NoSQL: NOT ONLY SQL or NOT SQL

- NoSQL is a non-relational database that uses key-value pairs to store and retrieve data unlike a relational database which stores data in rows and columns.
- A non-relational database is a collection of unstructured and semi structured items which do not store data in tabular form.

Types:

1. Key-Value Based (Redis)
2. Column Based (Cassandra)
3. Document Based (MongoDB)
4. Graph Based (Neo4J)

SQL V/S NoSQL:

SQL	NoSQL
1. SQL is relational database	1. NoSQL is a non-relational database.
2. Fixed Schema Design & Structure	2. Dynamic Schema design & Structure
3. Handles Complex Queries	3. Handle large volumes of data
4. Vertically Scalable	4. Horizontally Scalable
5. Follows ACID property. A – Atomicity C – Consistency I – Isolation D - Durability	5. CAP property C – Consistency A – Availability P – Partial Tolerance

Advantages:

- High Performance and Scalability.
- Availability and Flexibility.
- Open Source and Schema less.

Disadvantages:

- Lack of Standardization
- Consistency Issues
- Limited Query Capabilities

MongoDB:

- It is an open source, cross-platform, document-oriented NoSQL database written in C++.
- Document, field & key-value based.
- A non-relational database
- No need of designing table, directly start coding.
- Supports SQL & JSON query language.
- Horizontally scalable
- Process the data very fast, as data is unstructured.

Scaling: 2 types

1. Vertical Scaling; Increasing the capacity of the System (16Gb to 32Gb).
2. Horizontal Scaling: Increasing the number of System (one server to two or many server/system).

SQL		MongoDB
Tables	Represented as	Collections
Rows	Represented as	Documents
Columns	Represented as	Fields

- Fields (Key & Value pairs) are stored in documents.
- Documents are stored in collection.
- Collections are stored in database.

MongoDB Setup:

1. Download MongoDB software from the official website of MongoDB and install it.
2. Set path for MongoDB Server in Environment Variable (up to /bin/ folder).
3. Create folder for MongoDB (c://data/db, mongo will search this location).
4. Download mongosh command line client tool (zip) from the official website of MongoDB and extract to folder where you want (path should set) or extract to mongo server bin folder (no need to set path for mongosh).
5. Set path for mongosh in Environment Variable (up to /bin/ folder).
6. Start MongoDB server and client,
   a. Open CMD,
   b. To start Mongo server type,
      - ➢ mongod (starts the mongo server, by default port 27017
   c. To start mongo client type,
      - ➢ mongosh
        (opens mongo shell, then you can play with db by performing operations)

Mongo Commands:

Note: Check MongoDB shell Command as per the Mongo Version you will use.

Command	Description
show dbs	Returns all available databases
use mydb	Use specified db for operation
db	Shows current db
show collections	Shows available collections of current db
db.collectionName.insertOne({key:value})	Inserts data to collection. Ex: db.employee.insertOne({“id”:101,”name”:”Honnur”})
db.dropDatabase()	Deletes the db
db.createCollection(name, options)	Creates a new Collection. Collections created dynamically. If we want we can create collection and then insert or else at the time of insertion the collection will be created name: String type, specifies name of the collection. options: document type, specifies the memory size & indexing of the collection (optional) Ex: db.createCollection(“employee”)
Show collections	Displays all
db.collectionName.find()	Prints all the documents present in collection
db.collectionName.find().pretty()	Format the documents and prints to the console
db.collectionName.drop()	Drop the collection
db.collectionName.deleteOne({"name":"Honnur"}))	Deletes the document based on specified type
var allEmployees = [{}, {}, {}] db.collectionName.insertMany(allEmployees)	Insert the multiple documents in collection.
Db.collectonName.dropIndex({field})	Drops the index based on specified field

Features:

- Open-source code
- Advanced Security
- Flexible Schema
- Easy to use
- Powerful query language
- High Performance
- High Flexible
- Reliable indexing
- On-demand scaling

Important Concepts:

- Aggregation
- Replication
- GridFS
- Indexing
- Sharding
- Ad hoc queries

Data Modeling:

