**Week 9**

**Transaction management and compliance to ACID principles**

* A [**transaction**](https://www.geeksforgeeks.org/sql-transactions/) is a single logical unit of work that accesses and possibly modifies the contents of a database. Transactions access data using read and write operations.
* In order to maintain consistency in a database, before and after the transaction, certain properties are followed. These are called **ACID** properties.

**A.C.I.D. properties: Atomicity, Consistency, Isolation, and Durability**

If a database operation has these ACID properties, it can be called an ACID transaction, and data storage systems that apply these operations are called transactional systems.

ACID transactions guarantee that each read, write, or modification of a table has the following properties:

* **Atomicity** - It means if any operation is performed on the data, either it should be performed or executed completely or should not be executed at all.
* **Consistency** – Consistency guarantees that changes made within a transaction are consistent with database constraints.

This includes all rules, constraints, and triggers. If the data gets into an illegal state, the whole transaction fails.

* **Isolation** – Isolation is the property of a database where no data should affect the other one and may occur concurrently.

It means if two operations are being performed on two different databases, they may not affect the value of one another.

* **Durability** - ensures that the data after the successful execution of the operation becomes permanent in the database..

**Securing REST APIs with Spring Security**

In order to add security to our Spring Boot application, we need to add the security starter dependency

<dependency>

<groupId>org.springframework.boot</groupId>

<artifactId>spring-boot-starter-security</artifactId>

</dependency>

This will also include the SecurityAutoConfiguration class containing the initial/default security configuration.

**By default, the Authentication gets enabled for the Application. Also, content negotiation is used to determine if basic or formLogin should be used.**

There are some predefined properties:

spring.security.user.name=root

spring.security.user.password=root

If we don't configure the password using the predefined property *spring.security.user.password* and start the application, a default password is randomly generated and printed in the console log:

Using default security password: c8be15de-4488-4490-9dc6-fab3f91435c6

File - new – Project - spring starter project

Name: spring-basic-security

Package: com.example.security

Click Next - Add Dependencies: Spring Web, Spring Security, Spring Boot Dev Tools….

Finish

**Name:** SpringBasicSecurityApplication

**package** com.example.security;

**SpringBasicSecurityApplication.java**

**import** org.springframework.boot.SpringApplication;

**import** org.springframework.boot.autoconfigure.SpringBootApplication;

**import** org.springframework.security.config.annotation.web.configuration.EnableWebSecurity;

@SpringBootApplication

@EnableWebSecurity

**public** **class** SpringBasicSecurityApplication {

**public** **static** **void** main(String[] args) {

SpringApplication.*run*(SpringBasicSecurityApplication.**class**, args);

}

}

**Create Package**

**Name: com.example.security.controller**

**Create Class**

**Name: ApplicationController**

package com.example.security.controller;

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

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

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

@RestController

@RequestMapping("/auth")

public class ApplicationController {

@GetMapping("/getmsg")

public String greeting()

{

return "Spring Security Example";

}

}

**application.properties File**

spring.security.user.name=niranjan

spring.security.user.password=murthy

server.port=8090

**JUnit Tutorials**

**Testing**: Testing is the process of checking an application that it is working as expected. In other words testing is a process of verification and validation.

**Unit testing**: Unit testing is the testing of an individual unit (class/method) or group of related units

**Types of unit testing**:

* **Manual testing:**Manual testing is the process of executing a test case without any tool support.
* **Automated testing:** Automated testing is the process of executing a test case with any tool support.

**JUnit:** JUnit is an open-source unit testing framework for java programmers. It is only used for unit testing. Integration testing is done by TestNG.

**Unit test case:**

Unit test case is part of code which executes to check that another part of the code works as expected.

**Annotations for Junit testing:**

**1. @Test:** It is used to specify the test method.

**2. @BeforeClass:** It is used to specify that method will be called only once, before starting all the test cases.

**3. @AfterClass:** It is used to specify that method will be called only once, after finishing all the test cases.

**4. @Before:** It is used to specify that method will be called before each test case.

**5. @After:** It is used to specify that method will be called after each test case.

**6. @Ignore:** It is used to ignore the test case.

**Assert class:**

JUnit provides the Assert class to check the certain conditions. Assert class methods compare the output value to the expected value.

