NOSQL

MongoDB

NOSQL

Agenda

- NOSQL Patterns
- MongoDB
- Live Examples
- Features
- CAP theorem
- Java Integration

NOSQL

NoSQL Patterns

- Key-Value
- Graph database
- Document-oriented
- Column family

NOSQL

NoSQL Patterns

Key-Value

Concept: Data stored as key–value pairs (like a dictionary or hash map).

Strengths:

- Extremely fast lookups
- High scalability
- Simple data model

Use Cases: Caching, session management, user profiles

NOSQL

NoSQL Patterns

Graph

Concept: Data represented as **nodes, edges, and properties**.

Strengths:

- Optimized for relationships & traversals
- Ideal for queries like “shortest path”
- Schema-less

Use Cases: Social networks, recommendation engines

NOSQL

NoSQL Patterns

Document

Concept: Data stored in **documents** (JSON/BSON/XML-like).

Strengths:

- Schema flexibility
- Rich queries and indexing
- Natural mapping to application objects

Use Cases: Content management, catalogs, IoT, user data

NOSQL

NoSQL Patterns

Column

Concept: Data stored in **columns instead of rows** (optimized for large-scale analytics).

Strengths:

- Excellent for aggregations and analytical queries
- High write throughput
- Scales to petabytes

Use Cases: Big data, time-series data, recommendation systems

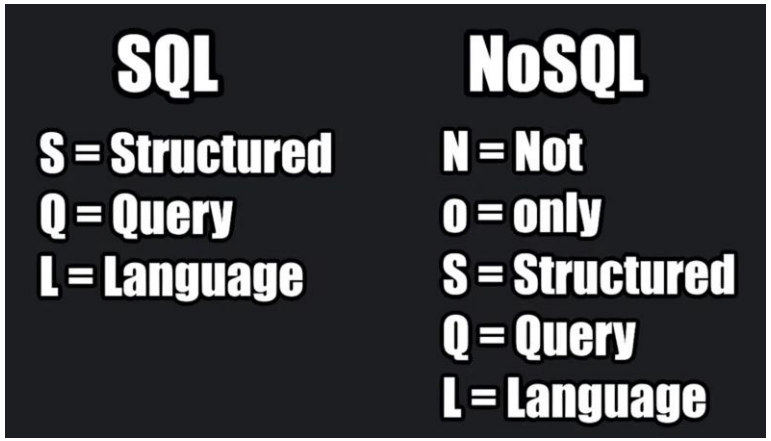
NOSQL

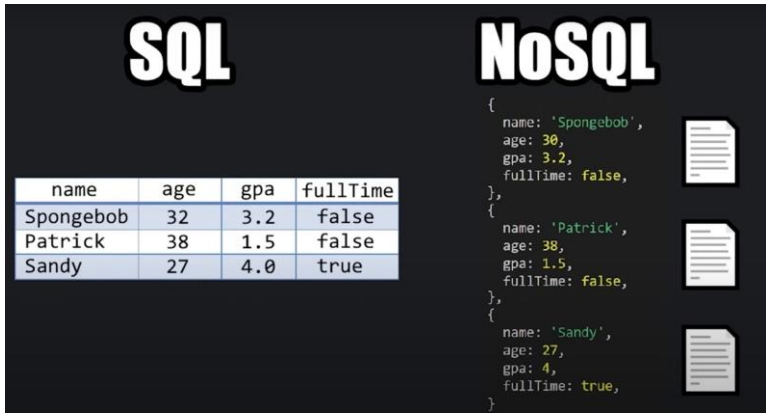
NoSQL Patterns

Type	Data Model	Best For	Examples
Key-Value	Key → Value	Caching, sessions	Redis, DynamoDB
Document	JSON/BSON docs	Flexible schema apps	MongoDB, CouchDB
Column-Family	Columns/rows	Analytics, big data	Cassandra, HBase
Graph	Nodes & Edges	Relationships	Neo4j, Neptune
Multi-Model	Hybrid	Mixed workloads	Cosmos DB, ArangoDB

NOSQL

SQL VS NOSQL





RDBMS		MongoDB
Database	➡	Database
Table, View	➡	Collection
Row	➡	Document (JSON, BSON)
Column	➡	Field
Index	➡	Index
Join	➡	Embedded Document
Foreign Key	➡	Reference
Partition	➡	Shard

