**Spring MongoDB Integration**

**Overview of MongDB**

MongoDB is an open-source document database and leading NoSQL database. MongoDB is written in C++. We can create and deploy a highly scalable and performance-oriented database.

**Installing MongoDB on Windows**

To install MongoDB on Windows, first download the latest release of MongoDB from

<https://www.mongodb.com/download-center>

After downloading this exe. Double click and install it on your machine. By default it will be installed at

C:\Program Files\MongoDB\

Before starting mongodb server, we need to configure the mongodb.cfg file for mongo data path etc as below.

dbpath = E:\MongoData

logpath = E:\MongoData\mongo.log

logappend = true

Before starting you need to add home directory path of mongodb in window environment PATH variable.

C:\Program Files\SSH Communications Security\SSH Secure Shell;C:\Program Files\Java\jdk1.8.0\_201\bin;C:\Users\Admin\AppData\Roaming\npm;C:\Program Files\MongoDB\Server\3.0\bin

After setting PATH in windows environment variable. Now open command prompt and type.

Mongo

Press enter.

It will show the following screen.

C:\Users\Admin>mongo

2020-03-31T12:20:42.279+0530 I CONTROL Hotfix KB2731284 or later update is not

installed, will zero-out data files

MongoDB shell version: 3.0.11

connecting to: test

Server has startup warnings:

2020-03-31T11:14:14.654+0530 I CONTROL [initandlisten]

2020-03-31T11:14:14.654+0530 I CONTROL [initandlisten] \*\* NOTE: This is a 32 bi

t MongoDB binary.

2020-03-31T11:14:14.654+0530 I CONTROL [initandlisten] \*\* 32 bit builds a

re limited to less than 2GB of data (or less with --journal).

2020-03-31T11:14:14.654+0530 I CONTROL [initandlisten] \*\* See http://doch

ub.mongodb.org/core/32bit

2020-03-31T11:14:14.654+0530 I CONTROL [initandlisten]

>

Use mydb;

Then execute mongo queries

**Integration of Spring and MongoDB**

Here we create a simple application that store and retrieve data from MongoDB, a document-based NO-SQL database.

**MongoDB**

is a document-oriented NoSQL database that stores JSON-like documents with dynamic schemas. It is commonly used for high volume data storage.

To Connect with MongoDB, There two approaches through we can connect to MongoDB database.

1. : MongoTemplate : The MongoTemplate follows the standard template pattern in spring

And provide ready to go basic API under persistence engine.

1. MongoRepository : The Repository follows the data centric approach and comes with more flexible and complex API operations based on well known access pattern in all spring data projects.
2. For both we need to defining dependency in pom.xml

<dependency>

<groupId>org.springframework.data</groupId>

<artifactId>spring-data-mongodb</artifactId>

<version>2.1.9.RELEASE</version>

</dependency>

And

<repositories>

<repository>

<id>spring-milestone</id>

<name>Spring Maven MILESTONE Repository</name>

<url>https://repo.spring.io/libs-milestone</url>

</repository>

</repositories>

**Step to create the simple project using maven.**

1. Create a simple maven project “mongodbspring” selecting maven-archetype-quickstart and create a package for our source files “com.mongodbspring” under src/main/java.
2. Now add the following dependencies into the pom.xml file

<project xmlns="http://maven.apache.org/POM/4.0.0" xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"

xsi:schemaLocation="http://maven.apache.org/POM/4.0.0 http://maven.apache.org/maven-v4\_0\_0.xsd">

