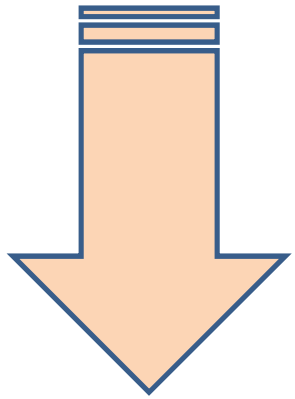
MongoDB



MongoBD

MongoDB:

- Understanding NoSQL and MongoDB,
- Using MongoBD Compass, MongoDB Shell
- Accessing MongoDB from Node.js.

NoSQL

No SQL - Characteristics

- The motivation behind NoSQL is mainly simplified designs**, horizontal scaling (increasing the count of the processing/storage units)**, and finer control of the availability of data.
- There are several different NoSQL technologies, such as

  HBase's column structure,

  Redis's key/value structure, and

  Neo4j's graph structure.
- MongoDB and the document model were chosen because of **great flexibility and scalability** when it comes to implementing backend storage for web applications and services.
- MongoDB is one of the most popular and well supported NoSQL databases currently available.
- MongoDB is a NoSQL database based on a document model where data objects are stored as separate documents inside a collection. The motivation of the MongoDB language is to implement a data store that provides
  - **high performance**
  - **high availability**
  - **and automatic scaling**

No SQL - Characteristics

- **Multi-Model:** This feature of NoSQL databases makes them extremely flexible when it comes to handling data.
- **Easily Scalable:** This feature of NoSQL databases easy scales to adapt to huge volumes and complexity of cloud applications. This scalability also improves performance, allowing for continuous availability and very high read/write speeds.
- **Flexible:** This feature of NoSQL databases allows you to process all varieties of data. **It can process structured, semi-structured and unstructured data**. It works on many processors—NoSQL systems allow you to store your database on multiple processors and maintain high-speed performance.
- **Less Downtime:** The elastic nature of NoSQL allows for the workload to automatically be spread across any number of servers.
-

SQL dbs

- SQL databases use the ACID database properties to ensure that the database transactions are reliable.
- ACID stands for
  - **Atomicity** (An “all or nothing” approach for the data that is committed to be saved),
  - **Consistency** (Interrupted changes are rolled back),
  - **Isolation** (Intermediate state of a transaction is not visible to other transactions), and
  - **Durability** (Completed transactions retain their state even in system failure).

SQL vs *No SQL*

You would choose an SQL database when:

- You need a database with **a predefined schema** so that applications adhere to that schema.
- You are designing an application that requires **multi-row transactions**.
- You require a database that has **no room for error and is very consistent**, for example in the case of data warehousing systems.

You would choose an NoSQL database when:

- You need a database that accounts for **exponential growth with no clear schema** definitions.
- You require a database which can accommodate **variable data structures** and plays well with big data platforms such as Hadoop.
- You need a **distributed database** system that scales easily and inexpensively.

No SQL

SQL	NoSQL
Relational Data Base Management System (RDBMS)	Non-relational or distributed database system. (Non – RDBMS)
These databases have fixed or static or predefined schema	They have dynamic schema
These databases are not suited for hierarchical data storage.	These databases are best suited for hierarchical data storage.
These databases are best suited for complex queries	These databases are not so good for complex queries
Vertically Scalable	Horizontally scalable
Follows **ACID** (Automocity, Consistency, Isolation and Durability) property	Follows **CAP**(consistency, availability, partition tolerance)

SQL vs No SQL

1. Type:

- SQL databases are primarily called as Relational Databases (RDBMS); whereas
- NoSQL database are primarily called as non-relational or distributed database.

2. Language difference:

- **SQL requires you to use predefined schemas** to determine the structure of your data before you work with it. Also all of your data must follow the same structure. This can require significant up-front preparation which means that a change in the structure would be both difficult and disruptive to your whole system.
- **A NoSQL database has dynamic schema for unstructured data**. Data is stored in many ways which means it can be **document-oriented, column-oriented, graph-based or organized as a KeyValue store.** This flexibility means that documents can be created without having defined structure first. Also **each document can have its own unique structure**. The syntax varies from database to database, and you can add fields as you go.

