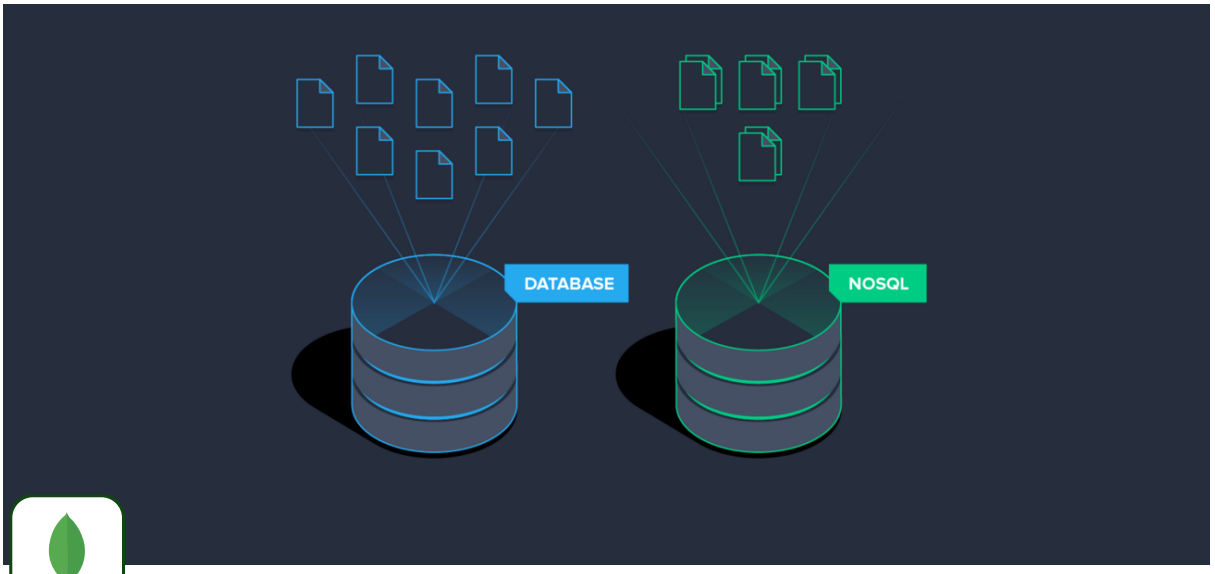

MongoDB Documentation

Database

A database is an organized collection of structured information, or data, typically stored electronically in a computer system. A database is usually controlled by a database management system (DBMS). Together, the data and the DBMS, along with the applications that are associated with them, are referred to as a database system, often shortened to just a database.

Differences between MongoDB and MySQL

MongoDB and MySQL both serve as robust database systems, but they differ in several significant ways:

1. **Structure:** MongoDB is a NoSQL database that stores data in flexible, JSON-like documents. This means fields can vary from document to document and data structure can be changed over time. On the other hand, MySQL is a relational database that uses tables to store data.

2. **Scaling:** MongoDB is designed for horizontal scaling by sharding data across many servers. MySQL, however, is typically vertically scaled by increasing the horsepower of the server.
3. **Query Language:** MongoDB uses a unique query language with powerful features like indexing, ad hoc queries, and real-time aggregation. MySQL uses the more traditional SQL (Structured Query Language) for querying data.
4. **Speed:** Due to its internal architecture, MongoDB can handle large amounts of unstructured data and is often faster for read and write operations.

MongoDB

MongoDB is a source-available cross-platform document-oriented database program. Classified as a NoSQL database program, MongoDB uses JSON-like documents with optional schemas.

MongoDB → noSQL, document, non-relational DBMS
→ Cassandra, HBase
SQL → RDBMS (Relational Database Management System)

MongoDB Structure:

- MongoDB's Physical database contains several logical databases.
- Each database contains several collections. The collection is something like tables in relational databases.
- Each collection contains several documents. A document is something like a record/row in a relational database.

