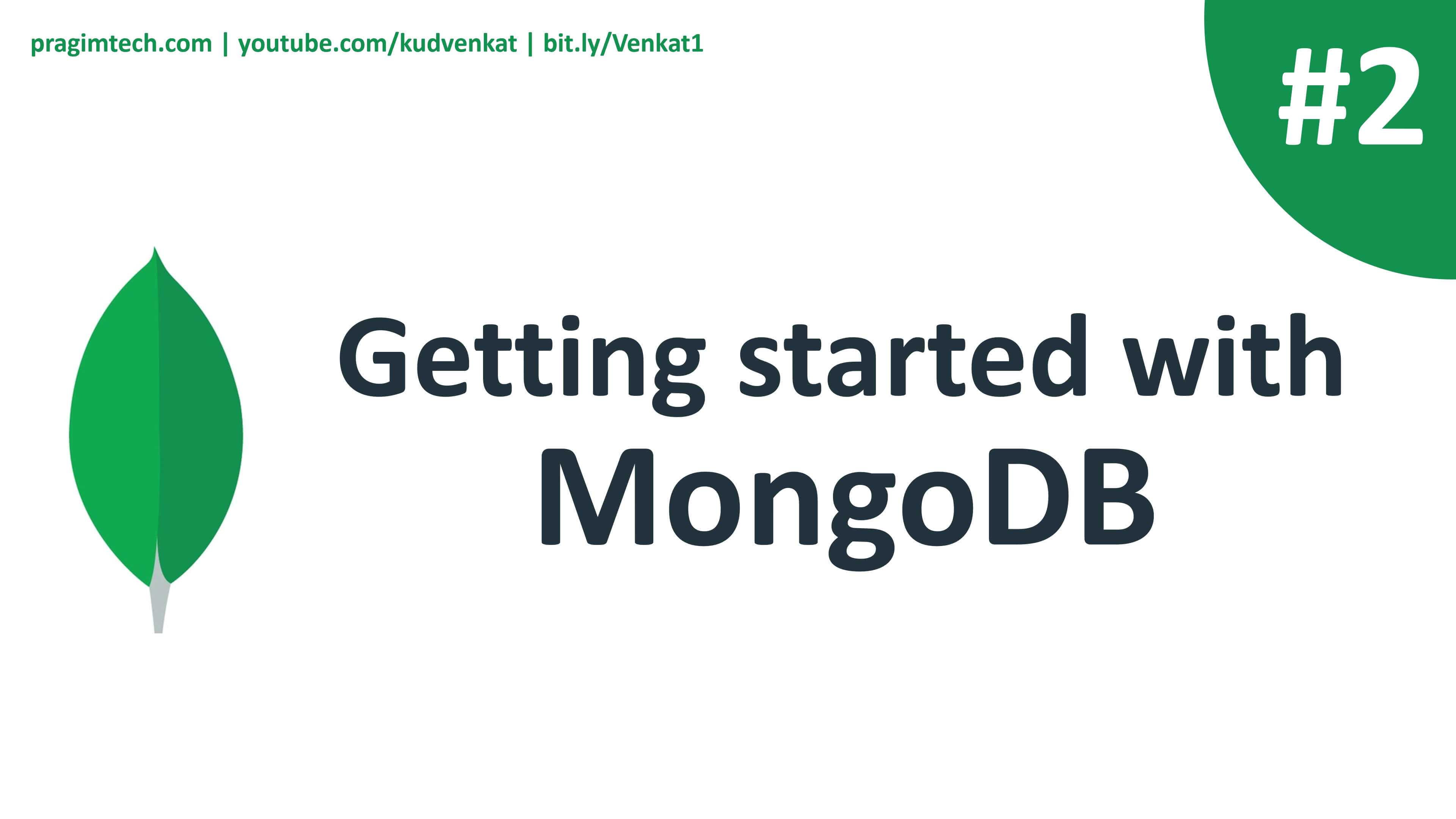
MongoDB on the other hand supports horizontal scaling which is also commonly called scaling out. In horizontal scaling, data is partitioned and distributed across many servers. So in simple terms, scaling horizontally means adding more servers to the already existing pool of servers. This means you are not limited to the capacity of a single server. There is no need to turn off existing servers to add a new server to the pool, so there is no downtime as well.

Non-relational databases like MongoDB have become very popular. The following is one of the common questions asked these days in a Software Engineer interview.

**Any experience with a non-relational database? What's the difference between a relational and a non-relational database?**