NOSQL

MongoDB

What is MongoDB

- Open-source, document-oriented NoSQL database
- Uses JSON-like documents (BSON)
- Provides high performance, scalability, and flexibility
- First released in 2009 by MongoDB Inc

NOSQL

MongoDB

Core Concepts

- **Database** → holds collections
- **Collection** → holds documents
- **Document** → JSON-like objects { key: value }
- **_id** → unique identifier for each document

NOSQL

MongoDB

Document store

```
> db.user.findOne({age:39})
{
        "_id" :
ObjectId("5114e0bd42…"),
        "first" : "John",
        "last" : "Doe",
        "age" : 39,
        "interests" : [
                "Reading",
                "Mountain Biking ]
        "favorites": {
                "color": "Blue",
                "sport": "Soccer"}
}
```

NOSQL

MongoDB

CRUD

- Create
  - db.collection.insert(<document>)
  - db.collection.save(<document>)
  - db.collection.update(<query>, <update>, { upsert: true })
- Read
  - db.collection.find(<query>, <projection>)
  - db.collection.findOne(<query>, <projection>)
- Update
  - db.collection.update(<query>, <update>, <options>)
- Delete
  - db.collection.remove(<query>, <justOne>)

NOSQL

MongoDB

Create Read

```
> db.user.insert({
        first: "John",
        last : "Doe",
        age: 39
})
```

```
> db.user.find ()
{
        "_id" : ObjectId("51..."),
        "first" : "John",
        "last" : "Doe",
        "age" : 39
}
```

NOSQL

MongoDB

Update Remove

```
> db.user.update(
        {"_id" : ObjectId("51…")},
        {
            $set: {
                age: 40,
                salary: 7000}
        }
)
```

```
> db.user.remove({
        "first": /^J/
})
```

NOSQL

MongoDB

- **Live Examples**

MongoDB Compass

NOSQL

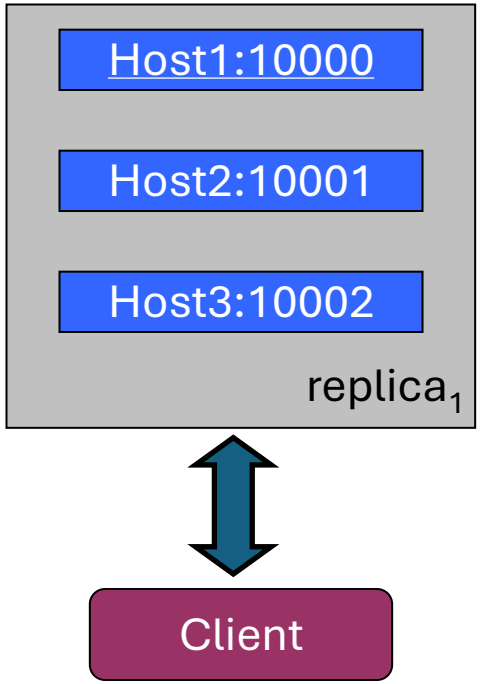
Features

- Document-Oriented Storage
- Full Index Support
- Replication & High Availability
- Auto-Sharding
- Querying
- Fast In-Place Updates
- Map/Reduce