<modelVersion>4.0.0</modelVersion>

<groupId>com.mongodbspring</groupId>

<artifactId>mongodbspring</artifactId>

<version>0.0.1-SNAPSHOT</version>

<name>mongodbspring Web App</name>

<url>http://maven.apache.org</url>

<properties>

<spring.version>5.1.8.RELEASE</spring.version>

</properties>

<dependencies>

<dependency>

<groupId>org.springframework</groupId>

<artifactId>spring-core</artifactId>

<version>5.1.8.RELEASE</version>

</dependency>

<dependency>

<groupId>org.springframework</groupId>

<artifactId>spring-context</artifactId>

<version>5.1.8.RELEASE</version>

</dependency>

<dependency>

<groupId>org.springframework</groupId>

<artifactId>spring-beans</artifactId>

<version>5.1.8.RELEASE</version>

</dependency>

<dependency>

<groupId>org.springframework.data</groupId>

<artifactId>spring-data-mongodb</artifactId>

<version>2.1.9.RELEASE</version>

</dependency>

<dependency>

<groupId>org.springframework</groupId>

<artifactId>spring-web</artifactId>

<version>${spring.version}</version>

</dependency>

<dependency>

<groupId>org.springframework</groupId>

<artifactId>spring-webmvc</artifactId>

<version>${spring.version}</version>

</dependency>

<dependency>

<groupId>javax.servlet</groupId>

<artifactId>javax.servlet-api</artifactId>

<version>3.1.0</version>

<scope>provided</scope>

</dependency>

<dependency>

<groupId>javax.servlet.jsp</groupId>

<artifactId>javax.servlet.jsp-api</artifactId>

<version>2.3.1</version>

<scope>provided</scope>

</dependency>

<dependency>

<groupId>javax.el</groupId>

<artifactId>javax.el-api</artifactId>

<version>3.0.0</version>

<scope>provided</scope>

</dependency>

<dependency>

<groupId>jstl</groupId>

<artifactId>jstl</artifactId>

<version>1.2</version>

</dependency>

<dependency>

<groupId>org.slf4j</groupId>

<artifactId>slf4j-api</artifactId>

<version>1.7.26</version>

</dependency>

<dependency>

<groupId>org.slf4j</groupId>

<artifactId>slf4j-jdk14</artifactId>

<version>1.7.26</version>

</dependency>

<dependency>

<groupId>log4j</groupId>

<artifactId>log4j</artifactId>

<version>1.2.17</version>

</dependency>

<dependency>

<groupId>com.fasterxml.jackson.core</groupId>

<artifactId>jackson-core</artifactId>

<version>2.9.4</version>

</dependency>

<dependency>

<groupId>com.fasterxml.jackson.core</groupId>

<artifactId>jackson-databind</artifactId>

<version>2.9.4</version>

</dependency>

</dependencies>

<repositories>

<repository>

<id>spring-milestone</id>

<name>Spring Maven MILESTONE Repository</name>

<url>https://repo.spring.io/libs-milestone</url>

</repository>

</repositories>

<build>

<finalName>mongodbspring</finalName>

</build>

<packaging>war</packaging>

</project>

1. Create the Java classes SpringConfig.java, Employee.java, EmployeeDAO.java, EmployeeDAOImpl.java, EmployeeRepository.java and Application.java under com.mongodbspring folder
2. Creating the model Employee.java in com.mongodbspring.model package.

package com.mongodbspring.model;

import org.springframework.data.annotation.Id;

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

@Document(value = "employee")

public class Employee {

@Id

private String \_id;

public String get\_id() {

return \_id;

}

public void set\_id(String \_id) {

this.\_id = \_id;

}

private String firstname;

private String lastname;

public Employee()

{

super();

}

public Employee(String firstname, String lastname)

{

super();

this.firstname = firstname;

this.lastname = lastname;

}

public String getFirstname()

{

return firstname;

}

public void setFirstname(String firstname)

{

this.firstname = firstname;

}

public String getLastname()

{

return lastname;

}

public void setLastname(String lastname)

{

this.lastname = lastname;

}

@Override

public String toString()

{

return "Employee [firstname=" + firstname + ", lastname=" + lastname + "]";

}

}

Similarly for Book model

package com.mongodbspring.model;

import java.io.Serializable;

import org.springframework.data.annotation.Id;

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

@Document(value="book")

public class Book implements Serializable{

/\*\*

\*

\*/

private static final long serialVersionUID = 1L;

public Book(Integer id, String title, Integer noOfPages, String writer, String category){

this.id = id;

this.title=title;

this.noOfPages=noOfPages;

this.writer = writer;

this.category = category;

}

@Id

private Integer id;

private String title;

private Integer noOfPages;

private String writer;

private String category;

public Integer getId() {

return id;

}

public void setId(Integer id) {

this.id = id;

}

public String getTitle() {

return title;

}

public void setTitle(String title) {

this.title = title;