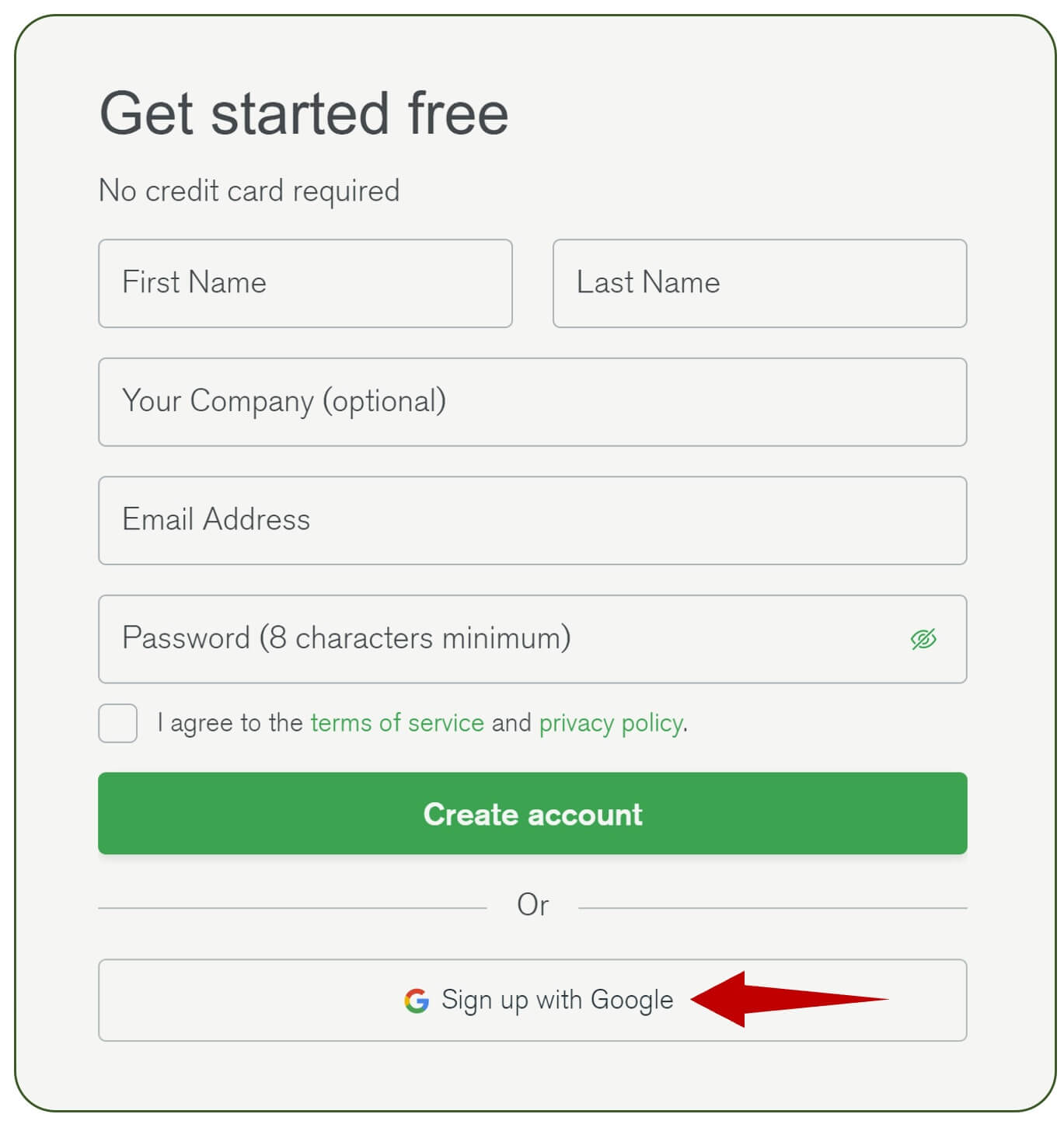
Non-relational databases are gaining great traction and they are here to stay. If you are a software engineer, it’s invaluable to have a non-relational database like MongoDB in your tool belt.

# Getting started with MongoDB databass

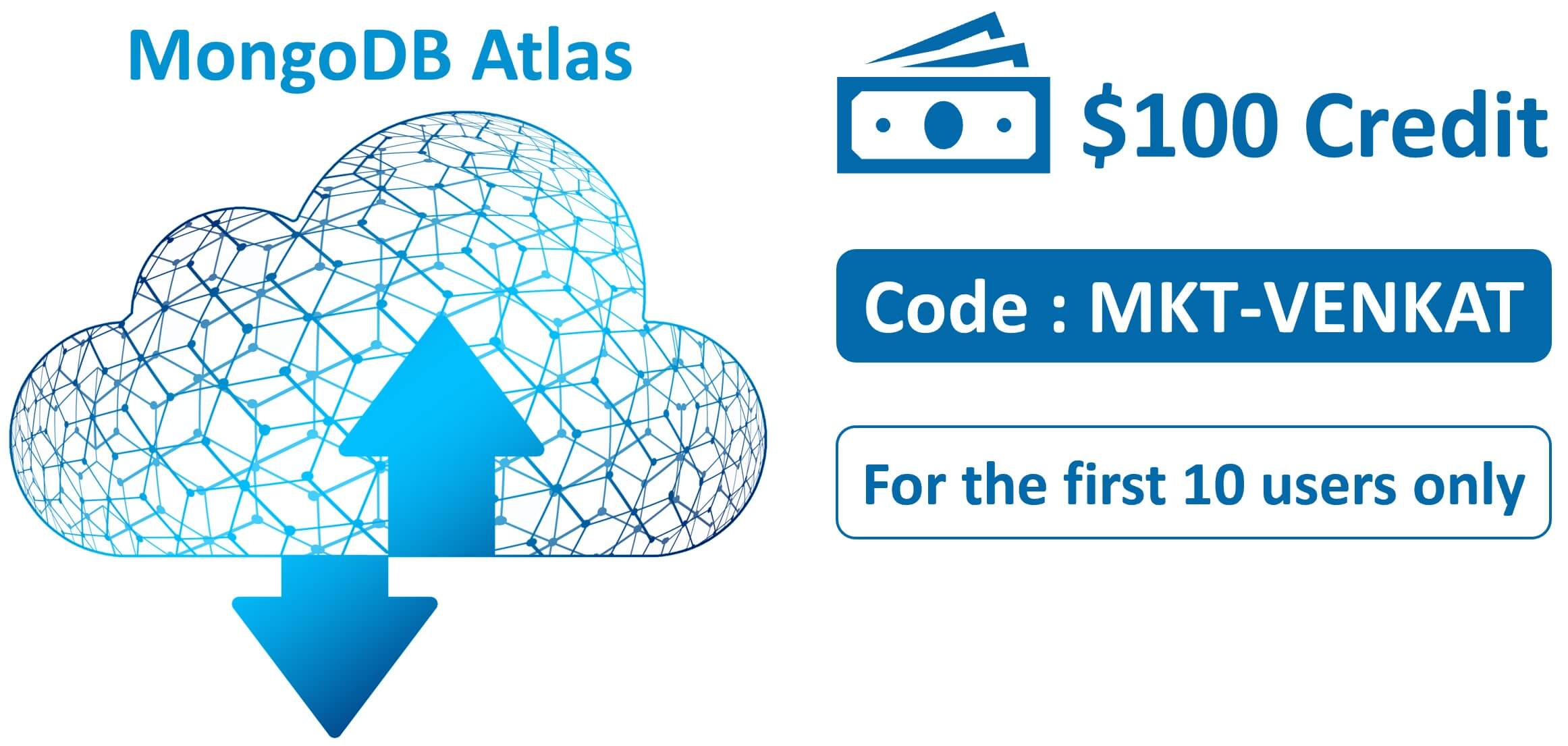


MongoDB is  a multi-cloud document database service. In short it's a document-oriented database.