NOSQL

Features

- Redundancy and Failover
- Zero downtime for upgrades and maintaince



Replica Sets

- Master-slave replication
  - Strong Consistency
  - Delayed Consistency

- Geospatial features

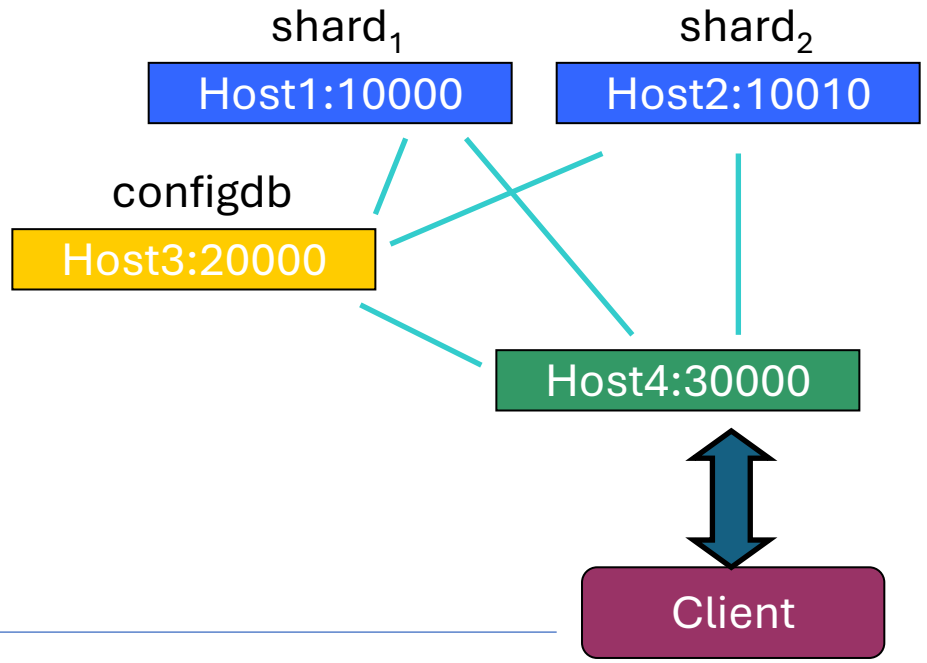
NOSQL

Features

id	company	customer	article	currency	price
4250250	020	073000	5994537812	00	142,50
4250251	020	073000	5994537852	00	141,12
4250252	020	073000	5994537854	00	105,99
4250253	020	073000	5994537856	00	109,52
4250254	020	073000	5994537862	00	131,49
4250255	020	073000	5994567308	00	29,86
4250256	020	073000	5994567422	00	57,13
4250257	020	073000	5994567428	00	68,59
4250258	020	073000	5994605089	00	51,09
4250259	020	073000	5994607975	00	93,93
4250260	020	073000	5994701005	00	74,22

Sharding

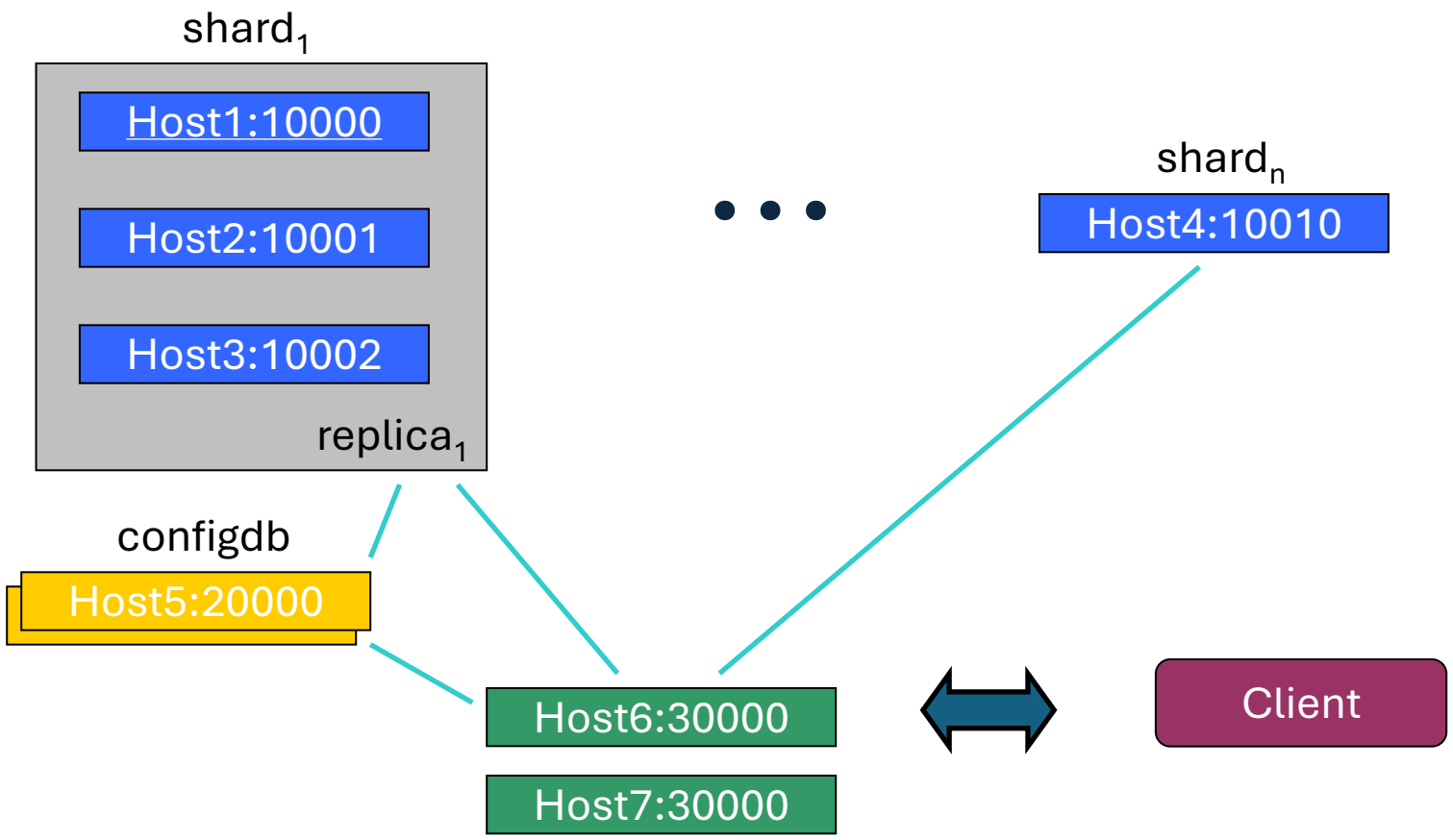
- Partition your data
- Scale write throughput
- Increase capacity
- Auto-balancing



