Mongo DB

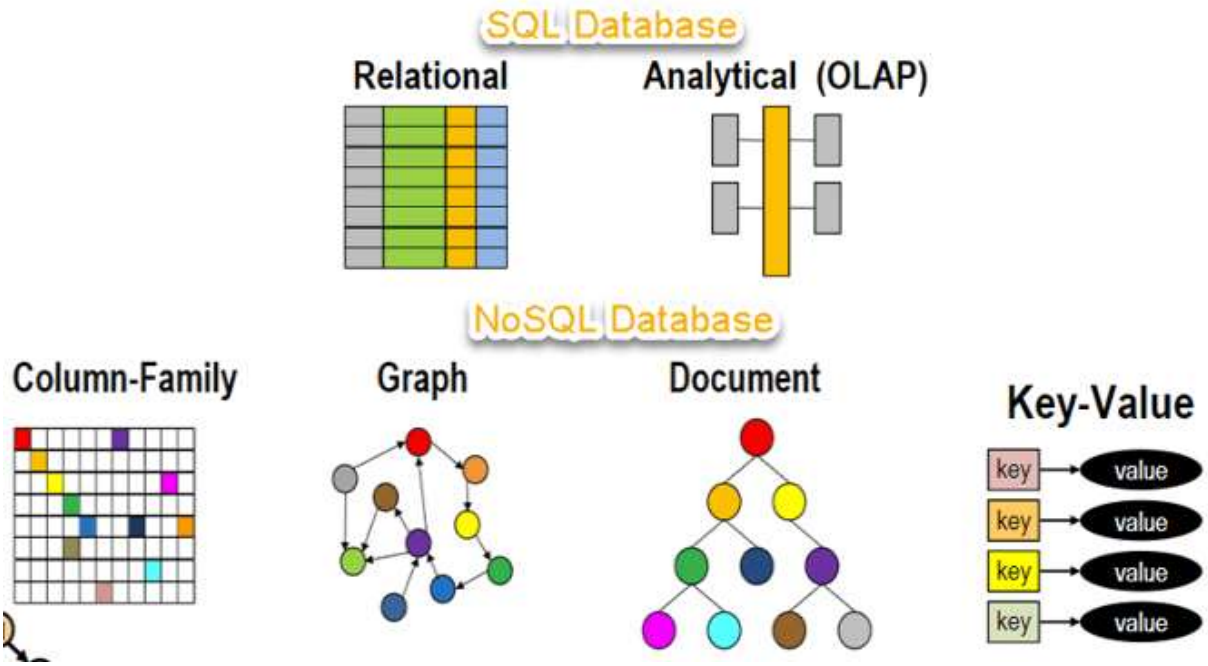
SL.No	INDEX	Pg.No
	NoSQL	
1.	What is NoSQL	2
2.	Why NoSQL?	2
3.	Features of NoSQL	3
	a.Non-relational	3
	b.Schema free	3
	c.Simple API	3
4.	Types of NoSQL Databases:	4
	MongoDB	5
5.	Why MongoDB	5
6.	Download MongoDB Community Edition.	6
7.	Dependency needed:	7
8.	Create the Repository	8
9.	Differences between MongoDB and RDBMS	9
10.	Project work:	9

What is NoSQL?

NoSQL is a non-relational DMS, that does not require a fixed schema, avoids joins, and is easy to scale. NoSQL database is used for distributed data stores with humongous data storage needs. NoSQL is used for Big data and real-time web apps. For example companies like Twitter, Facebook, Google that collect terabytes of user data every single day.

NoSQL database stands for "Not Only SQL" or "Not SQL."

Traditional RDBMS uses SQL syntax to store and retrieve data for further insights. Instead, a NoSQL database system encompasses a wide range of database technologies that can store structured, semi-structured, unstructured and polymorphic data.