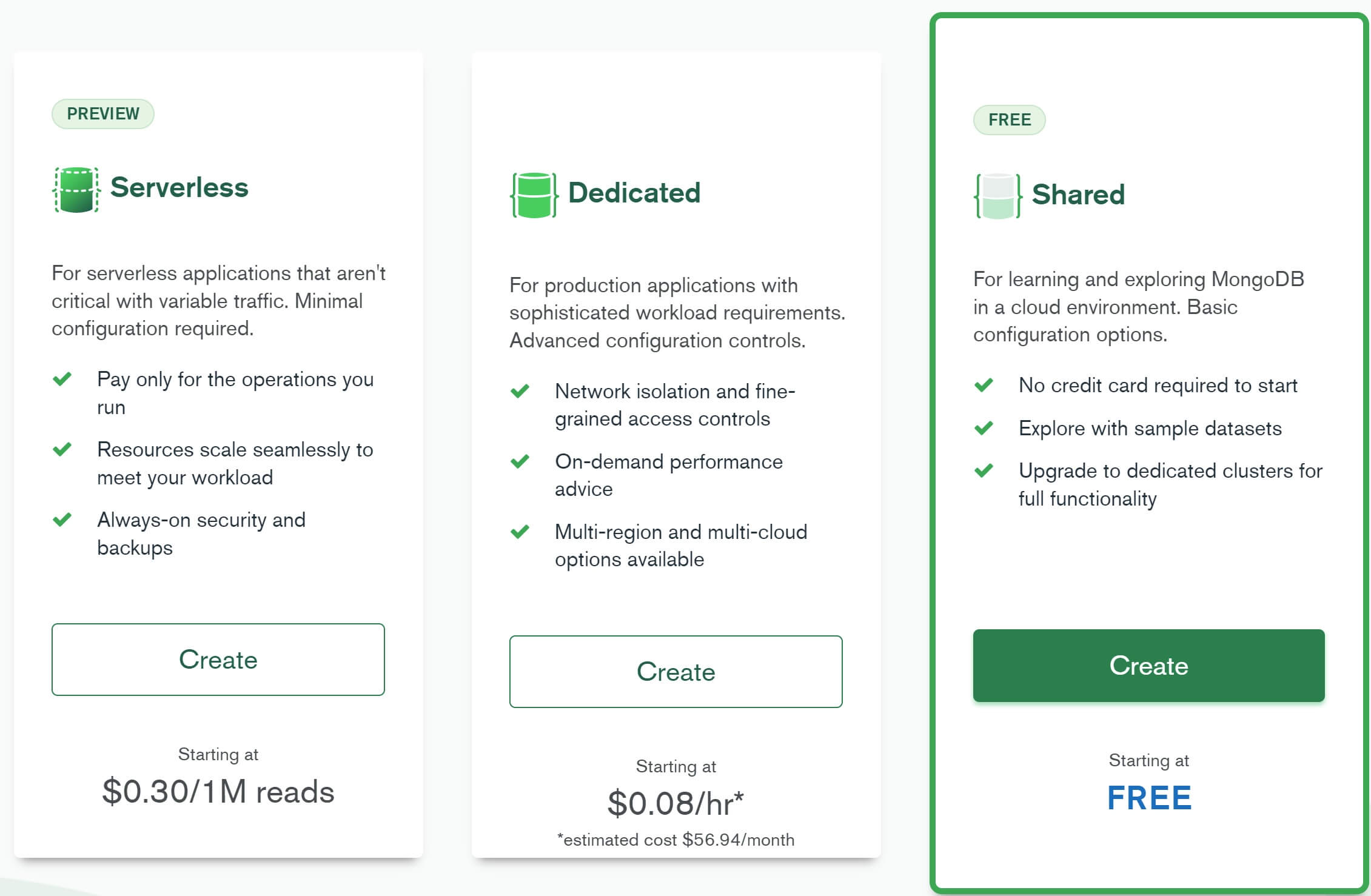
To easily get started with MongoDB, use MongoDB Atlas which is a fully managed cloud database on AWS, Azure, or GCP (Google Cloud Platform). In a web browser, navigate to <https://www.mongodb.com/> and click on the Try Free button. Provide your email address, password and then follow the on-screen instructions to create a free account. You can also use your Google account to register.



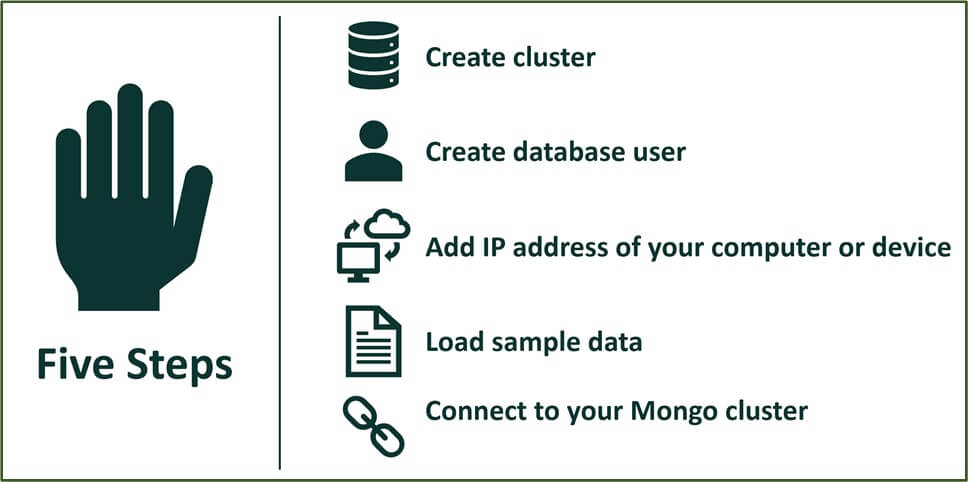
Use coupon code MKT-VENKAT for $100 credit.