NOSQL

Features

Mixed

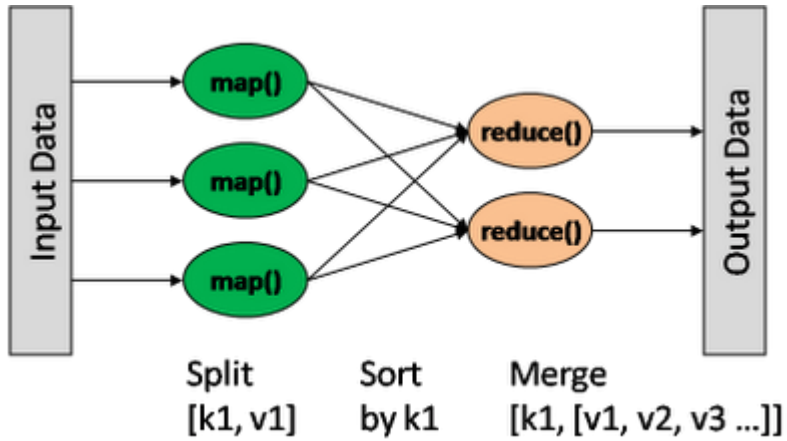


NOSQL

Features

Map/Reduce

```
db.collection.mapReduce(
        <mapfunction>,
        <reducefunction>,
        {
                out: <collection>,
                query: <>,
                sort: <>,
                limit: <number>,
                finalize: <function>,
                scope: <>,
                jsMode: <boolean>,
                verbose: <boolean>
        }
)
```




```
var mapFunction1 = function() { emit(this.cust_id, this.price); };

var reduceFunction1 = function(keyCustId, valuesPrices)
{ return sum(valuesPrices); };
```

NOSQL

Features

- Easy to install and use
- Detailed documentation
- Various APIs
  - JavaScript, Python, Ruby, Perl, Java, Scala, C#, C++, Haskell, Erlang
- Community
- Open source

Other features

NOSQL

CAP theorem

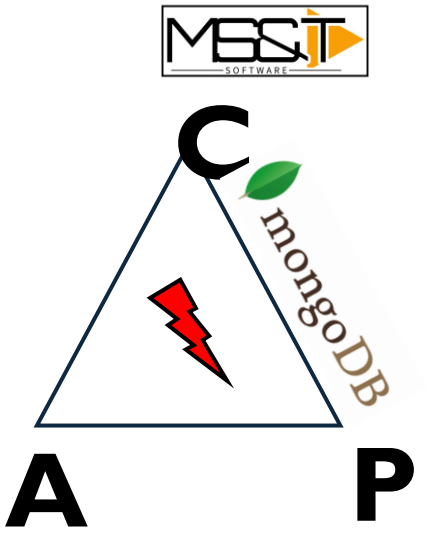
ACID - BASE

- **A**tomicity
- **C**onsistency
- **I**solation
- **D**urability

- **B**asically **A**vailable (CP)
- **S**oft-state
- **E**ventually consistent (AP)