Commonly used methods of Assert class:

1. **assertTrue(boolean condition):** It assert that the specified boolean condition is true.
2. **assertFalse(boolean condition):** It assert that the specified boolean condition is false
3. **assertNull(Object obj):** It assert that the specified object is null.
4. **assertNotNull(Object obj):** It assert that the specified object is not null.
5. **assertEquals(Object expected, Object actual):** It assert that two objects are equal.
6. **assertSame(Object expected, Object actual):** It assert that two objects refer to the same object.

**Writing Junit test cases for CRUD operations**

Download JUnit from https://junit.org/junit4/

Goto download & install

Find Plain-old Jar & Download the following

* [junit.jar](https://search.maven.org/search?q=g:junit%20AND%20a:junit)
* [hamcrest-core.jar](https://search.maven.org/artifact/org.hamcrest/hamcrest-core/1.3/jar)
* Create a folder in any drive by giving relevant name, copy and paste both jar files to the folder.
* Create a project in eclipse
* Right click on project select build path, click on configure build path
* Select java build path, Click on Libraries and click on class path in libraries, go to Add External JAR’s, select junit.jar and hamcrest-core.jar files, click on apply and then apply and close.
* Goto src/test/java folder find default package and Testclass
* Write the below code

**package** com.example.demo;

**import** **static** org.assertj.core.api.Assertions.*assertThat*;

**import** **static** org.junit.Assert.*assertNotEquals*;

**import** **static** org.junit.Assert.*assertNotNull*;

**import** java.util.List;

**import** org.junit.jupiter.api.Test;

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

**import** org.springframework.boot.test.context.SpringBootTest;

**import** com.example.demo.entity.User;

**import** com.example.demo.repository.UserRepository;

@SpringBootTest

**class** SpringbootFirstAppApplicationTests {

@Autowired

UserRepository userRepo;

@Test

**public** **void** testCreate()

{

User u=**new** User();

u.setId(3L);

u.setFirstname("Kavya");

u.setLasttname("shree");

userRepo.save(u);

*assertNotNull*(userRepo.findById(902L).get());

}

@Test

**public** **void** testReadAll()

{

List<User> list=userRepo.findAll();

*assertThat*(list).size().isGreaterThan(0);

}

@Test

**public** **void** testUpdate()

{

User u=userRepo.findById(902L).get();

u.setFirstname("Murthy");

userRepo.save(u);

*assertNotEquals*("Niranjan",userRepo.findById(902L).get().getFirstname());

}

@Test

**public** **void** testDelete()

{

userRepo.deleteById(852L);

*assertThat*(userRepo.existsById(852L)).isFalse();

}

}

**Introduction NoSQL**

A database Management System provides the mechanism to store and retrieve the data. There are different kinds of database Management Systems:

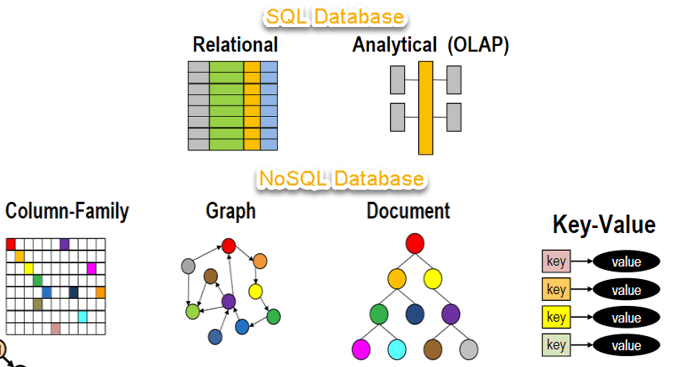
1. RDBMS (Relational Database Management Systems)

2. OLAP (Online Analytical Processing)

3. NoSQL (Not only SQL)

**NoSql**

* NoSQL database stands for “Not Only SQL” or “Not SQL.”
* NoSQL Database is a non-relational Data Management System that does not require a fixed schema.
* It avoids joins, and is easy to scale. The major purpose of using a NoSQL database is for distributed data stores with huge data storage needs.
* NoSQL is used for Big data and real-time web apps. For example, companies like Twitter, Face book and Google collect terabytes of user data every single day.
* NoSQL can store structured, semi-structured, unstructured and polymorphic data



**Brief History:**

* 1998- Carlo Strozzi use the term NoSQL for his lightweight, open-source relational database
* 2000- Graph database Neo4j is launched
* 2004- Google BigTable is launched
* 2005- CouchDB is launched
* 2007- The research paper on Amazon Dynamo is released
* 2008- Facebooks open sources the Cassandra project
* 2009- The term NoSQL was reintroduced