3. The Scalability:

- In almost all situations **SQL databases are vertically scalable**. This means that you can increase the load on a single server by increasing things like RAM, CPU or SSD.
- **NoSQL databases are horizontally scalable**. This means that you handle more traffic by sharing, or adding more servers in your NoSQL database. It is similar to adding more floors to the same building versus adding more buildings to the neighbourhood. Thus NoSQL can ultimately become larger and more powerful, making these databases the preferred choice for large or ever-changing data sets.

SQL vs No SQL

4. The Structure:

- **SQL databases are table-based**
- **NoSQL databases are either key-value pairs, document-based, graph databases or wide-column stores.** This makes relational SQL databases a better option for applications that require multi-row transactions such as an accounting system or for legacy systems that were built for a relational structure.

5. Property followed:

- **SQL databases follow ACID properties** (Atomicity, Consistency, Isolation and Durability) whereas
- **NoSQL database follows the Brewers CAP theorem** (Consistency, Availability and Partition tolerance).

6. Support:

- **Great support is available for all SQL database from their vendors.** Also a lot of independent consultations are there who can help you with SQL database for a very large scale deployments
- **NoSQL database you still have to rely on community support** and only limited outside experts are available for setting up and deploying your large scale NoSQL deployments.
- Some **examples of SQL databases include PostgreSQL, MySQL, Oracle and Microsoft SQL Server.**
- **NoSQL database examples include Redis, RavenDB Cassandra, MongoDB, BigTable, HBase, Neo4j and CouchDB, DynamoDB.**

MongoBD

- To run the mongo db server, from command prompt give command:

  mongod

C:\Users\kbhas>mongod --dbpath "C:\Program Files\MongoDB\Server\7.0\data\bapatla" → specifies the data files path.

- To display the database you are using, type db:

  db

- To switch databases, issue the use <db> helper, as in the following example:

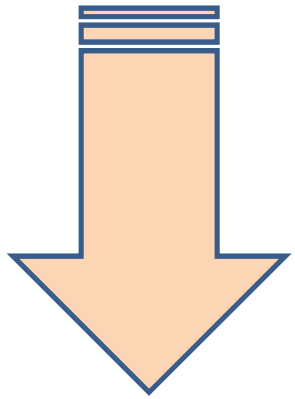
  use <database>

- To list the databases available to the user, use the helper show dbs.

  show dbs

Install Mongodb shell official location:

Using MongoDB Shell



MongoBD Shell

Step 1: start mongo db from command prompt using the command:

mongod

Step2: start mongo db shell from command prompt using the command:

mongosh

Step 3: show databases available

show dbs

Step 4: use a particular data base

use biketrip

Step 5: show tables in a database

show collections

Step 6: to clear the console screen: console.clear()

Step 6: exit from database: exit

- **use <database>:** Changes the current database handle. Database operations are processed on the current database handle.
- **db.help:** Displays help options for the database methods.
- **show <option>:** Shows a list based on the option argument. The value of option can be:
  - **dbs:** Displays a list of databases
  - **collections:** Displays a list of collections for the current database.
  - **profile:** Displays the five most recent system.profile entries taking more than 1 millisecond.
  - **databases:** Displays a list of all available databases.
  - **roles:** Displays a list of all roles for the current database, both built-in and user-defined.
  - **users:** Displays a list of all users for that database.
- **exit:** Exits the database.

MongoBD

- To display the database you are using, type db:

  db

Creating a new database and collection:

use <newdbName>

Ex: use tempdb → creates a new database 'tempdb'

Creating a Collection:

Using createCollection( ) method:

db.createCollection("newCollectionName", [options]);

db.createCollection("addresses"); → creates a collection named 'addresses' in current db.

Using insertOne( ) method:

db.<collectionName>.insertOne({ attribute:value});

Ex: db.addresses.insertOne({ cof:'xyz', houseno:'8-555', village: 'abcd'}

- If a collection does not exist, MongoDB creates the collection when you first add document for that collection.

ctrl C → terminates the current command execution

Managing collections

- MongoDB provides the functionality in the MongoDB shell to create, view, and manipulate collections in a database.
- **Creating a collection:**

  db.createCollection("newCollectionName", [options]);

  db.createCollection("addresses"); → creates a collection named 'addresses' in current db.

