MongoDB is a **NoSQL database** that allows you to store and retrieve data in a flexible, JSON-like format called BSON (Binary JSON). It is particularly useful for applications that require handling unstructured or semi-structured data.

In this introduction, we’ll explain how to use MongoDB in Java, with a focus on the basics.

**What is MongoDB?**

* **Type**: NoSQL database (document-oriented).
* **Data Storage**: Data is stored in flexible, JSON-like documents with dynamic schemas.
* **Scalability**: Horizontally scalable using sharding.
* **Key Features**:
  + Schema-less: No predefined structure for data.
  + High performance for read/write operations.
  + Support for indexing, aggregation, and replication.

**How MongoDB Differs from Relational Databases**

| **Feature** | **MongoDB** | **Relational Database (RDBMS)** |
| --- | --- | --- |
| **Data Model** | Document-oriented | Table-oriented |
| **Schema** | Dynamic (flexible) | Predefined (rigid) |
| **Scalability** | Horizontal (sharding) | Vertical (scaling hardware) |
| **Query Language** | JSON-like query | SQL |

**Setting Up MongoDB in Java**

**Step 1: Install MongoDB**

1. Download MongoDB from the [official MongoDB website](https://www.mongodb.com/try/download/community).
2. Follow the installation instructions for your operating system.
3. Start the MongoDB server using:

bash

CopyEdit

mongod

**Step 2: Add MongoDB Java Driver to Your Project**

MongoDB provides a Java driver for connecting to and interacting with the database.

1. **Using Maven**: Add the following dependency to your pom.xml:

xml

CopyEdit

<dependency>

<groupId>org.mongodb</groupId>

<artifactId>mongodb-driver-sync</artifactId>

<version>4.8.1</version>

</dependency>

1. **Without Maven**: Download the JAR file from MongoDB Maven Repository and add it to your project.

**Step 3: Connect to MongoDB Using Java**

Create a simple Java program to establish a connection with MongoDB.

java

CopyEdit

import com.mongodb.client.MongoClient;

import com.mongodb.client.MongoClients;

import com.mongodb.client.MongoDatabase;

public class MongoDBConnection {

public static void main(String[] args) {

// Connection URI

String uri = "mongodb://localhost:27017";

// Connect to MongoDB

try (MongoClient mongoClient = MongoClients.create(uri)) {

// Access the database

MongoDatabase database = mongoClient.getDatabase("mydb");

System.out.println("Connected to database: " + database.getName());

} catch (Exception e) {

e.printStackTrace();

}

}

}

**Explanation**:

* **MongoClient**: Used to create a connection to the MongoDB server.
* **getDatabase()**: Accesses a specific database (mydb in this case).
* **try-with-resources**: Ensures the connection is closed after use.

**Basic CRUD Operations in MongoDB with Java**

**1. Create a Collection**

Collections in MongoDB are equivalent to tables in RDBMS.

java

CopyEdit

database.createCollection("students");

System.out.println("Collection 'students' created.");

**2. Insert Data into a Collection**

java

CopyEdit

import com.mongodb.client.MongoCollection;

import org.bson.Document;

// Access the 'students' collection

MongoCollection<Document> collection = database.getCollection("students");

// Create a document

Document student = new Document("name", "John")

.append("age", 22)

.append("course", "Computer Science");

// Insert the document into the collection

collection.insertOne(student);

System.out.println("Document inserted.");

**3. Read Data from a Collection**

java

CopyEdit

import com.mongodb.client.FindIterable;

// Retrieve documents from the 'students' collection

FindIterable<Document> documents = collection.find();

for (Document doc : documents) {

System.out.println(doc.toJson());

}

**4. Update Data in a Collection**

java

CopyEdit

import com.mongodb.client.model.Filters;

import com.mongodb.client.model.Updates;

// Update a document

collection.updateOne(Filters.eq("name", "John"), Updates.set("age", 23));

System.out.println("Document updated.");

**5. Delete Data from a Collection**

java

CopyEdit

// Delete a document

collection.deleteOne(Filters.eq("name", "John"));

System.out.println("Document deleted.");

**Advantages of Using MongoDB with Java**

1. **Flexibility**: Easy to handle changing data models.
2. **Rich Query Language**: Supports filtering, sorting, and aggregations.
3. **Seamless Integration**: The MongoDB Java Driver makes it easy to interact with MongoDB.

**When to Use MongoDB**

* Applications that handle large, unstructured datasets.
* Projects requiring frequent schema changes.
* Real-time analytics and content management systems.