}

public Integer getNoOfPages() {

return noOfPages;

}

public void setNoOfPages(Integer noOfPages) {

this.noOfPages = noOfPages;

}

public String getWriter() {

return writer;

}

public void setWriter(String writer) {

this.writer = writer;

}

public String getCategory() {

return category;

}

public void setCategory(String category) {

this.category = category;

}

@Override

public String toString()

{

return "Book [id=" + id + ", title=" + title + ", noOfPages"+noOfPages+", writer"+writer+", category "+category+"]";

}

}

1. Adding repository as EmployeeRepository.java in com.mongodbspring.repository package.

package com.mongodbspring.repository;

import org.springframework.data.mongodb.repository.MongoRepository;

import com.mongodbspring.model.Employee;

public interface EmployeeRepository extends MongoRepository<Employee, String>

{

public Employee findByFirstname(String firstname);

}

Similarly for BookRepository

package com.mongodbspring.repository;

import java.util.List;

import org.springframework.data.mongodb.repository.MongoRepository;

import org.springframework.data.mongodb.repository.Query;

import org.springframework.stereotype.Component;

import com.mongodbspring.model.Book;

@Component

public interface BookRepository extends MongoRepository<Book, Integer>{

@Query(value="{id:?0}")

Book findBookById(int id);

@Query("{writer : ?0, category : ?1}")

List<Book> findBooksByWriterAndCategory(String writer,String category);

@Query("{noOfPages: {$gt: ?0}}")

List<Book> findBooksGtThanNoOfPages(int noOfPages);

@Query("{writer: ?0, noOfPages: {$lt: ?1}}")

List<Book> findBooksByWriterAndLtThanNoOfPages(String writer, int noOfPages);

//db.book.find({$or:[{writer:"Mahesh"},{category:"Backend"}]});

@Query("{$or:[{writer:?0},{category:?1}]}")

List<Book> findBooksByWriterOrCategory(String writer, String category);

@Query(value="{writer:?0,category:?1}",fields="{'title':1,'writer':1,'noOfPages':1}")

List<Book> findBooksWithCertainFields(String writer, String category);

//db.book.find({category:"Backend"}).count();

@Query(value="{category:?0}",count=true)

Integer findBookCountByCategory(String category);

//db.book.find( { writer: { $exists: true } } )

@Query(value = "{writer : ?0}", exists = true)

Boolean isBooksAvailableByWriter(String writer);

//db.book.find({writer:"Mahesh"}).sort({title:1});

@Query(value = "{writer : ?0}", sort = "{title : 1}") //sorting order by title ascending

List<Book> findBooksByWriter(String writer);

//db.book.find({writer:"Mahesh"}).sort({title:-1});

@Query(value = "{category : ?0}", sort = "{title : -1}") //sorting order by title descending

List<Book> findBooksByCategory(String category);

@Query(value = "{category : ?0}", delete = true)

Long deleteBooksByCategory(String category);

}

1. Creating the Configuration file

package com.mongodbspring.test;

import org.springframework.context.annotation.Bean;

import org.springframework.context.annotation.ComponentScan;

import org.springframework.context.annotation.Configuration;

import org.springframework.data.mongodb.MongoDbFactory;

import org.springframework.data.mongodb.core.MongoTemplate;

import org.springframework.data.mongodb.core.SimpleMongoDbFactory;

import org.springframework.data.mongodb.repository.config.EnableMongoRepositories;

import com.mongodb.MongoClient;

@Configuration

@ComponentScan (basePackages = {"com.mongodbspring"})

@EnableMongoRepositories (basePackages = {"com.mongodbspring"})

public class SpringConfig {

@Bean

public MongoDbFactory mongoDbFactory()

{

MongoClient mongoClient = new MongoClient("localhost", 27017);

return new SimpleMongoDbFactory(mongoClient, "mydb");

}

@Bean

public MongoTemplate mongoTemplate()

{

MongoTemplate mongoTemplate = new MongoTemplate(mongoDbFactory());

return mongoTemplate;

}

}

@Configuration indicates that our SpringConfig class should be used by the Spring IoC container as a source of bean definitions.

@ComponentScan scans for the stereotype annotations specified in @Controller, @Service, etc.. annotated classes.