Retrieving inserted document:

db.test.find({ title: **"The Favourite"** }) → retrieves a document with field title='The favourite'

Inserting Multiple documents:

insertMany( )

- db.collection.insertMany() can insert *multiple* documents into a collection. Pass an array of documents to the method. If the documents do not specify an _id field, MongoDB adds the _id field with an ObjectId value to each document.

The insertMany() method has the following syntax:

- db.collection.insertMany([<document 1> , <document 2>, ...], { writeConcern: <document>, ordered: <boolean> })

- ordered: boolean
- A boolean specifying whether the mongod instance should perform an **ordered or unordered insert.** Defaults to true.

MongoBD

Table 12.4 Options that can be specified when creating collections

Role	Description
`capped`	A Boolean; when `true`, the collection is a capped collection that does not grow bigger than the maximum size specified by the `size` attribute. Default: `false`.
`autoIndexID`	A Boolean; when `true`, an `_id` field is automatically created for each document added to the collection and an index on that field is implemented. This should be `false` for capped collections. Default: `true`.
`size`	Specifies the size in bytes for the `capped` collection. The oldest document is removed to make room for new documents.
`max`	Specifies the maximum number of documents allowed in a `capped` collection. The oldest document is removed to make room for new documents.
`validator`	Allows users to specify validation rules or expressions for the collection.

MongoBD

`validator`	Allows users to specify validation rules or expressions for the collection.
`validationLevel`	Determines the strictness MongoDB applies to the validation rules on documents during updates.
`validationAction`	Determines whether an invalid document is errored or warned but still can be inserted.
`indexOptionDefaults`	Allows users to specify a default index configuration when a collection is created.

MongoBD – deleting documents

methods:

- db.collection.deleteOne()

  Delete at most a single document that match a specified filter even though multiple documents may match the specified filter.
- db.collection.deleteMany()

  Delete all documents that match a specified filter.
- db.collection.remove()

  Delete a single document or all documents that match a specified filter.

Additional methods:

- findOneAndDelete() provides a sort option. The option allows for the deletion of the first document sorted by the specified order.
- db.collection.findAndModify() provides a sort option. The option allows for the deletion of the first document sorted by the specified order.
- db.collection.bulkWrite().

Syntax:

deleteOne(filter, options, callback)

The Filter used to select the documents to remove

Ex: db.collection(**'inventory'**).deleteOne({ status: **'D'** });

Db.test.deleteOne({rated:'R'})

MongoBD – deleting documents

deleteMany( ) → You can specify criteria, or filters, that identify the documents to delete. The filters use the same syntax as read operations.

Syntax:

- db.collection.deleteMany(<filter>, { writeConcern: <document>, collation: <document> })
- filter → Specifies deletion criteria using query operators.
- To delete all documents in a collection, pass in an empty document ({ }).

Example:

db.orders.deleteMany({ **"client"** : **"Crude Traders Inc."** });

db.restaurants.deleteMany({ category: **"cafe"**, status: **"A"** }, { collation: { locale: **"fr"**, strength: 1 } })

MongoBD – updating documents

- Collection.updateOne() → Update a single document in a collection
- Collection.updateMany() → Update multiple documents in a collection
- Collection.replaceOne() → Replace a document in a collection with another document

updateOne( ):

Update a single document in a collection

Syntax:

updateOne(filter, update, options, callback)

filter → The Filter used to select the document to update

update → The update operations to be applied to the document

options → optional settings

Ex:

```
db.collection('inventory').updateOne(
  { item: 'paper' },
  {
    $set: { 'size.uom': 'cm', status: 'P' },
    $currentDate: { lastModified: true }
  }
);
```

MongoBD – updating documents

updateOne( ):

Example:

```
db.collection('inventory').updateOne(
  { item: 'paper' },
  {
    $set: { 'size.uom': 'cm', status: 'P' },
    $currentDate: { lastModified: true }
  });
```

Ex 2:

```
use sample_mflix
db.movies.updateOne( { title: "Twilight" },
{
  $set: {
    plot: "A teenage girl risks everything–including her life–when she falls in love with a
      vampire."
  },
  $currentDate: { lastUpdated: true }
})
```

Uses the $set operator to update the value of the plot field for the movie Twilight. Uses the $currentDate operator to update the value of the lastUpdated field to the current date.