SQL Terms	MongoDB Terms

Database	Database
Table	Collection
Entity/Row	Document
Column	Key/Field
Table Join	Embedded Document
Primary Key	Primary Key (Default key **_id** provided by MongoDB itself)

Data Formats in MongoDB:

10kb JSON : **{name:'skillqode'}** → BSON and that BSON will be stored

4kb End-user/Developer will provide data in JSON form.

In MongoDB server, data will be stored in BSON Form.

Key Characteristics of MongoDB

1. The installation and configuration process is straightforward and user-friendly.
2. All data related to a specific document is stored in a unified location, eliminating the need for join operations and consequently accelerating data retrieval.
3. Given that documents are schema-less and independent of each other, it allows for the storage of unstructured data such as video and audio files.
4. Operations such as editing existing documents, deleting documents, and inserting new documents can be executed with ease.
5. The data retrieved is in the form of JSON, enhancing interoperability as it can be readily understood by any programming language without the need for conversion.

MongoDB Shell vs MongoDB Server

- Upon the successful installation of MongoDB, you will have access to two Javascript-based applications: MongoDB Shell and MongoDB Server.
- The MongoDB Server, also known as '**mongod**', is primarily responsible for handling the storage of data within the database.
- MongoDB Shell, or '**mongosh**', is used to manage the Server.

setup **db** folder path for environment variable

```
    mongod --dbpath "C:\data\db"
```

Default Databases:

MongoDB Admin will use these default databases.

```
show dbs

// result
admin 0.000GB
config 0.000GB
local 0.000GB
```

Admin:

admin database is used to store user authentication and authorization information like usernames, passwords, roles, etc

This database is used by administrators while creating, deleting, and updating users and while assigning roles.

config:

To store configuration information of MongoDB server.

local:

local database can be used by the admin while performing replication process.

ObjectId: ObjectId is not JSON type and it is of BSON type. For every document, MongoDB Server will associate a unique ID, which is nothing but ObjectId. ObjectId is 12 bytes.

- The first 4 bytes represent the timestamp when this document was inserted.
- The next 3 bytes represent machine identifiers (hostname) where the ObjectId was generated.
- The next 2 bytes represent process or thread identifier of the generating process.
- The next 3 bytes represent some random increment value.

Creating a Database:

To create a database in MongoDB, use the `use DATABASE_NAME` command. If the specified database does not exist, MongoDB creates a new database.
For example:

```
use MyDatabase
```

This command creates a new database named '**MyDatabase**'.

Creating a Collection:

Collections in MongoDB are like tables in relational databases. You can create a collection using the `db.createCollection(name, options)` command. Here, 'name' is the name of the collection, and 'options' is a document for specifying the configuration of the collection.
For example:

```
db.createCollection("students")
```

This command creates a new collection named '**students**'.

Inserting Documents:

To insert data into MongoDB collection, you will need to use `insert()` or `insertOne()` and `insertMany()` commands. The `insertMany()` function can insert multiple documents at a time.
For example:

```
// insert one document
db.students.insertOne({
   "name": 'John',
   "age": 20,
   "subjects": ['Math', 'Science']
});

// insert Many documents
db.students.insertMany([
{
   "name": 'Smith',
   "age": 21,
   "subjects": ['Math', 'Science']
},
{
   "name": 'Carry',
   "age": 18,
   "subjects": ['Math', 'Science']
```

```
}
]);
```

This command inserts documents into 'students' collection.

Querying Documents:

You can query documents in a collection using `find()` command. The `find()` function returns all documents that match a query.
For example:

```
db.students.find({"name": 'John'});
```

This command finds all documents in the 'students' collection where 'name' is 'John'.

```
db.students.findOne({"name": 'John'});
```

This command finds one document in the 'students' collection where 'name' is 'John'.

```
//show specific data only in a document using projection
db.students.findOne({"name": 'John'}, {"_id":1, "name":1 });
```

Updating Documents:

To update a document, MongoDB provides `updateOne()` and `updateMany()` command. The `updateOne()` and `updateMany()` function updates the values of the existing document.
For example:

```
// update a one document
db.students.updateOne({"name": 'John'}, {$set: {age: 21}},
{upsert:true});
```

```
// update many documents
db.students.updateMany({"name": 'John'}, {$set: {age: 21}},
{upsert:true})
```

This command updates the age of '**John**' to 21 in the '**students**' collection.