@EnableMongoRepositories scans the current package or packages mentioned in the basePackages attribute for any interface that extends Spring Data interface

1. The equivalent xml spring configurations are

<?xml version="1.0" encoding="UTF-8"?>

<beans xmlns="http://www.springframework.org/schema/beans"

xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"

xmlns:context="http://www.springframework.org/schema/context"

xmlns:mongo="http://www.springframework.org/schema/data/mongo"

xmlns:mvc="http://www.springframework.org/schema/mvc"

xsi:schemaLocation="http://www.springframework.org/schema/beans http://www.springframework.org/schema/beans/spring-beans.xsd

http://www.springframework.org/schema/mvc http://www.springframework.org/schema/mvc/spring-mvc.xsd

http://www.springframework.org/schema/context http://www.springframework.org/schema/context/spring-context.xsd

http://www.springframework.org/schema/data/mongo http://www.springframework.org/schema/data/mongo/spring-mongo.xsd">

<mongo:mongo-client id="mongoClient" host="localhost" port="27017" />

<mongo:db-factory id="mongoDbFactory" dbname="mydb" mongo-ref="mongoClient" />

<bean id="mongoTemplate" class="org.springframework.data.mongodb.core.MongoTemplate">

<constructor-arg ref="mongoDbFactory" />

</bean>

<bean class="org.springframework.dao.annotation.PersistenceExceptionTranslationPostProcessor"/>

<context:annotation-config />

<context:component-scan base-package="com.mongodbspring" />

<bean class="org.springframework.web.servlet.view.InternalResourceViewResolver">

<property name="prefix">

<value>/WEB-INF/jsp/</value>

</property>

<property name="suffix">

<value>.jsp</value>

</property>

</bean>

<mvc:resources mapping="/resources/\*\*" location="/resources/" />

<mvc:annotation-driven />

<mongo:repositories base-package="com.mongodbspring.repository" mongo-template-ref="mongoTemplate"/>

</beans>

1. Create Data Access Layer (EmployeeDao)

package com.mongodbspring.dao;

import java.util.List;

import com.mongodbspring.model.Employee;

public interface EmployeeDao {

public Employee createEmployee(Employee employee);

public Employee getEmployeeByFirstName(String firstname);

public List<Employee> getAllEmployees();

public void updateEmployee(Employee employee);

public void deleteEmployee(String id);

public Employee findAndUpdate(String name,String lastName);

}

1. DAO Implementation (EmployeeDaoImpl)

package com.mongodbspring.dao;

import java.util.List;

import java.util.Optional;

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

import org.springframework.data.mongodb.core.MongoTemplate;

import org.springframework.data.mongodb.core.query.Criteria;

import org.springframework.data.mongodb.core.query.Query;

import org.springframework.stereotype.Repository;

import com.mongodbspring.model.Employee;

import com.mongodbspring.repository.EmployeeRepository;

@Repository

public class EmployeeDaoImpl implements EmployeeDao{

@Autowired

private MongoTemplate mongoTemplate;

@Autowired

public EmployeeRepository employeeRepository;

public Employee createEmployee(Employee employee)

{

return employeeRepository.insert(employee);

}

public Employee getEmployeeByFirstName(String firstname)

{

return employeeRepository.findByFirstname(firstname);

}

public Employee getEmployeeById(String id)

{

Optional<Employee> e = employeeRepository.findById(id);

return e.get();

}

public List<Employee> getAllEmployees()

{

return employeeRepository.findAll();

}

public void updateEmployee(Employee employee)

{

employeeRepository.save(employee);

}

public void deleteEmployee(String id)

{

employeeRepository.deleteById(id);

}

public Employee findAndUpdate(String name, String lastName) {

Employee emp=mongoTemplate.findOne(Query.query(Criteria.where("firstname").is(name)), Employee.class);

emp.setLastname(lastName);

return employeeRepository.save(emp);

}

}

1. Standalone Application

package com.mongodbspring.test;

import java.util.ArrayList;

import java.util.List;

import java.util.stream.Stream;

import org.springframework.context.annotation.AnnotationConfigApplicationContext;

import org.springframework.context.support.AbstractApplicationContext;

import com.mongodbspring.dao.EmployeeDaoImpl;

import com.mongodbspring.model.Book;

import com.mongodbspring.model.Employee;

import com.mongodbspring.repository.BookRepository;

public class Application {

public static void main( String[] args )