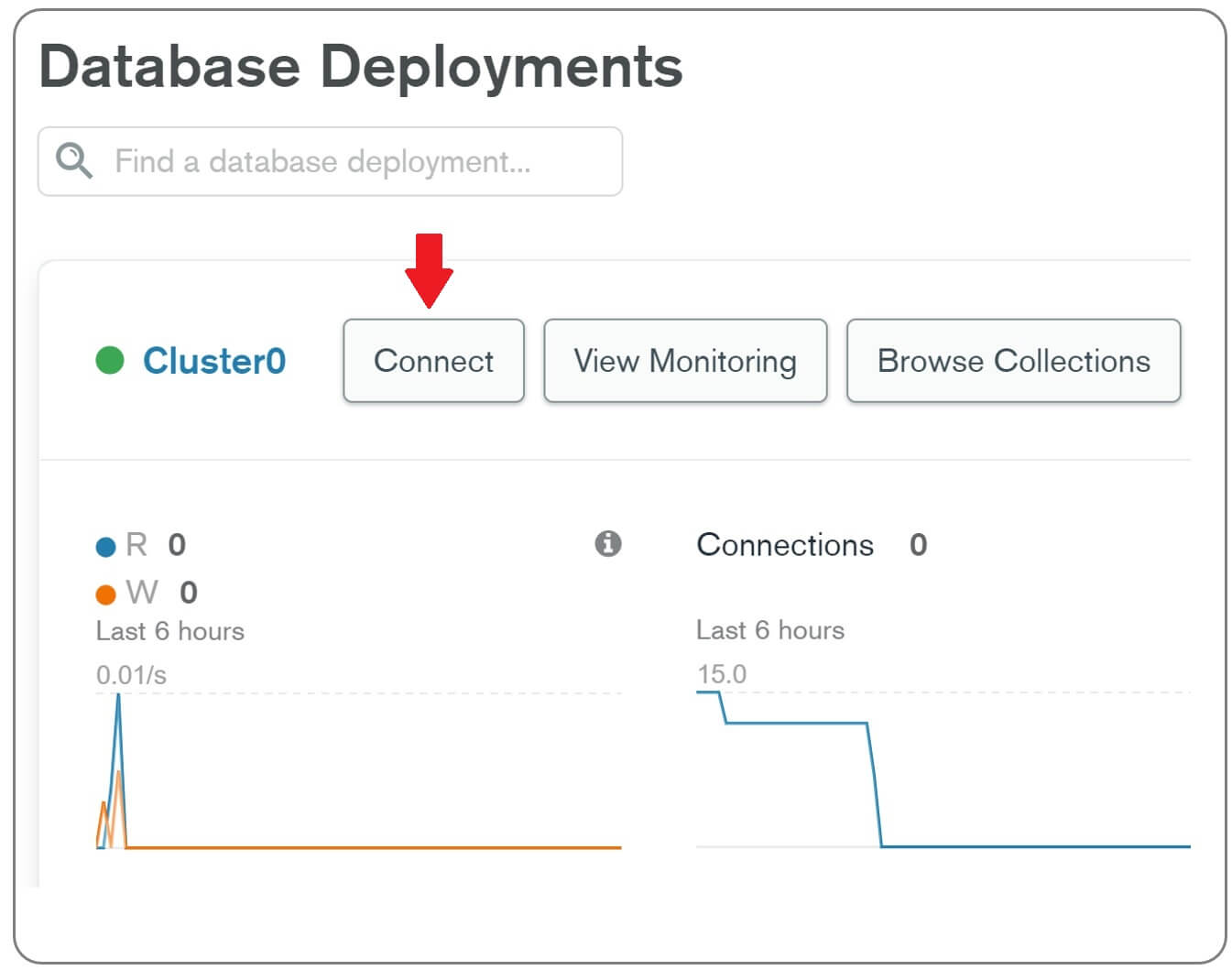
Use the free shared option to learn and explore MongoDB in a cloud environment. With the free option we get 512 MB of storage space, choose our preferred cloud provider - AWS, Google Cloud or Azure and the region where we want the Mongo database instance to be hosted.



The following are the steps to **get started with Mongo cluster in the cloud**.



After the Mongo cluster is created, click on the Connect button.



In the window that appears you have options to

* Create database users and
* Add IP addresses of machines that you want to allow to connect to your Mongo database

## How to connect to MongoDB atlas

There are 3 ways we can connect to MongoDB Atlas.



* MongoDB Shell
* MongoDB Compass
* Custom Application

### MongoDB Shell

MongoDB Shell is an interactive JavaScript interface for querying and modifying database data. We can also use it to perform administrative operations like creating databases, collections etc.

### MongoDB Compass

MongoDB Compass is a GUI (Graphical User Interface) for querying, visualizing and modifying database data. Behind the scenes, both the Shell and Compass issues the same commands to the database server.