**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
* 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**

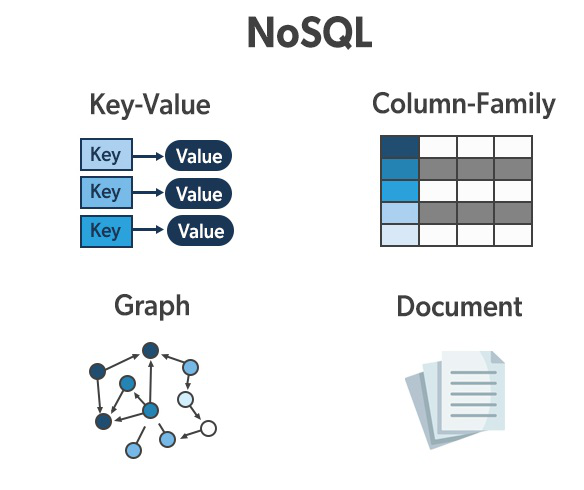
* Offers easy to use interfaces for storage and querying data provided
* APIs allow low-level data manipulation & selection methods
* Text-based protocols mostly used with HTTP REST with JSON
* Mostly used no standard based NoSQL query language
* Web-enabled databases running as internet-facing services

**Distributed**

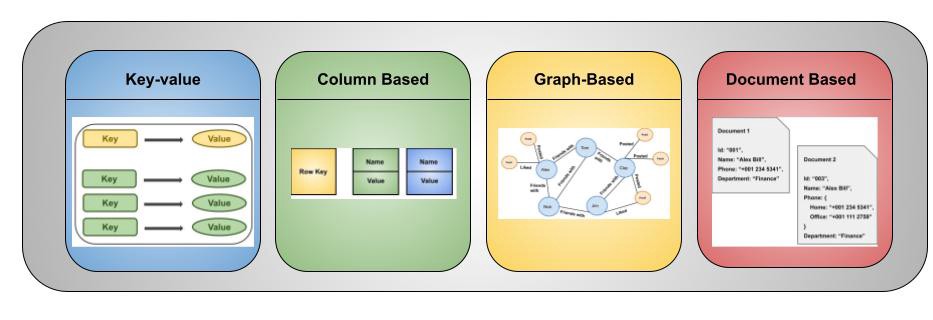
* Multiple NoSQL databases can be executed in a distributed fashion
* Offers auto-scaling and fail-over capabilities
* Often ACID concept can be sacrificed for scalability and throughput
* Only providing eventual consistency

**Types of NoSQL Databases**

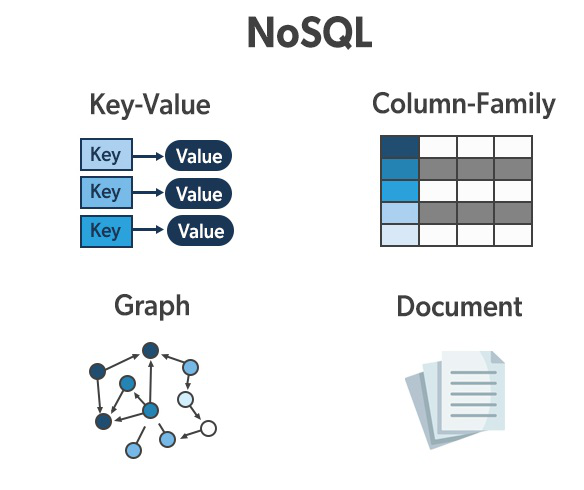
* 1. **Key-value Pair Based:**
* Data is stored in key/value pairs. It is designed in such a way to handle lots of data and heavy load.
* Key-value pair storage databases store data as a hash table where each key is unique, and the value can be a JSON, BLOB(Binary Large Objects), string, etc.
* For example, a key-value pair may contain a key like “Website” associated with a value like “Sanpoly”.



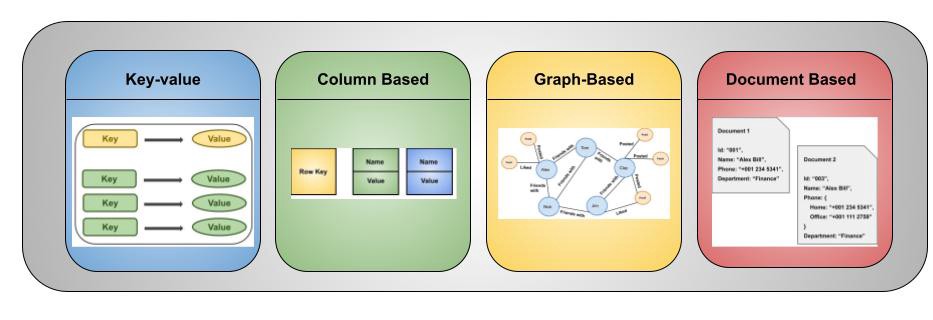
* 1. **Column-oriented Graph**
* They store data in tables, rows, and dynamic columns. Here each row doesn’t need to have the same columns.
* These were created to store and process very large amounts of data distributed over many machines.