- Data Modeling is the process of determining how data is stored and what connection exists between various entities in our data.
- Why We Use?
  Data models help to create a simplified and optimized logical database that eliminates redundancy, reduces storage requirements, and enables efficient retrieval.
  - High data quality
  - Understanding data flow and characteristics.
  - Development & maintenance
  - Performance
- Data models for database system can be conceptualized inti 3 categories based on the level of specificity or detail,
  1. Conceptual Data Model
  2. Logical Data Model
  3. Physical Data Model
- **Types of Data Models:**
  1. **Embedded Data Model:** Embedded Documents capture relationships between data by storing related data in a single document structure.
  2. **Reference Data Model:** Store the relationship between data by including links or references from one document to another.

Operators: 3 types of operators

1. **Query and Projection Operator:** Projection refers to picking only the data that is required rather than entire documents data.
2. **Update Operator:** Updates the value of the fields of documents matching the specified conditions.
3. **Aggregation Pipeline**: As a result of processing the data records and documents, aggregation operations provide computed results.

Query and Projection Operator:

a. Comparison Operator
b. Logical Operator
c. Array Operator
d. Element Operator
e. Bitwise Operator
f. Evaluation Operator
g. Geospatial Operator

- **Comparison Operator:**

Operator	Description
$eq	Matches value that are equal to the value specified in query
$ne	Matches values that are not equals to the value specified in query
$gt	Matches values that are greater than the value specified in query
$gte	Matches values that are greater than or equal to the value specified in query
$lt	Matches values that are less than the value specified in query
$lte	Matches values that are less than or equal to the value specified in query
$in / $nin	Matches any of the values that exist or non-exist in the given array

Ex: > db.employees.find({name:{$eq:"honnur"}})

- **Logical Operators:**

Operator	Description
$and	Returns all documents that match the given condition of both expression
$or	Returns all documents that match the given condition of either expression
$nor	Returns all documents that do not match the given condition of either expression
$not	Inverts the effect of a query expression and returns documents that do not match the query expression

Ex: > db.employees.find({$and:[{address.city:"Bangalore"}, {dept:"IT"}]})

- **Element Operator:**

Operator	Description
$exists	Returns documents that have a specified field
$type	Returns documents if field is of a specified type

Ex: > db.employees.find({"age":{$exists:true, $gte:25}})

- **Array Operator:**

Operator	Description
$all	Returns documents from a collection if it matches all values in the specified array
$size	Returns documents from the collection to match an array with the provided number of elements in it
$elemMatch	Returns those documents that match specified condition with in each array element

Ex: > db.employees.find({"age":{$all:[25, 35,40, 55]}})

- **Update Operator:**

Operator	Description
$currentDate	This operator is used to set the value of a specified field to current date, either as a Date or a Timestamp
$inc	Used to increment the value of the field by the specified amount
$min	Used only to update the field if the specified value is less than the existing field value
$max	Used only to update the field if the specified value is greater than the existing field value
$mul	This operator is used to multiply the value of the field by the specified amount
$rename	This operator is used to rename the field in document

Regular Expression:

- Used to match the patterns in a document.
- **$regex** provides regular expression capabilities for pattern matching strings in the queries.
  Syntax:
  ```
  db.collectionName.find({field:{$regex: /pattern/}})
  db.collectionName.find({field:{$regex:"^pattern$"}})
  db.collectionName.find({field:{$regex:/pattern/, $options: 's/x/i/m'}})
                    i – case insensitive
  ```

Projection:

- Is a unique feature that allows you to select only the necessary data rather than the entire set of data from the document.
- Why We Use?
  - Removing indexed query results without fetching full documents.
  - Removing not needed fields from query results.
  - Filter data without impacting the overall database performance.

  Ex: db.employees.find({}, {"name":1, "dept":1, "_id":0})
  1 – Inclusive in query results
  0 – Exclusive from the query results

sort(): Specifies the order in which a query returns the matching documents from a collection.
 1 – Ascending order
-1 – Descending order

Syntax:
```
db.collectionName.find().sort({field:order})
```

limit(): Used to limit the number of records. This method takes only one number type argument which is the number of documents to display.