{

AbstractApplicationContext context = new AnnotationConfigApplicationContext(SpringConfig.class);

//ClassPathXmlApplicationContext context = new ClassPathXmlApplicationContext("beans.xml");

//EmployeeDaoImpl employeeDaoImpl = (EmployeeDaoImpl) context.getBean("employeeDaoImpl");

// Create Employee

/\* Employee employee1 = new Employee("Tomwwws","Jerrywwws");

employeeDaoImpl.createEmployee(employee1);

// Get Employee by FirstName

Employee employee2 = employeeDaoImpl.getEmployeeByFirstName("Tomwwws");

System.out.println("\*\*\* Get Employee By FirstName \*\*\*");

System.out.println("First Name : "+employee2.getFirstname());

System.out.println("Last Name : "+employee2.getLastname());\*/

// Get all Employees

/\* List<Employee> employeeList = employeeDaoImpl.getAllEmployees();

System.out.println("\*\*\* Get All Employees \*\*\*");

for(Employee emp : employeeList)

{

System.out.println("First Name : "+emp.getFirstname());

System.out.println("Last Name : "+emp.getLastname());

System.out.println("#######################################");

}\*/

// Update Employee - Read from DB and Update the Employee

/\*Employee employee3 = employeeDaoImpl.getEmployeeById("5e7c77a96f1b124e469e56f6");

employee3.setLastname("Jonny");

employeeDaoImpl.updateEmployee(employee3);\*/

// Delete Employee

// employeeDaoImpl.deleteEmployee("5e7c77a96f1b124e469e56f6");

BookRepository repository = context.getBean(BookRepository.class);

/\*Book b1 = new Book(101, "Angular Tutorials", 200, "Krishna", "Frontend");

Book b2 = new Book(102, "JavaScript Tutorials", 200, "Krishna", "Frontend");

Book b3 = new Book(103, "Spring Tutorials", 300, "Mahesh", "Backend");

Book b4 = new Book(104, "Java Tutorials", 250, "Krishna", "Backend");

Book b5 = new Book(105, "Hibernate Tutorials", 150, "Mahesh", "Backend");

List<Book> list = new ArrayList<Book>();

list.add(b1);

list.add(b2);

list.add(b3);

list.add(b4);

list.add(b5);

List<Book> obj = repository.saveAll(list);\*/

//https://www.concretepage.com/spring-5/spring-data-mongodb-query

Book b=repository.findBookById(103);

System.out.println(b);

System.out.println("\n");

List<Book> books=repository.findBooksByWriterAndCategory("Krishna", "Frontend");

for(Book book:books){

System.out.println(book);

}

System.out.println("--- findBooksGtThanNoOfPages() ---");

List<Book> bookgt=repository.findBooksGtThanNoOfPages(200);

for(Book bgt:bookgt){

System.out.println(bgt);

}

System.out.println("--- findBooksByWriterAndLtThanNoOfPages() ---");

List<Book> booklt= repository.findBooksByWriterAndLtThanNoOfPages("Mahesh", 250);

for(Book blt:booklt){

System.out.println(blt);

}

System.out.println("--- findBooksByWriterOrCategory() :---");

List<Book> b1List=repository.findBooksByWriterOrCategory("Mahesh","Backend");

for(Book b1:b1List){

System.out.println(b1);

}

System.out.println("--- findBooksWithCertainFields() ---");

List<Book> b2List=repository.findBooksWithCertainFields("Mahesh", "Backend");

for(Book b2:b2List){

System.out.println(b2);

}

System.out.println("--- findBookCountByCategory() ---");

Integer count=repository.findBookCountByCategory("Backend");

System.out.println(count);

context.close();

}

}

To access by web we can configure in web.xml

<?xml version="1.0" encoding="UTF-8"?>

<web-app xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance" xmlns="http://java.sun.com/xml/ns/javaee" xsi:schemaLocation="http://java.sun.com/xml/ns/javaee http://java.sun.com/xml/ns/javaee/web-app\_3\_0.xsd" id="WebApp\_ID" version="3.0">