NOSQL

CAP theorem



- Many nodes: Nodes contain *replicas of partitions* of data
- **C**onsistency
  - all replicas contain the same version of data
- **A**vailability
  - System remains operational on failing nodes
- **P**artition tolarence
  - multiple entry points
  - system remains operational on system split

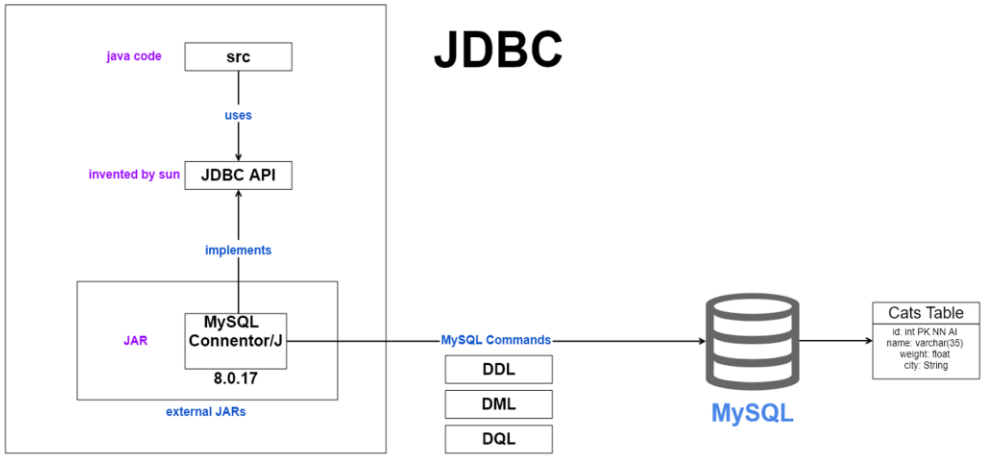
C-A-P

CAP Theorem: satisfying all three at the same time is impossible

NOSQL

Java integration

JDBC



- **Definition:** JDBC (Java Database Connectivity)-is a standard Java API that allows Java programs to connect and interact with databases.
- **Purpose:** Enables Java applications to execute SQL statements, retrieve data, and manage database transactions.

NOSQL

Java integration

JDBC

JDBC Architecture?

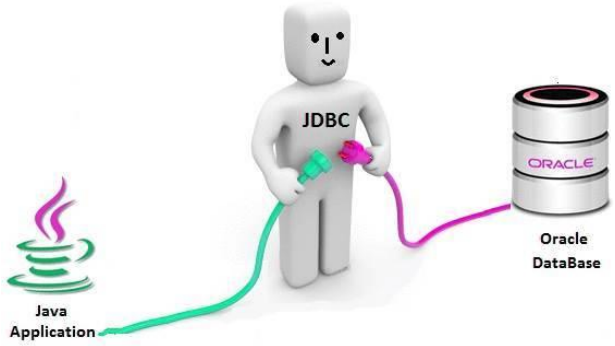
Components:

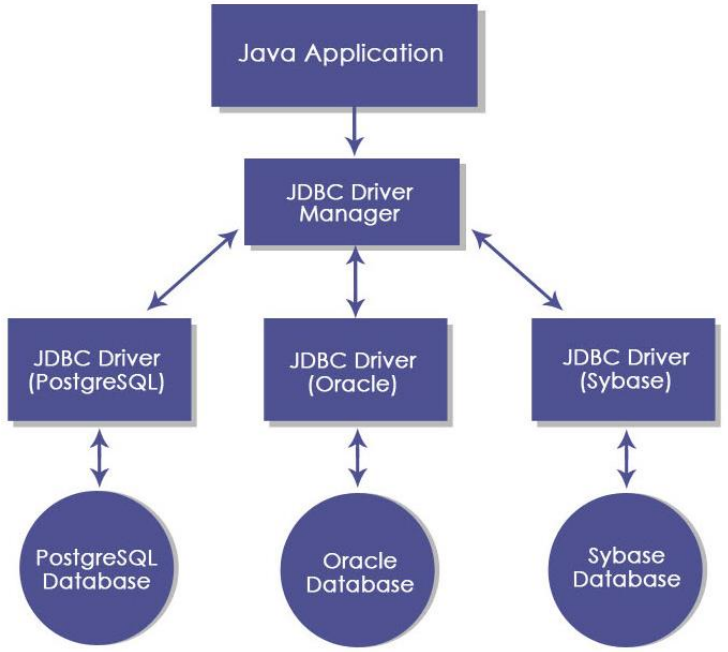
- **Driver Manager:** Manages a list of database drivers.
- **Driver:** Handles communication with the database.
- **Connection:** Represents a connection to the database.
- **Statement:** Used to execute SQL queries.
- **ResultSet:** Represents the result set of a query.