* 1. **Graphs based**
* A graph type database stores entities as well the relations amongst those entities. The entity is stored as a node with the relationship as edges. An edge gives a relationship between nodes. Every node and edge has a unique identifier.



* 1. **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.



**Benefits**

There are many advantages of working with NoSQL databases such as MongoDB and Cassandra. The main advantages are high scalability and high availability.

1. **High scalability –** NoSQL databases use sharding(dividing a larger part into smaller parts) for horizontal scaling. Partitioning of data and placing it on multiple machines in such a way that the order of the data is preserved is sharding. Vertical scaling means adding more resources to the existing machine whereas horizontal scaling means adding more machines to handle the data. Vertical scaling is not that easy to implement but horizontal scaling is easy to implement. Examples of horizontal scaling databases are MongoDB, Cassandra, etc. NoSQL can handle a huge amount of data because of scalability, as the data grows NoSQL scale itself to handle that data in an efficient manner.
2. **High availability –** Auto replication feature in NoSQL databases makes it highly available because in case of any failure data replicates itself to the previous consistent state.

**Cap Theorm**

The CAP theorem is a belief from theoretical computer science about [distributed data stores](https://en.wikipedia.org/wiki/Distributed_data_store) that claims, in the event of a network failure on a distributed database, it is possible to provide either consistency or availability—but not both.

* **Consistency:**

The data should remain consistent even after the execution of an operation. This means once data is written, any future read request should contain that data. For example, after updating the order status, all the clients should be able to see the same data.

* **Availability:**

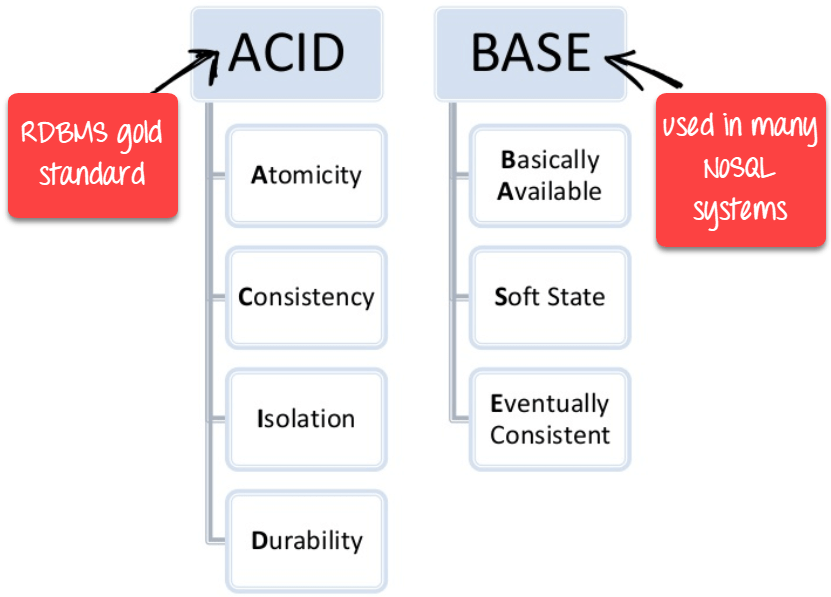
The database should always be available and responsive. It should not have any downtime.

* **Partition Tolerance:**

Partition Tolerance means that the system should continue to function even if the communication among the servers is not stable. For example, the servers can be partitioned into multiple groups which may not communicate with each other. Here, if part of the database is unavailable, other parts are always unaffected.