<display-name>MongodbSpring Application</display-name>

<servlet>

<servlet-name>dispatcher</servlet-name>

<servlet-class>

org.springframework.web.servlet.DispatcherServlet

</servlet-class>

<load-on-startup>1</load-on-startup>

</servlet>

<servlet-mapping>

<servlet-name>dispatcher</servlet-name>

<url-pattern>/</url-pattern>

</servlet-mapping>

<context-param>

<param-name>contextConfigLocation</param-name>

<param-value>/WEB-INF/dispatcher-servlet.xml</param-value>

</context-param>

<listener>

<listener-class>

org.springframework.web.context.ContextLoaderListener

</listener-class>

</listener>

</web-app>

1. To access on web as controller we need to create controller

package com.mongodbspring.controller;

import java.util.List;

import java.util.Map;

import org.apache.log4j.Logger;

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

import org.springframework.web.bind.annotation.RequestBody;

import org.springframework.web.bind.annotation.RequestMapping;

import org.springframework.web.bind.annotation.RequestMethod;

import org.springframework.web.bind.annotation.RestController;

import com.mongodbspring.model.Employee;

import com.mongodbspring.service.EmployeeService;

@RestController

public class MongodbController {

Logger logger=Logger.getLogger(MongodbController.class);

@Autowired

EmployeeService employeeService;

@RequestMapping(value="/getAllEmp",method=RequestMethod.GET)

public List<Employee> getAllEmployees() {

List<Employee> employees=null;

employees=employeeService.getAllEmployees();

return employees;

}

@RequestMapping(value="/addEmployee",method=RequestMethod.POST)

public Employee addEmployee(@RequestBody Map<String,Object> reqestMap) {

Employee emp=null;

emp=employeeService.addEmployee(reqestMap);

return emp;

}

@RequestMapping(value="/updateLastName",method=RequestMethod.POST)

public Employee updateLastName(@RequestBody Map<String,Object> reqestMap) {

return employeeService.findAndUpdate((String)reqestMap.get("firstName"), (String)reqestMap.get("lastName"));

}

}

1. The service layer we need to create service and their implementations

package com.mongodbspring.service;

import java.util.List;

import java.util.Map;

import com.mongodbspring.model.Employee;

public interface EmployeeService {

List<Employee> getAllEmployees();

Employee addEmployee(Map<String,Object> requestMap);

public Employee findAndUpdate(String name,String lastName);

}

The EmployeeServiceImpl.java is below

package com.mongodbspring.service;

import java.util.List;

import java.util.Map;

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

import org.springframework.stereotype.Service;

import com.mongodbspring.dao.EmployeeDao;

import com.mongodbspring.model.Employee;

@Service

public class EmployeeServiceImpl implements EmployeeService{

@Autowired

EmployeeDao employeeDao;

public List<Employee> getAllEmployees() {

// TODO Auto-generated method stub

return employeeDao.getAllEmployees();

}

public Employee addEmployee(Map<String,Object> requestMap) {

Employee emp=new Employee();

String firstName=(String)requestMap.get("firstName");

emp.setFirstname(firstName);

String lastName=(String)requestMap.get("lastName");

emp.setLastname(lastName);

// TODO Auto-generated method stub

return employeeDao.createEmployee(emp);

}

public Employee findAndUpdate(String name, String lastName) {

// TODO Auto-generated method stub

return employeeDao.findAndUpdate(name, lastName);

}

}

--------------------------------------------------------------------------------------------------------------------

1. **MongoDB Index**

An index in MongoDB is a special data structure that holds the data of few fields of documents on which the index is created. Indexes improve the speed of search operations in database because instead of searching the whole document, the search is performed on the indexes that holds only few fields. On the other hand, having too many indexes can hamper the performance of insert, update and delete operations because of the additional write and additional data space used by indexes.

**Default \_id index :**

The mongoDB creates a unique index on the field \_id during creation of collection.

The \_id index prevents clients from inserting two documents with the same value for the \_id field. You cannot drop this index on the \_id field.