MongoBD – updating multiple documents

updateMany( ):

To update all documents in the sample_airbnb.listingsAndReviews collection to update where security_deposit is less than 100:

Ex:

```
use sample_airbnb
db.listingsAndReviews.updateMany(
  { security_deposit: { $lt: 100 } },
  {
    $set: { security_deposit: 100, minimum_nights: 1 }
  }
)
```

The update operation uses the $set operator to update the value of the security_deposit field to 100 and the value of the minimum_nights field to 1.

MongoBD – Replace documents

- To replace the entire content of a document except for the _id field, pass an entirely new document as the second argument to db.collection.replaceOne().
- When replacing a document, the replacement document must contain only field/value pairs. Do not include update operators expressions.
- The replacement document can have different fields from the original document. In the replacement document, you can omit the _id field since the _id field is immutable; however, if you do include the _id field, it must have the same value as the current value.

Example:

```
db.accounts.replaceOne(
  { account_id: 371138 },
  { account_id: 893421, limit: 5000, products: [ "Investment", "Brokerage" ] }
)
```

replaces the *first* document from the *accounts* collection where *account_id: 371138*:

MongoBD – Additional Methods

The following methods can also update documents from a collection:

- db.collection.findOneAndReplace().
- db.collection.findOneAndUpdate().
- db.collection.findAndModify().
- db.collection.bulkWrite().

db.collection.findOneAndReplace()

Replaces a single document based on the specified filter.

db.collection.findOneAndReplace(filter, replacement, options)

Ex:

```
db.collection.findOneAndReplace(
  <filter>,
  <replacement>,
  {
    writeConcern: <document>,
    projection: <document>,
    sort: <document>,
    maxTimeMS: <number>,
    upsert: <boolean>,
    returnDocument: <string>,
    returnNewDocument: <boolean>,
    collation: <document>
  })
```

MongoBD – findOneAndUpdate

db.collection.findOneAndUpdate(filter, update, options)

- Returns the original document by default. Returns the updated document if returnNewDocument is set to true or returnDocument is set to after.

Syntax:

```
db.collection.findOneAndUpdate(
selection_criteria: <document>,
update_data: <document>,
{
  projection: <document>,
   sort: <document>,
   maxTimeMS: <number>,
   upsert: <boolean>,
   returnNewDocument: <boolean>,
   collation: <document>,
   arrayFilters: [ <filterdocument1>, … ]
})
```

Parameters: The first parameter is the selection criteria for the update. The type of this parameter is document.

- The second parameter is a document that to be updated. The type of this parameter is document.
- The third parameter is optional.

MongoBD –

Example 1:

```
> db.student.findOneAndUpdate({name:"Nikhil"},{$inc:{score:4}})
{
        "_id" : ObjectId("60226a70f19652db63812e8a"),
        "name" : "Nikhil",
        "language" : "c++",
        "score" : 245
}
```

Example 2:

```
> db.student.findOneAndUpdate({name:"Vishal"},{$inc:{score:4}},{returnNewDocumen
t:true})
{
        "_id" : ObjectId("60226a70f19652db63812e8b"),
        "name" : "Vishal",
        "language" : "python",
        "score" : 249
}
>
```

MongoBD – difference between findOneAndReplace() and findOneAndUpdate( )

Feature	findOneAndUpdate	findOneAndReplace
Update vs. Replace	Modifies specific fields within a document	Replaces the entire document with a new one
Atomic vs. Non-Atomic	Performs atomic updates, ensuring consistency in concurrent operations	Not atomic and may result in data inconsistency if multiple operations are performed simultaneously
Field-Level Control	Allows for granular control over which fields to update	Replaces all fields in the document

MongoBD –

- The **findAndModify()** method in MongoDB modifies and returns a single document that matches the given criteria. By default, **db.collection.findAndModify(document)** method returns a pre-modification document.
- To return the document with the modifications made on the update, use the new option and set its value to true. It takes a document as a parameter.
- If you want to find fields of the embedded document, then use the following syntax:

  “field.nestedfieldname”: <value>

  or

  {field: {nestedfieldname: <value>}}
- The document returned by this method always contains the _id field. If you don’t want the _id field, then set _id:0 in the projection.
- You can use this method in multi-document transactions.