**BASE: Basically Available, Soft state, Eventual consistency**

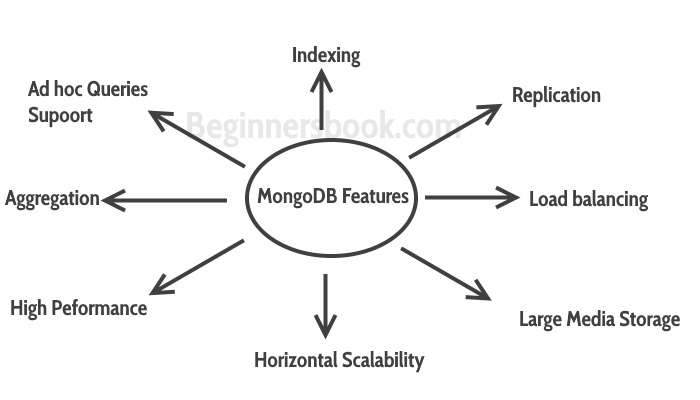
* Basically, available means DB is available all the time as per CAP theorem
* Soft state means even without an input; the system state may change
* Eventual consistency means that the system will become consistent over time.



**Mongo DB**

* MongoDB is a document-oriented NoSQL database used for high volume data storage.
* Instead of using tables and rows, MongoDB makes use of collections and documents.
* Documents consist of key-value pairs which are the basic unit of data in MongoDB.
* Collections contain sets of documents and function which is the equivalent of relational database tables.

**MongoDB Features**



* MongoDB provides **high performance**. Most of the operations in the MongoDB are faster compared to relational databases.
* MongoDB provides **auto replication** feature that allows you to quickly recover data in case of a failure.
* **Horizontal scaling** is possible in MongoDB because of sharing. Sharding is partitioning of data and placing it on multiple machines in such a way that the order of the data is preserved.
* **Vertical scaling** means adding more resources to the existing machine.
* **Load balancing:** horizontal scaling allows MongoDB to balance the load.
* **High Availability:** Auto Replication improves the availability of MongoDB database.
* **Indexing**: Index is a single field within the document. Indexes are used to quickly locate data without having to search every document.

**Key Components of MongoDB Architecture**

**1.\_id:**

* The \_id field represents a unique value in the MongoDB document.
* The \_id field is like the document’s primary key.
* If you create a new document without an \_id field, MongoDB will automatically create the field.

**2. Collection**: This is a grouping of MongoDB documents. A collection is the equivalent of a table.

**3. Cursor**: This is a pointer to the result set of a query. Clients can iterate through a cursor to retrieve results.

**4. Database**: This is a container for collections (tables).

**5. Document**: A record in a MongoDB collection is basically called a document. The document, in turn, will consist of field name and values.

**6. Field**: A name-value pair in a document. A document has zero or more fields. Fields are analogous to columns in relational databases.

**7. JSON**: This is known as JavaScript Object Notation. This is a human-readable, plain text format for expressing structured data.

**Data Modelling in MongoDB**

MongoDB provides two types of data models

* **Embedded data model:** In this model, you can have (embed) all the related data in a single document, it is also known as de-normalized data model

{

\_id: , Emp\_ID: "10025AE336"

Personal\_details:

{

First\_Name: "Radhika",

Date\_Of\_Birth: "1995-09-26"

},

Address:

{

city: "Mysore",

}

}

* **Normalized data model:** In this model, you can refer the sub documents in the original document, using references.

**Employee:**

{

\_id: <ObjectId101>,

Emp\_ID: "10025AE336"

}

**Personal\_details:**

{

\_id: <ObjectId102>,

empDocID: " ObjectId101",

First\_Name: "Radhika",

Date\_Of\_Birth: "1995-09-26"

}

**Address:**

{

\_id: <ObjectId103>,

empDocID: " ObjectId101",

city: "Mysore"

}

**Working with MongoDB**

**Mongo Compass GUI Setup**

**Download and Install MongoDB Community Server**

* Visit www.mongodb.com and click on MongoDB Community Edition, select a version, platform, and package to download.
* To install follow the instructions.

**MongoDB Shell – mongosh**

* MongoDB Shell is the quickest way to connect, configure, query, and work with your MongoDB database. It acts as a command-line client of the MongoDB server.

**Install mongosh**

* Visit www.mongodb.com and click on Product menu -> Tools -> Compass, then click on Download.
* This will take you to a page where you can select a version, platform, and package to download. Zip file will be downloaded.
* Extract the file.
* Open mongosh folder, goto bin, double click on mongosh, press enter
* Type show dbs

**Mongo Compass GUI Setup**

**Download and Install MongoDB Community Server**

* Visit www.mongodb.com and click on MongoDB Community Edition, select a version, platform, and package to download.
* To install follow the instructions.