Syntax:
```
db.collectionName.find().limit(value)
db.collectionName.find({}, {field:value}).limit(value)
```

Indexing:

- Indexes provide users with an efficient way of querying data.
- Avoids scanning the entire collection.
- Effective indexing strategy.
- Search efficiency.
- Creating index in mongodb is done by using the **createIndex()** method.
  Syntax:
  ```
  db.collectionName.createIndex({field:1 or -1})
             1 – Ascending
            -1 – Descending
  ```
- Single field index is used to create an index on the single field of a document.
  ```
  db.employees.createIndex({_id:-1})
  ```
- A compound index is an index that holds a reference to multiple fields of a collection.
  ```
  db.employees.createIndex({age:-1})
  ```
- Indexes in mongodb are essential in improving the performance of their data retrieval speed.

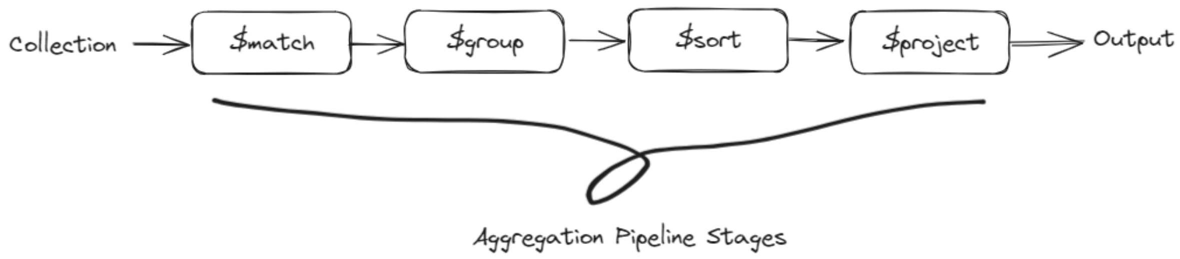
Advance Indexing Techniques:

- **Geospatial Index:** MongoDB uses 2D index to increase the efficiency of database queries when dealing with geospatial coordinate data.
- **Text Index:** MongoDB provides text indexes to support text search queries on string content. Text indexes can include any field whose value is a string or an array of string elements.
- **Hashed Index:** Hashed Index keep track of the entries with hashes of the values of the indexed field, which is "_id" field in all collections. This kind of index is primarily necessary when sharding data to distribute it evenly.

Aggregation:

- Is the process of selecting data from a collection to process multiple documents and returns computed results.
- Why Use?
  - Group values from multiple documents together.
  - Fetching a lot of nested data to perform complex operations.
  - Filter and sort documents and analyze data changes.

Aggregation Pipeline stages:



Operations:

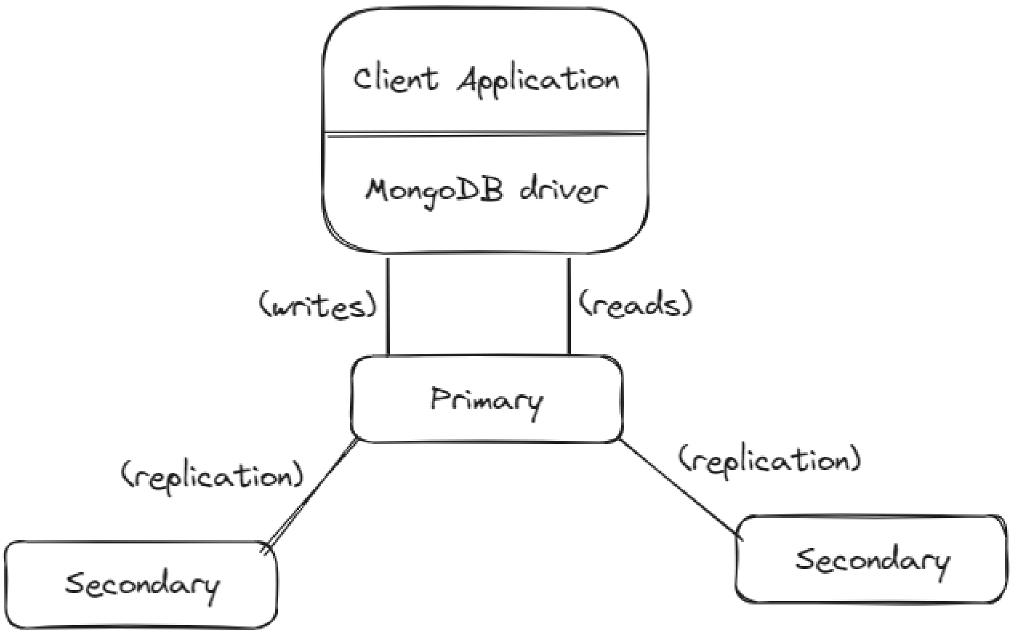
1. **$sum:** adds up the definite values.
2. **$avg:** Computes average value.
3. **$min:** returns minimum value.
4. **$max:** returns maximum value.
5. **$push:** adds the value to an array in the associated document.