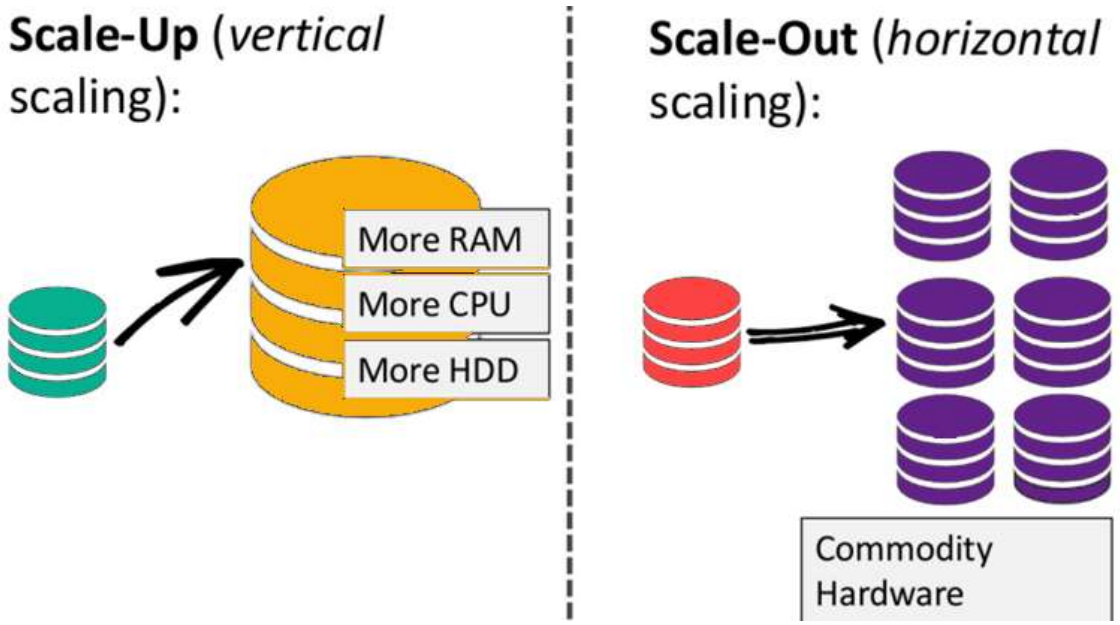


Why NoSQL?

The concept of NoSQL databases became popular with Internet giants like Google, Facebook, Amazon, etc. who deal with huge volumes of data. The system response time becomes slow when you use RDBMS for massive volumes of data.

To resolve this problem, we could "scale up" our systems by upgrading our existing hardware. This process is expensive.

The alternative for this issue is to distribute database load on multiple hosts whenever the load increases. This method is known as "scaling out."



NoSQL database is non-relational, so it scales out better than relational databases as they are designed with web applications in mind.

Features of NoSQL

Non-relational

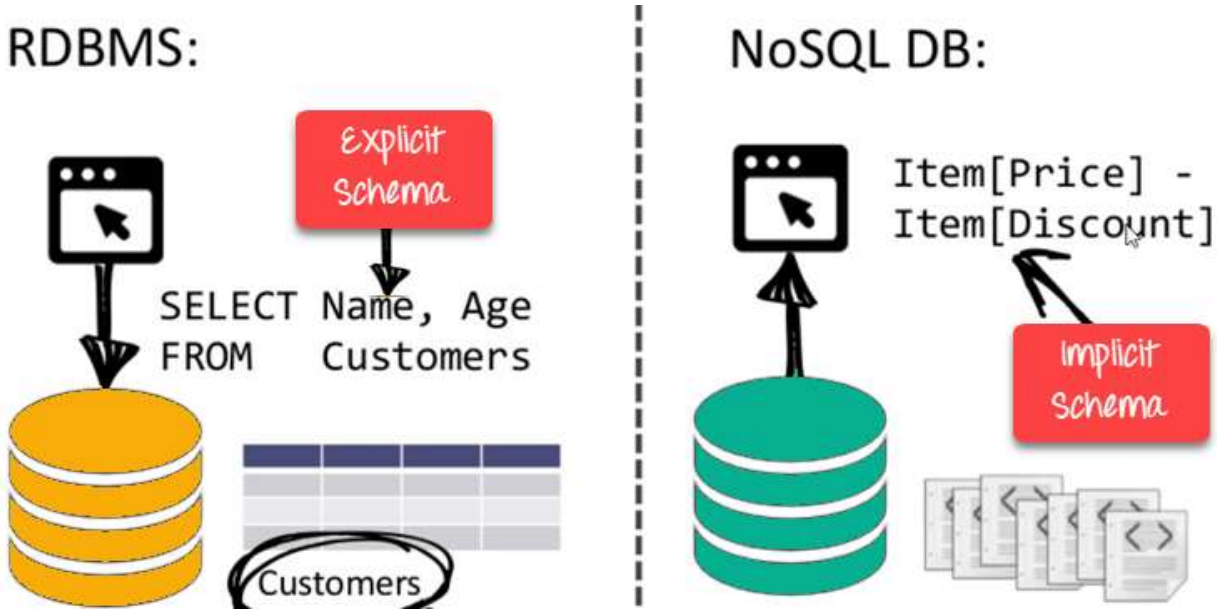
- NoSQL databases never follow the relational model
- Never provide tables with flat fixed-column records
- Work with self-contained aggregates or BLOBs
- Doesn't require object-relational mapping and data normalization
- No complex features like query languages, query planners, referential integrity joins, ACID

Schema-free

- NoSQL databases are either schema-free or have relaxed schemas
- Do not require any sort of definition of the schema of the data
- Offers heterogeneous structures of data in the same domain

Simple API

- Offers easy to use interfaces for storage and querying data provided
- APIs allow low-level data manipulation & selection methods
- Text-based protocols mostly used with HTTP REST with JSON
- Mostly used no standard based query language



NoSQL is Schema-Free

Types of NoSQL Databases:

There are mainly four categories of NoSQL databases. Each of these categories has its unique attributes and limitations. No specific database is better to solve all problems. You should select a database based on your product needs.