**Key MongoDB Concepts**

* **Database**: A container for collections.
* **Collection**: A group of BSON documents.
* **Document**: JSON-like objects that represent data.
* **BSON**: Binary JSON used for efficient storage and querying.

This introduction provides a solid foundation for using MongoDB with Java. Let me know if

**1. MongoDB Connection Program**

import com.mongodb.client.MongoClient;

import com.mongodb.client.MongoClients;

import com.mongodb.client.MongoDatabase;

public class MongoDBConnection {

public static void main(String[] args) {

// Connection URI

String uri = "mongodb://localhost:27017";

// Connect to MongoDB

try (MongoClient mongoClient = MongoClients.create(uri)) {

// Access the database

MongoDatabase database = mongoClient.getDatabase("mydb");

System.out.println("Connected to database: " + database.getName());

} catch (Exception e) {

e.printStackTrace();

}

}

}

**Explanation**

1. **Connection URI**:
   * The uri string (mongodb://localhost:27017) specifies the address of the MongoDB server running locally on port 27017 (the default MongoDB port).
2. **MongoClient**:
   * The MongoClient object is created using the MongoClients.create(uri) method.
   * It is the main object for interacting with the MongoDB server.
3. **MongoDatabase**:
   * The getDatabase("mydb") method fetches a database named mydb. If it doesn’t exist, MongoDB will create it when you perform an operation on it.
4. **try-with-resources**:
   * Ensures that the MongoClient connection is closed automatically when the try block exits.
5. **Error Handling**:
   * The catch block ensures that if an error occurs (e.g., MongoDB is not running), it prints the stack trace.

**2. Creating a Collection**

database.createCollection("students");

System.out.println("Collection 'students' created.");

**Explanation**

1. **Method**:
   * The createCollection("students") method creates a new collection named students in the mydb database.
2. **Collections**:
   * Collections in MongoDB are equivalent to tables in relational databases.
   * Collections don’t require predefined schemas, allowing flexibility.
3. **Output**:
   * Prints a confirmation message that the collection has been created.

**3. Insert Data into a Collection**

MongoCollection<Document> collection = database.getCollection("students");

// Create a document

Document student = new Document("name", "John")

.append("age", 22)

.append("course", "Computer Science");

// Insert the document into the collection

collection.insertOne(student);

System.out.println("Document inserted.");

**Explanation**

1. **MongoCollection**:
   * The getCollection("students") method retrieves the students collection.
   * MongoCollection<Document> specifies that the collection contains BSON documents.
2. **Document**:
   * A Document represents a single record in the collection.
   * The append method is used to add key-value pairs to the document.
3. **Insert Operation**:
   * The insertOne(student) method inserts the document into the students collection.
4. **Output**:
   * Prints a confirmation message that the document has been inserted.

**4. Read Data from a Collection**

FindIterable<Document> documents = collection.find();

for (Document doc : documents) {

System.out.println(doc.toJson());

}

**Explanation**

1. **FindIterable**:
   * The find() method retrieves all documents from the collection.
   * Returns a FindIterable object, which is an iterable collection of documents.
2. **Loop**:
   * The for loop iterates over the documents and prints each one.
3. **toJson**:
   * Converts the BSON document into a JSON string for human-readable output.
4. **Output**:
   * Displays all documents in the collection.

**5. Update Data in a Collection**

collection.updateOne(Filters.eq("name", "John"), Updates.set("age", 23));

System.out.println("Document updated.");

**Explanation**

1. **Filters**:
   * The Filters.eq("name", "John") specifies the condition for selecting documents. Here, it finds a document where the name field is John.
2. **Updates**:
   * The Updates.set("age", 23) specifies the update operation. It sets the age field to 23.
3. **updateOne**:
   * Updates the first document that matches the filter condition.
4. **Output**:
   * Prints a confirmation message that the document has been updated.

**6. Delete Data from a Collection**

collection.deleteOne(Filters.eq("name", "John"));

System.out.println("Document deleted.");

**Explanation**

1. **Filters**:
   * The Filters.eq("name", "John") specifies the condition for selecting documents to delete. Here, it matches documents where the name field is John.
2. **deleteOne**:
   * Deletes the first document that matches the filter condition.
3. **Output**:
   * Prints a confirmation message that the document has been deleted.

**Summary**

These examples demonstrate the basic operations you can perform on a MongoDB database using Java:

1. **Connecting** to the database.
2. **Creating** collections.
3. **Inserting** data into collections.
4. **Reading** data from collections.
5. **Updating** existing data.
6. **Deleting** data