Note:

- db.employees.find({name:"Honnur"}).explain(), explain() will return details of query execution.
- Hearbeat process is the method of finding out the current state of the mongodb nodes in a replica set.

Replication:

- Duplicating the same data across multiple mongodb servers.
  - Data redundancy and high availability.
  - Multiple copies across servers.
  - Data recovery and backup option
- Why we need?
  - Replication provides high (24 * 7) availability of data.
  - Protection from single server loss, hardware failure and service interruptions.
  - Ensure the data is always available to every client.



Replica set:

- MongoDB manages replication using replica sets, which are collections of related mongodb nodes.
- A replica set requires a minimum of three mongodb nodes.
- One of the nodes will be considered the primary node that receives all the write operations.
- The others are secondary nodes that will replicate the data from the primary node.
- If a failed node is recovered, works as a secondary node again.

Advantages:

- Replication helps in disaster recovery and backup of data.
- Improve application reliability.
- Replication minimizes downtime for maintenance.

- Load balancing is achieved.

Limitations:

- Higher costs and time constraints.
- Redundant data is stored, so more space and server processing required.

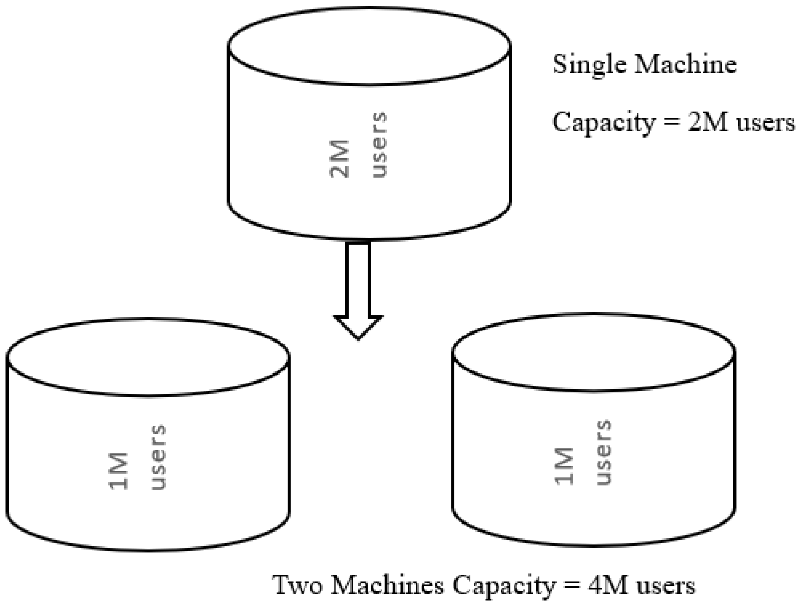
Sharding:

- Is a data distribution process and stores a single data set into multiple databases.
- Provides scalability to data.
- Why we need?
  - Database may struggle to handle more and more data and query traffic.
  - MongoDB instance is unable to manage write operations.
  - Memory cannot be outsized enough in case your dataset is large.
  - Vertical scaling is too costly.

Advantages:

- Increased storage capacity.
- Increased read/write throughput.
- High availability.
- Facilitates horizontal scaling.

Note: MongoDB handles horizontal scaling through Sharding.



Relationships:

- Representation of how the multiple documents are logically connected to each other in MongoDB.