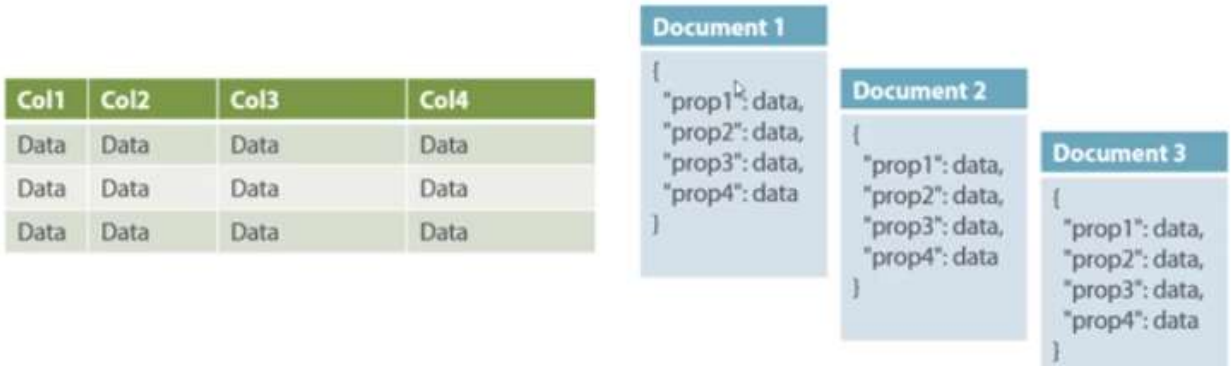
Let see all of them:

- Key-value Pair Based
- Column-oriented Graph
- Graphs based
- Document-oriented

We are using Document Oriented database

Document-Oriented:

Document-Oriented NoSQL DB stores and retrieves data as a key value pair but the value part is stored as a document. The document is stored in JSON or XML formats. The value is understood by the DB and can be queried.



Relational Vs. Document

In this diagram on your left you can see we have rows and columns, and in the right, we have a document database which has a similar structure to JSON. Now for the relational database, you have to know what columns you have and so on. However, for a document database, you have data store like JSON object. You do not require to define which make it flexible.

MONGODB

MongoDB is an open-source NoSQL document database that uses a JSON-like schema instead of traditional table-based relational data. MongoDB is a document-oriented NoSQL database used for high volume data storage. Spring Boot offers several conveniences for working with MongoDB, including the spring-boot-starter-data-mongodb 'Starter'. Spring Data includes repository support for MongoDB.

Why MongoDB

Mongo is quickly growing in popularity among NoSQL databases. MongoDB, specifically, is perfect for developing a REST API with Spring Boot for a couple of key reasons:

- **Document Oriented Storage** − Data is stored in the form of JSON style documents.
- Index on any attribute
- Replication and high availability
- Auto-sharding
- Rich queries
- Fast in-place updates
- Professional support by MongoDB

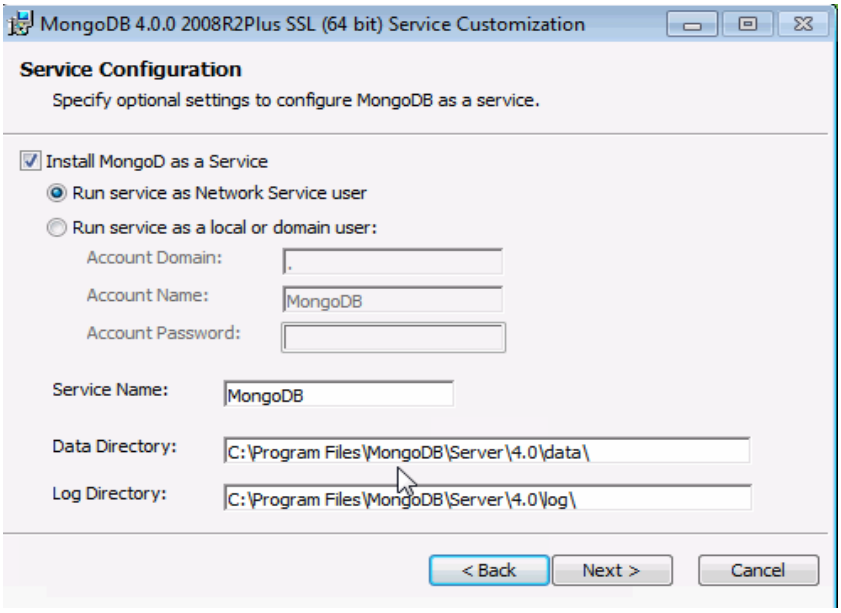
Where to Use MongoDB?