**How to create Index**

Index type **Single field**

In addition to the MongoDB-defined \_id index, MongoDB supports the creation of user-defined ascending/descending indexes on a single field of a document

**Syntax of Index**

db.collection\_name({field\_name:1 or -1})

The value 1 is for ascending order, and the value -1 is for descending order

Example of index :

db.book.createIndex({writer:1});

After executing the above command, it shows like

db.book.createIndex({writer:1});

{

"createdCollectionAutomatically" : false,

"numIndexesBefore" : 1,

"numIndexesAfter" : 2,

"ok" : 1

}

Where

numIndexesBefore – The number of indexes before command is executed {-id}

numIndexesAfter – The number of indexes after the command is executed {-id and the one we have created}

ok – Command is successful.

**Finding the indexes on collection**

Syntax of finding index on a collection :

db.collection\_name.getIndexes();

for example :

db.book.getIndexes();

**Droping the Indexes :**

db.book.dropIndex({writer:1})

After executing it shows as :

{"nIndexesWas" : 2, "ok" : 1 }

Where

nIndexesWas – It shows how many indexes were there before this command is executed.

ok – This means the command is executed successfully.

**Dropping all indexes on a collection**

db.collection\_name.dropIndexes()

db.book.dropIndexes();

Index type **Compound field**

MongoDB also supports user-defined indexes on multiple fields

Syntax of creating compound index

db.collection.createIndex( { <field1>: <type>, <field2>: <type2>, ... } )

for example :

db.book.createIndex({writer:1,category:1});

**The text indexes**

db.book.createIndex( { title: "text" } )

db.book.find({$text:{$search:"Spring"}});

1. **MongoDB aggregation**

Aggregations operations process data records and return computed results. Aggregation operations group values from multiple documents together, and can perform a variety of operations on the grouped data to return a single result.

In MongoDB, there are three ways to perform the aggregation.

1. Aggregation pipeline
2. Map reduce Function
3. Single purpose aggregation methods

**Aggregation Pipeline**

In Aggregation Pipeline, documents enter in multi stage pipeline that

Transforms the documents into a aggregated results.

The most basic pipeline provides filters that operate like queries and document transformations that modify the form of output document. Other pipeline operations provide tools for grouping and sorting documents by specific field or fields as well as tools for aggregating the contents of arrays, including arrays of documents.

There is a set of possible stages and each of those is taken as a set of documents as an input and produces a resulting set of documents.

**$project** – used to select some specific fields from a collection.

**$match** – It is a filtering operation, and this can reduce the amount of documents that are given as input to the next stage.

**$group** – This does actual aggregation.

**$sort** – This does sorting the documents

**$skip** - With this, it is possible to skip forward in the list of documents for a given amount of documents.

**$limit** - This limits the amount of documents to look at, by the given number starting from the current positions.

**$unwind** -This is used to unwind document that are using arrays.

There are some examples given for aggregation.

The customers collections contains some records having fields product, total and customer.

1. find out how many toothbrushes were sold

db.customers.count({product:"toothbrush"});

Output :

3

1. Find the list of all products sold

db.customers.distinct("product");

Output:

["toothbrush", "guitar", "milk", "pizza" ]

1. Find the total amount of money spent by each customer.

db.customers.aggregate(

[

{$match:{}},

{$group:{\_id:"$customer",total:{$sum:"$total"}}}

]

);

Output :

{ "\_id" : "Dave", "total" : 4.75 }

{ "\_id" : "Karen", "total" : 13.25 }

{ "\_id" : "Tom", "total" : 199.99 }

{ "\_id" : "Mike", "total" : 20.78 }

4- Find the total amount of money spent by each customer

db.customers.aggregate(

[

{$match:{}},

{$group:{\_id:"$product",total:{$sum:"$total"}}}

]

);

Output :

{ "\_id" : "pizza", "total" : 13.25 }

{ "\_id" : "milk", "total" : 11.33 }

{ "\_id" : "guitar", "total" : 199.99 }

{ "\_id" : "toothbrush", "total" : 14.2 }

5- Find the total amount of money spent by each product in desending order

db.customers.aggregate(

[

{$match:{}},

{$group:{\_id:"$product",total:{$sum:"$total"}}},

{$sort:{total:-1}}

]