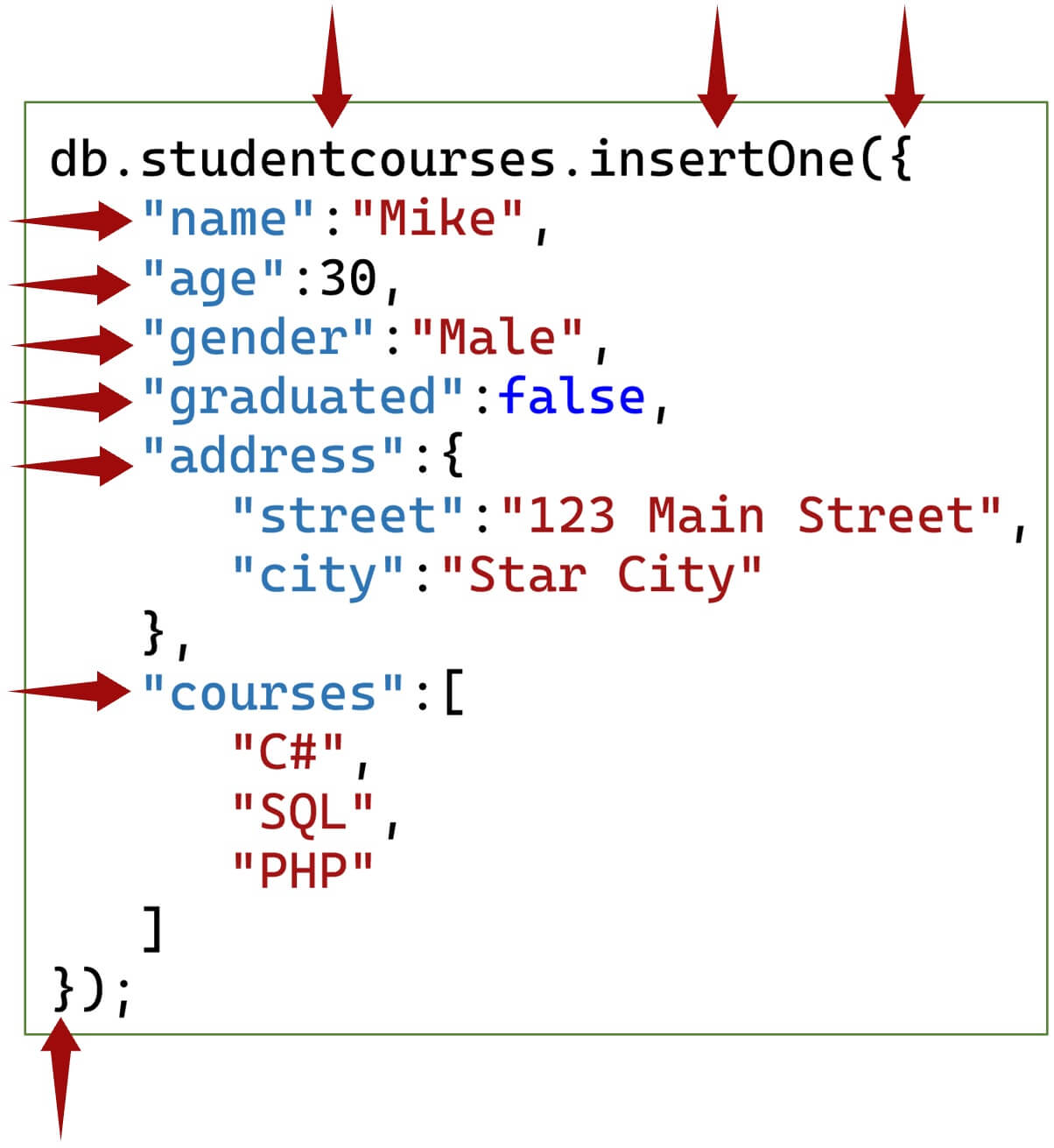
### Your own custom application

You can also connect your own custom application. This application may be built using any technology (.NET, Java, PHP, Node, Python, etc), it doesn't matter.

## Common Mongo Shell Commands

* show dbs : Print a list of all databases on the server.
* use <dbname> : Switch current database to <dbname>.
* db : Displays the current database in use.
* show collections : Prints the list of collections in the current database.
* db.createCollection() : Creates a new collection
* db.collection.drop() : Drops or removes the collection completely.
* db.collection.insertOne() : Insert a new document into the collection.
* db.collection.insertMany() : Insert multiple new documents into the collection.
* db.collection.updateOne() : Update a single existing document in the collection.
* db.collection.updateMany() : Update multiple existing documents in the collection.
* db.collection.deleteOne() : Delete a single document from the collection.
* db.collection.deleteMany() : Delete documents from the collection.
* db.collection.createIndex() : Create a new index on the collection if the index does not exist; otherwise, the operation has no effect.

## db.collection.insertOne() example



## db.collection.insertMany() example

db.studentcourses.insertMany([

{

"name": "Mary",

"age": 28,

"gender": "Female",

"graduated": true,

"address": {

"street": "99 George Street",

"city": "Super City"

},

"courses": [ "ASP.NET", "LINQ", "Angular" ]

},

{

"name": "Pam",

"age": 29,

"gender": "Female",

"graduated": false,

"address": {

"street": "101 King Street",

"city": "Mongo City"

},

"courses": [ ".NET Core", "C++", "Bootstrap" ]

},

{

"name": "Ben",

"age": 31,

"gender": "Male",

"graduated": true,

"address": {

"street": "35 Queen Street",

"city": "London"

},

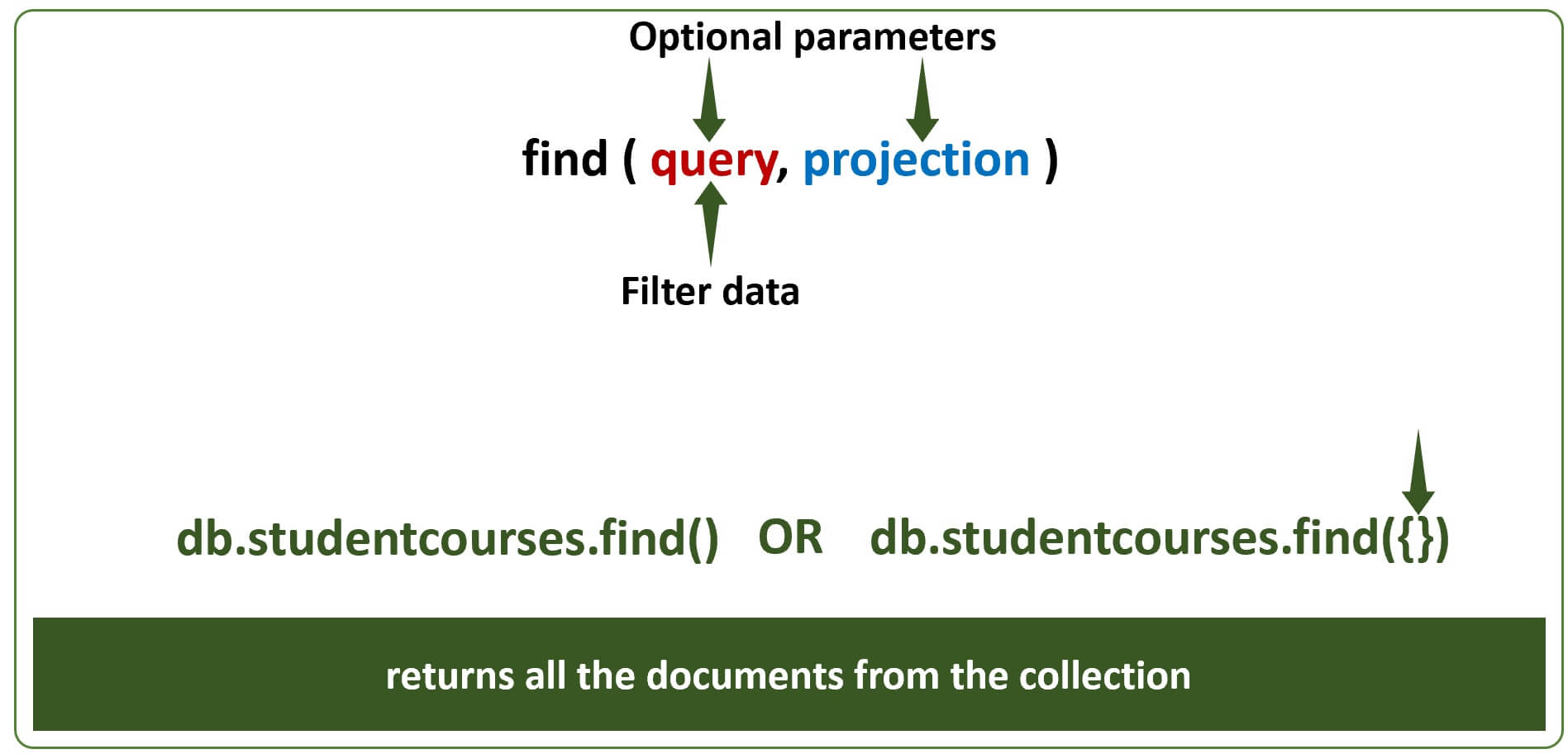
"courses": [ "JavaScript", "C#", "HTML" ]