- Big Data
- Content Management and Delivery
- Mobile and Social Infrastructure
- User Data Management
- Data Hub

Download MongoDB Community Edition.

Download the installer (.msi) from the MongoDB Download Center:

a. The Download Center should display **MongoDB Community Server** download information. If not, select **Server**, then click the **MongoDB Community Server** tab.
b. In the **Version** dropdown, select the version that corresponds to the latest MongoDB Server 4.0.
c. In the **OS** dropdown, **Windows 64-bit X64** should be selected.
d. In the **Package** drop down, **MSI** should be selected.
e. Click **Download**.
f. Accept the End user License Agreement and click next.
g. Click on either the **Complete** (recommended for most users) or **Custom** setup type. If you choose the **Custom** installation option, you may specify which executables are installed and where. Select complete option.
h. Select "Run Service as Network Service User"



i. Click on install to start installation
j. Finally click on finish

If You Installed MongoDB as a Service

The MongoDB service is started upon successful installation [1].

To begin using MongoDB, connect a mongo.exe shell to the running MongoDB instance. Either:

- From Windows Explorer/File Explorer, go to C:\Program Files\MongoDB\Server\4.0\bin\directory and double-click on mongo.exe.
- Or, open a **Command Interpreter** with Administrative privileges and run:

```
"C:\Program Files\MongoDB\Server\4.0\bin\mongo.exe"
```

TOOLS USED :

1. Java
2. SpringBoot
3. Maven
4. MongoDB
5. Postman

Adding the MongoDB Connection Info

To tell Spring the connection information for our MongoDB, we will need to add conection details to the application.properties file, located in the "src/main/resources" folder.

```
spring.data.mongodb.host=localhost
spring.data.mongodb.port=27017
spring.data.mongodb.authentication-database=test
server.port=8095
```

Dependency needed:

```
<dependencies>
		<dependency>
			<groupId>org.springframework.boot</groupId>
			<artifactId>spring-boot-starter-datamongodb</artifactId>
		</dependency>
		<dependency>
			<groupId>org.springframework.boot</groupId>
			<artifactId>spring-boot-starter-web</artifactId>
		</dependency>

		<dependency>
			<groupId>org.springframework.boot</groupId>
			<artifactId>spring-boot-starter-test</artifactId>
			<scope>test</scope>
		</dependency>
	</dependencies>
```

Create the Repository

Now, after the configuration, we need to create a repository – extending the existing *MongoRepository* interface:

```
public interface EmployeeRepository extends MongoRepository<Employee, String> {
   //
}
```

Now we can auto-wire this *EmployeeRepository* and use operations from *MongoRepository* or add custom operations.

Below are some of the key term differences between MongoDB and RDBMS

RDBMS	MongoDB	Difference
Table	Collection	In RDBMS, the table contains the columns and rows which are used to store the data whereas, in MongoDB, this same structure is known as a collection. The collection contains documents which in turn contains Fields, which in turn are key-value pairs.
Row	Document	In RDBMS, the row represents a single, implicitly structured data item in a table. In MongoDB, the data is stored in documents.
Column	Field	In RDBMS, the column denotes a set of data values. These in MongoDB are known as Fields.
Joins	Embedded documents	In RDBMS, data is sometimes spread across various tables and in order to show a complete view of all data, a join is sometimes formed across tables to get the data. In MongoDB, the data is normally stored in a single collection, but separated by using Embedded documents. So, there is no concept of joins in MongoDB.

JPA	MongoDB

`@Entity`
`@Table(name="TUSR")`
`public class User {`
 	
`  @Id`
`  private String id;`
 	
`  @Column(name="fn")`
`  private String name;`
 	
`  private Date lastLogin;`
 	
`...`
`}`
	`@Document(`
	`collection="usr")`
	`public class User {`
	
	`  @Id`
	`  private String id;`
	
	`  @Field("fn")`
	`  private String name;`
	
	`  private Date lastLogin;`
	
	`...`
	`}`

Project work:

1.create connection by adding properties in application.properties

2.Add dependencies

3. Create project hierarchy by creating packages and subpackages

Ex: com.cg.<artificatId>(Root package),

com.cg.<artificatId>.pojo

com.cg.<artificatId>.controller and

com.cg.<artificatId>.repository

- Create main class in root package
- Now, Rightclick on project-> Maven->UpdateProject

4.create repository extending MongoRepository

5.Run the application and hit the details in postman automatically data will be created in collection database.