NOSQL

Java integration

JDBC





NOSQL

Java integration

JDBC

Types of JDBC Drivers

- JDBC-ODBC Bridge Driver.
- Native-API Driver (Partially Java Driver).
- Network Protocol Driver (Middleware Driver).
- Thin Driver (Pure Java Driver).

NOSQL

JDBC

Java integration

Setup IntelliJ IDE to use MySQL JDBC Driver:

- Download the latest MySQL JDBC Driver
  - https://dev.mysql.com/downloads/connector/j/
  - Select: Platform Independent
  - Click the: 'Download' button (e.g. the ZIP file)
  - Click: 'No thanks, just start my download.'
- Extract the zip file, we only need the jar inside
- Copy the jar to your 'lib' folder in your java project
- Configure the jars in your IDE, e.g. for IntelliJ go to:
  - File → Project Strucuture → Modules → Dependencies
  - Click the '+' button → JARs or directories…
  - Select the relevant jar under the lib/ folder
  - Confirm
- Write java code

NOSQL

Java integration

JDBC

Establishing a Connection

- Load the JDBC driver.
- Establish a connection using DriverManager.getConnection().
- Example code snippet.

NOSQL

Java integration

JDBC

Executing SQL Queries

- **Statement:** For simple SQL statements.
- **PreparedStatement:** For parameterized queries.
- **CallableStatement:** For calling stored procedures.

NOSQL

Java integration

JDBC

- Result Set and Data Retrieval
- **ResultSet Methods:** next(), getString(), getInt(), etc.
- **Code Example:** Demonstrate how to iterate through a ResultSet to retrieve data

NOSQL

Java integration

JDBC

Managing Transactions

Transactions:

- **Commit:** Save changes.
- **Rollback:** Undo changes.

Auto-Commit: Explain auto-commit mode and how to disable it.

NOSQL

Java integration

MongoDB Java Integration

- Download the latest MongoDB Java Drivers files
  - mongodb-driver-sync-5.5.1.jar
  - mongodb-driver-core-5.5.1.jar
  - bson-5.5.1.jar
- Copy the jars to your ‘lib’ folder in your java project
- Configure the jars in your IDE, e.g. for IntelliJ go to:
  - File → Project Strucuture → Modules → Dependencies
  - Click the ‘+’ button → JARs or directories…
  - Select all the jars under the lib/ folder
  - Confirm
- Write java code

NOSQL

Spring Boot + MySQL
(Relational DB)

Java integration

- Add dependency
- Add spring-boot-starter-data-jpa and mysql-connector-java in pom.xml.
- Configure DB connection
- Add database details in application.properties / application.yml:
- spring.datasource.url, spring.datasource.username, spring.datasource.password
- spring.jpa.hibernate.ddl-auto (optional: update, create, validate)
- Define an Entity
- Create a class annotated with @Entity to map to a table.
- Create a Repository
- Define an interface extending JpaRepository<Entity, ID>.
- Use it in your Service/Controller
- Spring Data JPA provides CRUD methods automatically.

NOSQL

Spring Boot + MongoDB (NoSQL)

Java integration

- Add dependency
- Add spring-boot-starter-data-mongodb in pom.xml.
- Configure DB connection
- In application.properties / application.yml:
- spring.data.mongodb.uri = mongodb://localhost:27017/dbname
- Define a Document
- Create a class annotated with @Document(collection = "name").
- Create a Repository
- Define an interface extending MongoRepository<Document, ID>.
- Use it in your Service/Controller
- Spring Data MongoDB provides ready-made CRUD as well.

NOSQL

Thank You !!