}

]);

## MongoDB collection find examples

* Use find() method to find documents in a collection.
* find() method has two parameters - query & projecttion.
* Both are optional parameters.
* Use the query parameter to filter data.
* You can think of it as a WHERE clause in SQL.
* find() method returns all the documents in a collection, if we do not include the query parameter or pass an empty object ({}).



#### Find all documents in studentcourses collection where name is Ben

db.studentcourses.find({name: "Ben"})

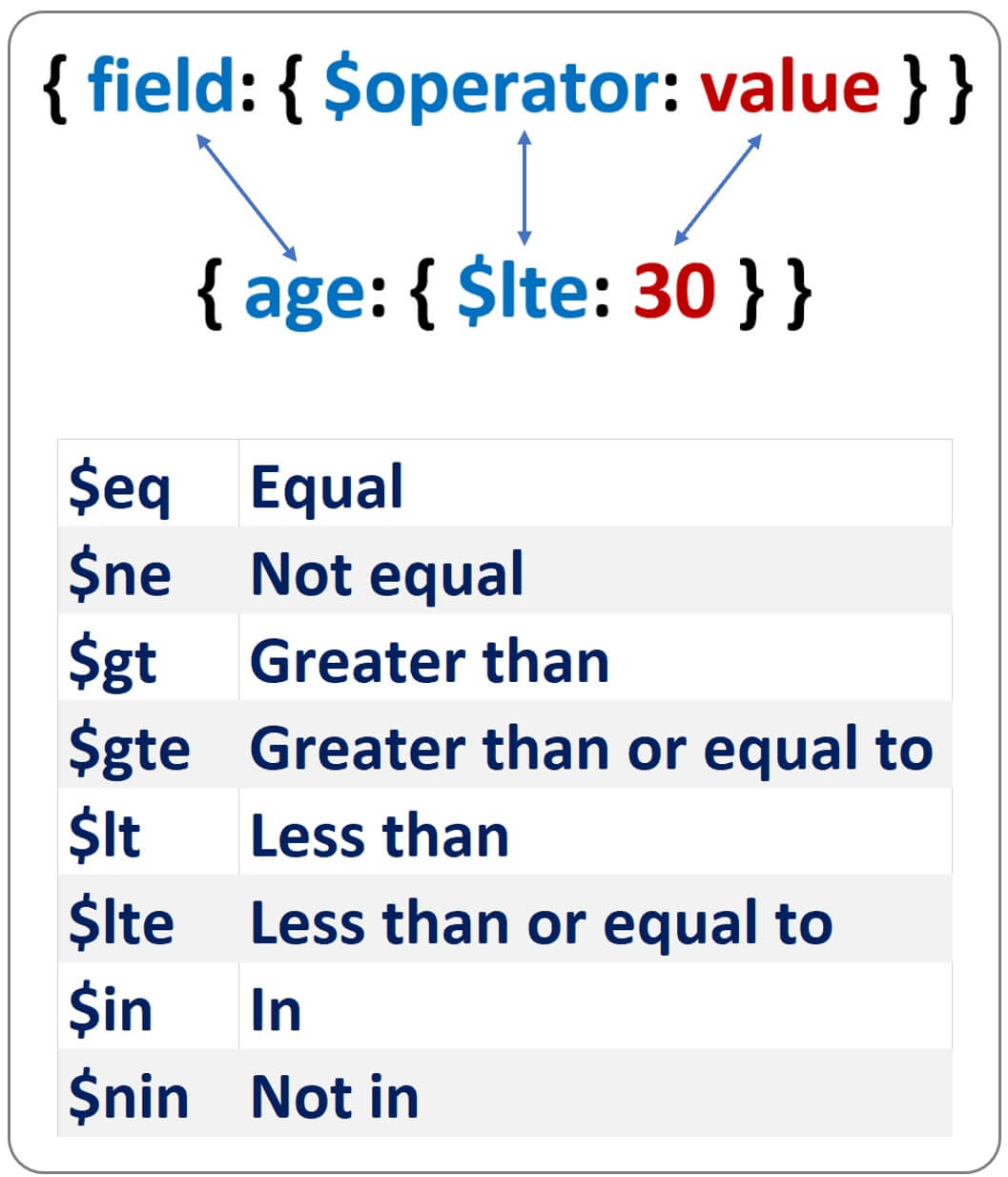
#### Find all documents in studentcourses collection where age is 28

db.studentcourses.find({age: 28})

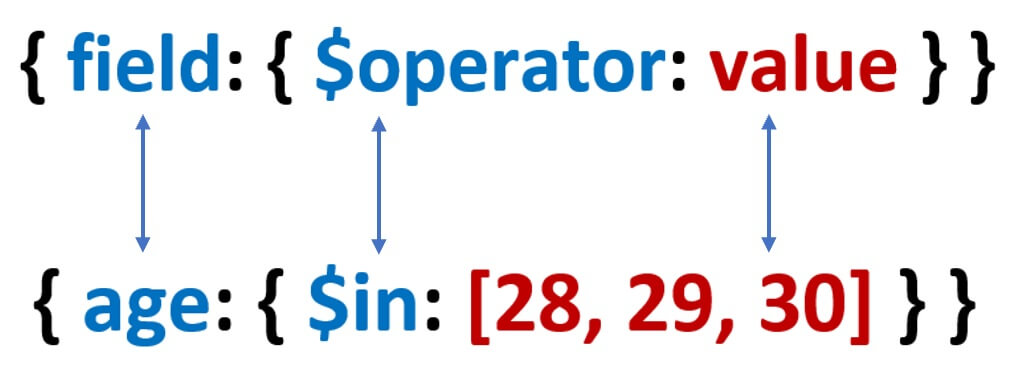
#### Find all documents where age is less then or equal to 30

db.studentcourses.find({age: {$lte: 30}})