);

output :

{ "\_id" : "guitar", "total" : 199.99 }

{ "\_id" : "toothbrush", "total" : 14.2 }

{ "\_id" : "pizza", "total" : 13.25 }

{ "\_id" : "milk", "total" : 11.33 }

6 :Find the total amount of money spent by each in given customers

db.customers.aggregate(

[

{$match:{customer:{$in:["Mike","Karen"]}}},

{$group:{\_id:"$customer",total:{$sum:"$total"}}},

{$sort:{total:-1}}

]

);

Output :

{ "\_id" : "Mike", "total" : 20.78 }

{ "\_id" : "Karen", "total" : 13.25 }

we have grouped documents by field customer and on each occurrence of by customer previous value of sum is incremented.

**$sum** – sums the defined values from all documents in the collection.

Query :

db.customers.aggregate([{$group : {\_id : "$customer", total : {$sum : "$total"}}}]);

Output :

{ "\_id" : "Dave", "total" : 4.75 }

{ "\_id" : "Karen", "total" : 13.25 }

{ "\_id" : "Tom", "total" : 199.99 }

{ "\_id" : "Mike", "total" : 20.78 }

**$avg** - Calculates the average of all given values from all documents in the collection.

Query : db.customers.aggregate([{$group : {\_id : "$customer", total : {$avg : "$total"}}}]);

Output :

{ "\_id" : "Dave", "total" : 4.75 }

{ "\_id" : "Karen", "total" : 6.625 }

{ "\_id" : "Tom", "total" : 199.99 }

{ "\_id" : "Mike", "total" : 6.926666666666667 }

**$min** - Gets the minimum of the corresponding values from all documents in the collection.

Query :

db.customers.aggregate([{$group : {\_id : "$customer", total : {$min : "$total"}}}]);

Output :

{ "\_id" : "Dave", "total" : 4.75 }

{ "\_id" : "Karen", "total" : 4.75 }

{ "\_id" : "Tom", "total" : 199.99 }

{ "\_id" : "Mike", "total" : 4.7 }

**$max** - Gets the maximum of the corresponding values from all documents in the collection.

Query :

db.customers.aggregate([{$group : {\_id : "$customer", total : {$max : "$total"}}}]);

**$push** - Inserts the value to an array in the resulting document.

Query :

db.customers.aggregate([{$group : {\_id : "$customer", product : {$push: "$product"}}}]);

Output :

{ "\_id" : "Dave", "product" : [ "pizza" ] }

{ "\_id" : "Karen", "product" : [ "pizza", "toothbrush" ] }

{ "\_id" : "Tom", "product" : [ "guitar" ] }

{ "\_id" : "Mike", "product" : [ "toothbrush", "milk", "toothbrush" ] }

**$addToSet** - Inserts the value to an array in the resulting document but does not create duplicates.

Query :

db.customers.aggregate([{$group : {\_id : "$customer", product : {$addToSet : "$product"}}}]);

Output :

{ "\_id" : "Dave", "product" : [ "pizza" ] }

{ "\_id" : "Karen", "product" : [ "toothbrush", "pizza" ] }

{ "\_id" : "Tom", "product" : [ "guitar" ] }

{ "\_id" : "Mike", "product" : [ "milk", "toothbrush" ] }

**$first** - Gets the first document from the source documents according to the grouping.

Query :

db.customers.aggregate([{$group : {\_id : "$customer", first\_product : {$first : "$product"}}}]);

Output :

{ "\_id" : "Dave", "first\_product" : "pizza" }

{ "\_id" : "Karen", "first\_ product " : "pizza" }

{ "\_id" : "Tom", "first\_ product " : "guitar" }

{ "\_id" : "Mike", "first\_ product " : "toothbrush" }

$last - Gets the last document from the source documents according to the grouping.

Query :

db.customers.aggregate([{$group : {\_id : "$customer", last\_product : {$first : "$product"}}}]);

Output :

{ "\_id" : "Dave", "last\_product" : "pizza" }

{ "\_id" : "Karen", "last\_product" : "toothbrush" }

{ "\_id" : "Tom", "last\_product" : "guitar" }

{ "\_id" : "Mike", "last\_product" : "toothbrush" }