MongoBD – findAndModify( )

Ex:

```
db.Collection_name.findAndModify(
{
   selection_criteria:<document>,
   sort: <document>,
   remove: <boolean>,
   update: <document>,
   new: <boolean>,
   fields: <document>,
   upsert: <boolean>,
   bypassDocumentValidation: <boolean>,
   writeConcern: <document>,
   collation: <document>,
   arrayFilters: [ <filterdocument1>, ... ]
})
```

- **Parameters:**
- **remove:** It is a must if the update field does not exist. If it is true, removes the selected document. The default value is false.
- **update:** It is a must if the remove field does not exist. Performs an update of the selected document. The update field employs the same update operators or field: value specifications to modify the selected document.
- Others are optional.

MongoBD – findAndModify( )

Optional parameters:

- **selection_criteria:** It specifies the selection criteria for the modification. The query field employs the same query selectors as used in the db.collection.find() method. Although the query may match multiple documents, findAndModify() will only select one document to modify.
- **sort:** Determines which document the operation will modify if the query selects multiple documents. findAndModify() will modify the first document in the sort order specified by this argument.
- **new:** When true, returns the modified document rather than the original. The findAndModify() method ignores the new option for remove operations. The default is false.
- **fields:** A subset of fields to return. The fields document specifies an inclusion of a field with 1, as in the following:
  - fields: { <field1>: 1, <field2>: 1, … }
- **Upsert:** The default value of this parameter is false. When it is true it will make a new document in the collection when no document matches the given condition in the update method.
- **writeConcern:** It is only used when you do not want to use the default write concern. The type of this parameter is a document.
- **Collation:** It specifies the use of the collation for operations. It allows users to specify the language-specific rules for string comparison like rules for lettercase and accent marks. The type of this parameter is a document.
- **arrayFilters:** It is an array of filter documents that indicates which array elements to modify for an update operation on an array field. The type of this parameter is an array.

MongoBD – findAndModify( )

- **MongoDB** is a versatile document-based NoSQL database and can perform DB write operations efficiently using its **bulkWrite()** method.
- The **db.collection.bulkWrite() method** allows multiple documents to be **inserted/updated/deleted** at once.
- **db.collection.bulkWrite()** method can be used in multi-document transactions.
- If this method encounters an error in the transaction, then it will throw a BulkWriteException.
- By default, this method executes operations in order.

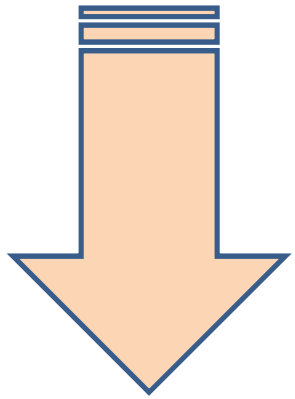
Syntax:

db.collection.bulkWrite(
[<opr1>, <opr2>, ...,<oprn>],
{
writeConcern : <your document and this is optional>,
ordered : <true/false, defaults to true and this is optional>
}
)

bulkWrite() --

- **Parameters:**
- **[<opr1>, <opr2>, …,<oprn>]:** It is an array of write operations, i.e., insertOne, updateOne, updateMany, deleteOne, deleteMany, replaceOne.
- **writeConcern:** It is a document that expresses write concern. If you want to use the default write concern then remove this parameter, It is an optional parameter.
- **ordered:** As multiple operations can be performed when ordered (default to true) is not provided, all the operations are proceeded one by one and if given as ordered : false, results of the operation differ as sometimes insertOne will be the first followed by the rest and sometimes deleteOne is followed first and if it is the case, without any document existence, it cannot be completed. Hence providing the “ordered” parameter to false should be taken into consideration whether required or not
- **Return:**
- This method will return a document that contains a boolean acknowledged as true (if the write concern is enabled) or false (if the write concern is disabled), count for every write operation, and an array that contains an _id for every successfully inserted or upserted document.

Accessing MongoDB from Node.js application



mongodb access from Node.js application