## MongoDB query operators examples



#### Find all documents where age in [ 28, 29, 30 ]



db.studentcourses.find({age: {$in: [28, 29, 30]}})

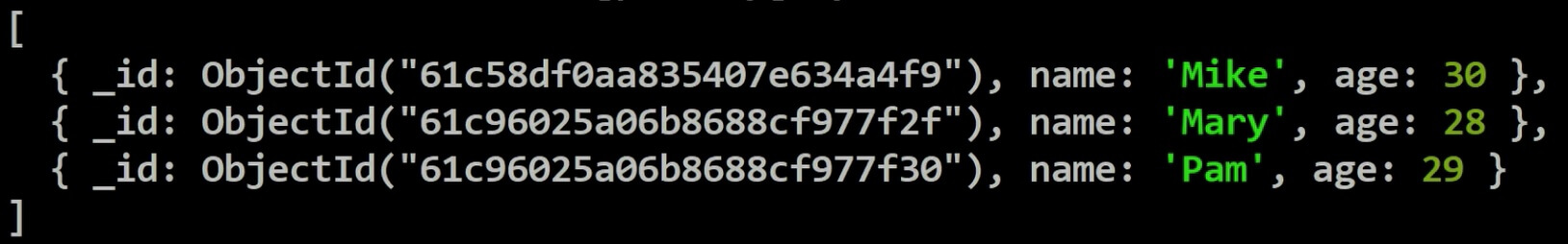
## MongoDB find projection example



* By default, queries in MongoDB return all fields in matching documents.
* To limit the amount of data that MongoDB returns, you can include a projection.
* A projection basically tells what fields to include or exclude in the matching documents.
* Use 1 to include a field and 0 to exclude a field.

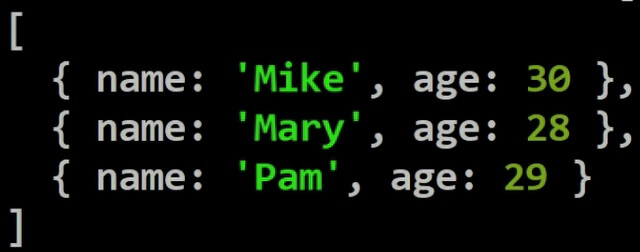
#### Include only name and age fields in the result

db.studentcourses.find({age: {$lte: 30}}, {name:1, age:1})



\_id field is always returned. To exclude it, you have to explicitly specify it using an exclusion projection (\_id:0) as shown below.

db.studentcourses.find({age: {$lte: 30}}, {name:1, age:1, \_id:0})



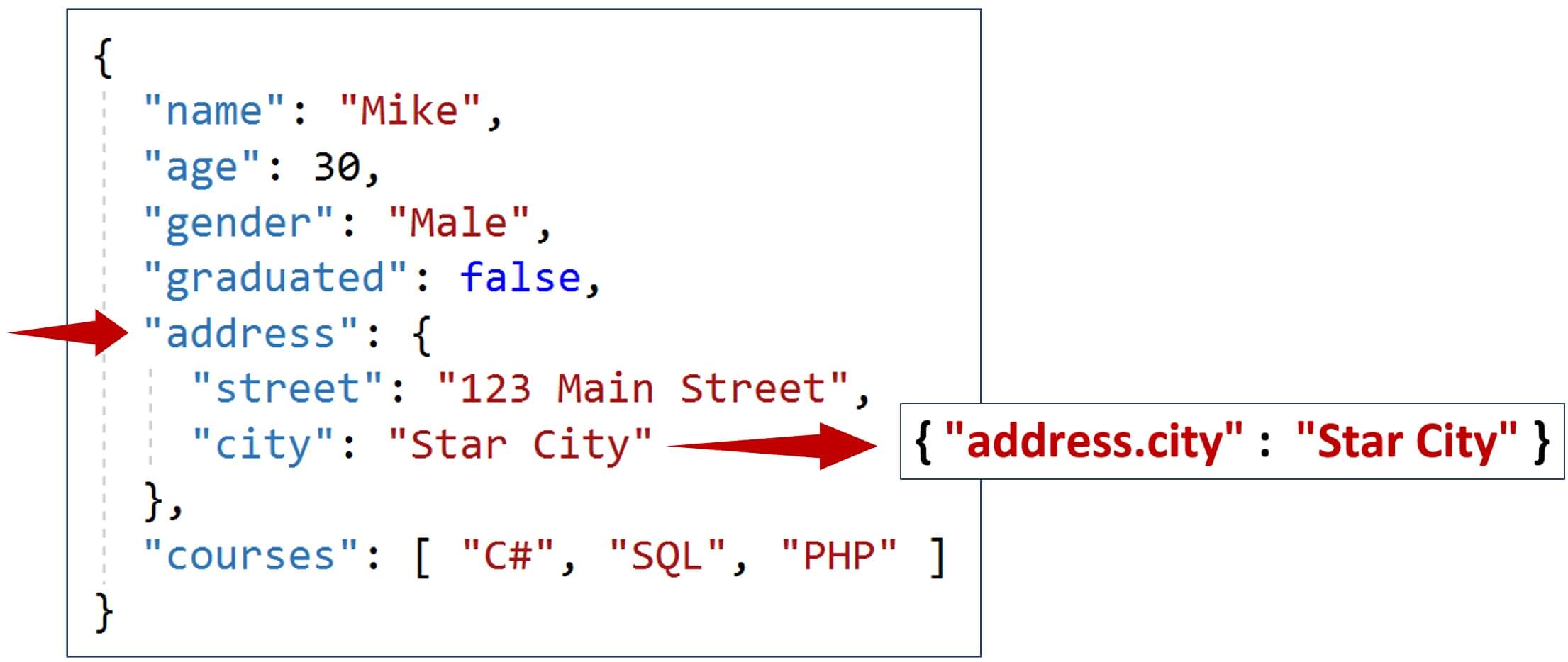
#### Inclusion and exclusion projection

1. You cannot specify the fields that you want to include and exclude at the same time in a projection.
2. You can only specify an inclusion projection or exclusion projection.
3. \_id field is the only exception.