Types:

1. **Embedded Method:** Capture relationships between data by storing related data in a single document structure.

One to One and One to Many relationships.

2. **Referenced Method:** Store the relationships between data by including links or references from one document to another.
   Many to many relationships.

GridFS:

- GridFS is a driver specification for uploading and retrieval of MongoDB files.
- Document size > 16MB such as Images, audio, video files are stored.
- Divides large files into equal chunks and stores them as a separate document.
- GridFS by default uses two collections fs.files and fs.chunks to store the files metadata and the chunks.

MapReduce:

- Is a data processing paradigm for condensing large volumes of data into useful aggregated results.
- Parameters:

  Map → Reduce → Query

  Syntax:

```
        db.employees.mapReduce(

                function(){emit(this.id, this.amount);},                → map

                function(key, values){ return Array.sum(values)},  → reduce

                {

                        query : {status : “A”},                         → query

                        out ; “emp_details”                             → out

                }

        );
```

  Ex:

```
        var map = function(){emit(this._id, this.salary)};

        var reduce = function(_id, salary){return Array.sum(salary)};

        db.employees.mapReduce(map, reduce, {out : “results”});

        db.results.find();
```

MongoDB Database References:

- MongoDB Dbref allows you to more easily reference documents stored in multiple collections or db’s.
- Parameters:
  - $ref: Field holds the name of the collection where the reference document resides.

- $id: Field contains the value of the id field in the referenced documents.
- $DB – Contains the name of the database where the referenced document resides.

Spring Boot with MongoDB:

1. Develop Spring Boot Application with below dependencies,

```
<dependencies>
        <dependency>
                <groupId>org.springframework.boot</groupId>
                <artifactId>spring-boot-starter-data-mongodb</artifactId>
        </dependency>
        <dependency>
                <groupId>org.springframework.boot</groupId>
                <artifactId>spring-boot-starter-web</artifactId>
        </dependency>
        <dependency>
                <groupId>org.projectlombok</groupId>
                <artifactId>lombok</artifactId>
                <optional>true</optional>
        </dependency>
        <dependency>
                <groupId>org.springframework.boot</groupId>
                <artifactId>spring-boot-devtools</artifactId>
                <scope>runtime</scope>
                <optional>true</optional>
        </dependency>
        <dependency>
                <groupId>org.springframework.boot</groupId>
                <artifactId>spring-boot-starter-test</artifactId>
                <scope>test</scope>
        </dependency>
</dependencies>
```

2. Configure database properties for MongoDB in application.properties or yml file,

```
spring:
  data:
    mongodb:
      host: localhost
      port: 27017
      database: SpringMongoDb
```

3. Create Entity class and annotated with @Document (Maps Entity class objects to JSON formatted documents)

```
package com.mongo.entity;

import java.io.Serializable;

import org.springframework.data.mongodb.core.mapping.Document;

import jakarta.persistence.Id;
import jakarta.persistence.Transient;
import lombok.AllArgsConstructor;
import lombok.NoArgsConstructor;
import lombok.ToString;
import lombok.Data;

@Document(collection = "Student")
@ToString
@AllArgsConstructor
@NoArgsConstructor
@Data
public class Student implements Serializable{
        @Transient
        private static final long serialVersionUID = 2864401892705212550L;

        @Id
        private String id;
        private String name;
        private String address;
        private String mobileNumber;
        private List<String> course;
}
```

4. Create a Repository interface and extend with MongoRepository,
   Redis as Message Broker
   Spring Batch
   AOP

```
package com.mongo.repository;

import org.slf4j.Logger;
import org.slf4j.LoggerFactory;
import org.springframework.cache.annotation.Cacheable;
import org.springframework.data.mongodb.repository.MongoRepository;

import com.mongo.entity.Student;

public interface StudentRepository extends MongoRepository<Student, Long> { }
```

5. Create a Service interface and its implementation class and perform CRUD operation. (Inject Repository),

```
package com.mongo.service;

import java.util.List;

import com.mongo.entity.Student;

public interface StudentService {

        public List<Student> getAllStudents();
```

```
		public Student saveStudent(Student student);

		public Student getByStudentId(String studentId);

		public Student updateStudent(Student student);

		public Boolean deleteStudentById(String studentId);
	}
```