**Create and Manage MongoDB**

test>show dbs 🡪 display list of existing database

test> use jsspn 🡪 switch to jsspn database

jsspn>

**Datatypes and Operators**:

MongoDB's data types allow us to store [data](https://www.simplilearn.com/what-is-data-article) more efficiently while simultaneously performing very efficient and robust queries.

[1. String](https://www.simplilearn.com/tutorials/mongodb-tutorial/mongodb-data-types#1_string): To represent text, the string type is utilized

[2. Integer](https://www.simplilearn.com/tutorials/mongodb-tutorial/mongodb-data-types#2_integer): Numeric values are stored using the integer data type. Depending on the server, it can store 32-bit or 64-bit numbers.

[3. Double](https://www.simplilearn.com/tutorials/mongodb-tutorial/mongodb-data-types#3_double): Numeric numbers containing 8 bytes floating-point are stored using the double data type.

[4. Boolean](https://www.simplilearn.com/tutorials/mongodb-tutorial/mongodb-data-types#4_boolean): Boolean (true or false) values are stored with the boolean data type

[5. Array](https://www.simplilearn.com/tutorials/mongodb-tutorial/mongodb-data-types#5_array): We can store several values in a single key of the document with an array data type.

6. Object : An embedded document is a key-value pair that is put inside another document. Embedded documents are stored using the object data type.

7. Date : The current date or time is stored in the ‘Date’ data type. The returning date can be done in a variety of ways; either a string or a date object. There are three strategies that can be used in this situation.

1. The Date() function returns a string.
2. Return a date object with New Date().
3. ISODate() returns a date object as well.

8. Timestamp : The term "Timestamp" refers to a set of characters used to describe the date and time of an occurrence. The timestamp data type is commonly used to track the creation, editing, and updating of documents. Such characters are stored in the timestamp data type. To construct a timestamp, use the new Timestamp().

[9. Null](https://www.simplilearn.com/tutorials/mongodb-tutorial/mongodb-data-types#9_null): The NULL data type is used to represent a value of zero or no value, as the name implies.

[10. Binary](https://www.simplilearn.com/tutorials/mongodb-tutorial/mongodb-data-types#10_binary): This data type is used in fields to hold binary data.

{

"\_id": "1",

"product\_code": "1234-ABCD",

"product\_name": "TV",

"product\_price": 49999.99,.

"product\_feature":  ["HD", "10wt"],

"product\_dimensions": {

"product \_height": 55,

"product\_width": 75,

},

"mfd\_date": ISODate("2022-04-10T12:45:42"),

"product\_created": Timestamp(1531456567, 1),

"product\_availability": true,

"product\_picture":BinData(1, "rk56tyvbu5677ghugf456.")

}

Operators are special symbols or keywords that inform a compiler or an interpreter to carry out mathematical or logical operations.

MongoDB offers the following query operator types:

* Comparison
* Logical
* Element
* Evaluation
* Geospatial
* Array
* Bitwise
* Comments

**Comparison Operators**

MongoDB comparison operators can be used to compare values in a document. The following table contains the common comparison operators.

|  |  |
| --- | --- |
| **Operator** | **Description** |
| $eq | Matches values that are equal to the given value. |
| $gt | Matches if values are greater than the given value. |
| $lt | Matches if values are less than the given value. |
| $gte | Matches if values are greater or equal to the given value. |
| $lte | Matches if values are less or equal to the given value. |
| $in | Matches any of the values in an array. |
| $ne | Matches values that are not equal to the given value. |
| $nin | Matches none of the values specified in an array. |

**Logical Operators**

MongoDB logical operators can be used to filter data based on given conditions. These operators provide a way to combine multiple conditions. Each operator equates the given condition to a true or false value.

|  |  |
| --- | --- |
| **Operator** | **Description** |
| $and | Joins two or more queries with a logical AND and returns the documents that match all the conditions. |
| $or | Join two or more queries with a logical OR and return the documents that match either query. |
| $nor | The opposite of the OR operator. The logical NOR operator will join two or more queries and return documents that do not match the given query conditions. |
| $not | Returns the documents that do not match the given query expression. |

Array Operators

MongoDB array operators are designed to query documents with arrays. Here are the array operators provided by MongoDB.

|  |  |
| --- | --- |
| **Operator** | **Description** |
| $all | Matches arrays that contain all the specified values in the query condition. |
| $size | Matches the documents if the array size is equal to the specified size in a query. |
| $elemMatch | Matches documents that match specified $elemMatch conditions within each array element. |