The following projection results in error - Cannot do exclusion on field age in inclusion projection

{name: 1, age: 0}

#### MongoDB find in nested object



* address is a nested JSON document.
* Find all documents where city is Star City.

db.studentcourses.find({"address.city": "Star City"})

#### MongoDB find cursor example



* find method returns a cursor.
* You can think of the cursor as a virtual object that stores the documents returned by the find method.
* So this means along with the find method we can also use methods like count, sort, skip, limit etc.
* These 3 methods - sort, skip and limit are particularly useful to implement server side sorting and paging.

Find the total count of documents in a collection

db.studentcourses.find().count()

Find the total count of documents where gender is Female

db.studentcourses.find({gender: "Female"}).count()

#### Mongodb sort example

* To sort documents in MongoDB use sort() method.
* 1 to sort in ascending order and -1 in descending order.

Sort documents by age field in descending order

db.studentcourses.find().sort({"age":-1})

Sort documents by age field in descending order and then by name in ascending order.

db.studentcourses.find().sort({"age":-1, "name": 1})

Sort by age field in descending order, skip first 1 document and take only the next 2 documents

db.studentcourses.find().sort({"age":-1}).skip(1).limit(2)

#### MongoDB updateOne example

Update name to Sara and age to 100 where name is Mary.

db.studentcourses.updateOne({"name":"Mary"},{$set: {"name":"Sara", "age":100}})

#### MongoDB updateMany example

Update gender in all documents to Unknown where gender is Female

db.studentcourses.updateMany({"gender":"Female"},{$set: {"gender":"Unknown"}})

#### MongoDB deleteOne example

Delete one document where name is Sara

db.studentcourses.deleteOne({"name":"Sara"})

#### MongoDB deleteMany example

Delete all document where age is less than 31

db.studentcourses.deleteMany({"age":{$lt:31}})

## MongoDB Compass Commands

#### Find all documents in a collection

{}

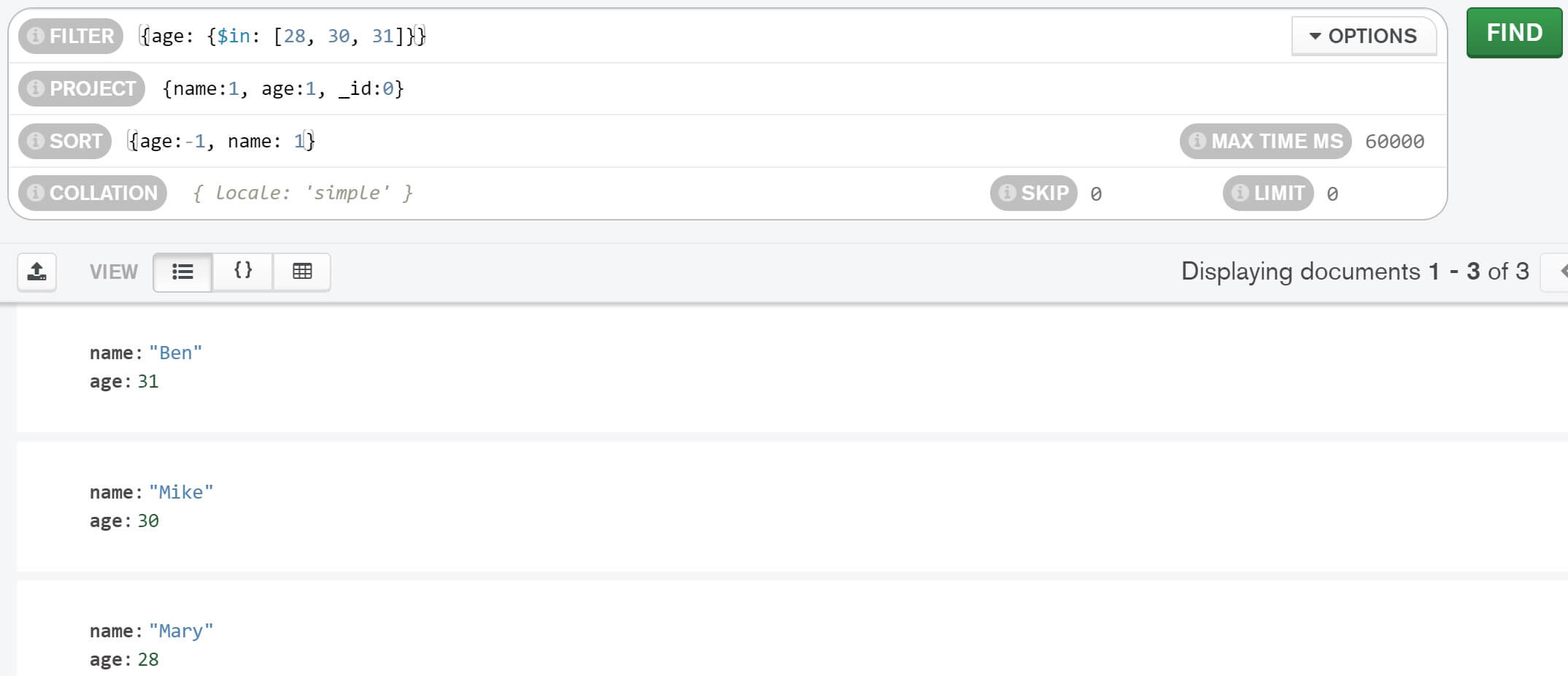
#### Find all documents where name = "Venkat"

{name: "Venkat"}

The above query can also be written as shown below. In this example we are using the equal ($eq) query operator.

{name: {$eq: "Venkat"}}

Filter, project and sort example - All in one query



## Summary

* MongoDB Atlas - Mongo Database in the cloud (Azure, AWS or GCP).
* MongoDB Shell - CLI (Command Line Interface) interface to write and test Mongo database queries.
* MongoDB Compass - Graphical User Interface to connect to and work with Mongo databases. It is the easiest way to interact with your data in MongoDB. Under the hood, Compass uses Mongo Shell.