6. Create Service Implementation class and implement Service interface,

```
	package com.mongo.service.impl;

	import java.util.List;
	import java.util.Optional;

	import org.springframework.beans.factory.annotation.Autowired;
	import org.springframework.stereotype.Service;

	import com.mongo.entity.Student;
	import com.mongo.repository.StudentRepository;
	import com.mongo.service.StudentService;

	@Service
	public class StudentServiceImpl implements StudentService {

		@Autowired
		private StudentRepository repository;

		@Override
		public List<Student> getAllStudents() {

			return repository.findAll();
		}

		@Override
		public Student saveStudent(Student student) {
			Optional<Student> findById = repository.findById(student.getId());
			return findById.isEmpty() ? repository.save(student) : null;
		}

		@Override
		public Student getByStudentId(String studentId) {

			Optional<Student> findById = repository.findById(studentId);
			return findById.isPresent() ? findById.get() : null;

		}

		@Override
		public Student updateStudent(Student student) {
			if (repository.existsById(student.getId())) {
				return repository.save(student);
			} else {
				return null;
			}
		}

		@Override
		public Boolean deleteStudentById(String studentId) {
			Optional<Student> findById = repository.findById(studentId);
			if (findById.isPresent()) {
				repository.deleteById(studentId);

				return true;
			} else {
				return false;
			}
		}
	}
```

7. Create a Controller class which performs the CRUD operations (Inject Service interface)

```
package com.mongo.restcontroller;

import java.util.Collection;
import java.util.List;

import org.slf4j.Logger;
import org.slf4j.LoggerFactory;
import org.springframework.beans.factory.annotation.Autowired;
import org.springframework.http.HttpStatus;
import org.springframework.http.ResponseEntity;
import org.springframework.web.bind.annotation.GetMapping;
import org.springframework.web.bind.annotation.PathVariable;
import org.springframework.web.bind.annotation.PostMapping;
import org.springframework.web.bind.annotation.PutMapping;
import org.springframework.web.bind.annotation.RequestBody;
import org.springframework.web.bind.annotation.RequestMapping;
import org.springframework.web.bind.annotation.RestController;

import com.mongo.entity.Student;
import com.mongo.service.StudentService;

@RestController
@RequestMapping("/sms/api/")
public class StudentRestController {

	private Logger logger = LoggerFactory.getLogger(StudentRestController.class);

	@Autowired
	private StudentService service;

	@GetMapping("/all")
```

```
    public ResponseEntity<Collection<Student>> findAllStudents() {
        List<Student> students = service.getAllStudents();
        logger.info(""+students);
        return ResponseEntity.ok(students);
    }

    @GetMapping("/student/{studentId}")
    public ResponseEntity<Student> findStudentById(@PathVariable String studentId) {
        Student student = service.getByStudentId(studentId);
        if (student != null) {
            return new ResponseEntity<Student>(student, HttpStatus.OK);
        } else {
            return new ResponseEntity<Student>(student,
HttpStatus.NOT_FOUND);
        }
    }

    @PostMapping
    public ResponseEntity<Student> saveStudent(@RequestBody Student student) {

        Student saveStudent = service.saveStudent(student);
        return ResponseEntity.ok(saveStudent);
    }

    @PutMapping
    public ResponseEntity<Student> updateStudent(@RequestBody Student student) {

        Student updateStudent = service.updateStudent(student);

        if (updateStudent != null) {
            return ResponseEntity.ok(updateStudent);
        } else {
            return new ResponseEntity<Student>(updateStudent,
HttpStatus.NOT_FOUND);
        }
    }

    @DeleteMapping("/student/{studentId}")
    public ResponseEntity<String> deleteStudent(@PathVariable String studentId) {
        boolean deleteStudentById = service.deleteStudentById(studentId);
        if (deleteStudentById) {
            return ResponseEntity.ok("Student with id " + studentId + " has been
deleted.");
        } else {
            return ResponseEntity.ok("Student with id " + studentId + " is not
found.");
        }
    }
}
```

8. Run the application and test the endpoints using Postman.