MongoDB Update Operators:

There are many update operators that can be used during document updates.

Fields:

The following operators can be used to update fields:

$currentDate: Sets the field value to the current date

$inc: Increments the field value

$rename: Renames the field

$set: Sets the value of a field

$unset: Removes the field from the document

Array:

The following operators assist with updating arrays.

$addToSet: Adds distinct elements to an array.

$pop: Removes the first or last element of an array.

$pull: Removes all elements from an array that match the query.

$push: Adds an element to an array.

Deleting Documents:

To delete a document or a collection, you can use `deleteOne()` or `deleteMany()` commands. The `deleteOne()` function deletes a document

from the collection.
For example:

```
// delete a document
db.students.deleteOne({"name": 'John'});

// delete many documents
db.students.deleteMany({"name": 'John'});
```

This command deletes documents where 'name' is 'John' from the 'students' collection.

Drop Statement:

To drop a collection or a database, you can use `db.collection.drop()` or `db.dropDatabase()` command. The `db.collection.drop()` function drop a collection from the current database.
For example:

```
// Drop a Collection
db.collection.drop();

// Drop Current Database
db.dropDatabase();
```

Import file into Database

Database-tools (command line database tool)

Steps for importing database:

1. Download the above tool
2. Locate the folder where MongoDB tool is downloaded.
3. Open the "bin" folder inside the MongoDB tool downloaded.
4. Copy the 'mongoimport.exe' file from the "bin" folder.

5. Look in the "**Program Files**" folder on your computer.
6. Open the **MongoDB** folder inside "**Program Files**".
7. Inside the **MongoDB** folder, find and open the "**Server**" folder.
8. Go into the "**bin**" folder within the "**Server**" folder.
9. Paste the '**mongoimport.exe**' file you copied into this "**bin**" folder.

1. JSON File Import in MongoDB

- **Make a JSON file.**
- open terminal or command prompt and type **mongod**.
- open terminal where JSON file is located type

```
mongoimport --db databaseName --collection collectionName --f
```

2. JS File Import in MongoDB

- **Make a JS file.**
- open terminal or command prompt and type **mongod**.
- open terminal or command prompt and type **mongosh**.

```
load("fileName.js")
```

Comparison operator

The following operators can be used in queries to compare values:

$eq: It is used To match the value of the fields that are equal to the specified value.

```
db.collection.find({"field": { $eq : value }})
```

$ne: It is used to match all values of the field that are not equal to specified value.

```
db.collection.find({"field": { $ne: value }})
```

$gt: It is used to match the values of the field that are greater than a specified value.

```
db.collection.find({"field": { $gt: value }});
```

$gte: It is used to match values of the field that are greater than equal to the specified value.

```
db.collection.find({"field": { $gte: value }});
```

$lt: It is used to match values of the field that are less than a specified value.

```
db.collection.find({"field": { $lt: value }});
```

$lte: It is used to match values of the fields that are less than equals to the specific values.

```
db.collection.find({"field": { $lte: value }});
```

$in: It is used to match any of the values specified in an array.

```
db.collection.find({"field": { $in: [val1, val2, …] }});
```

$nin: It is used to match none of the values specified in an array.

```
db.collection.find({"field": { $nin: [val1, val2, …] }});
```

Logical operator

The following operators can logically compare multiple queries.

$and: It is used to join query clauses with a logical **AND** and return all documents that match the given conditions of both clauses.

```
db.collection.find( { $and: [ {exp1},{exp2},...] } );
```

$or: It is used to join query clauses with a logical **OR** and return all documents that match the given conditions of either clause.

```
db.collection.find( { $or: [ {exp1},{exp2},...] } );
```

$nor: it is used to join query clauses with a logical **NOR** and return all documents that fail to match both clauses.

```
db.collection.find( { $nor: [ {exp1},{exp2},...] } );
```

$not: It is used to invert the effect of the query expressions and return documents that do not match the query expression.

```
db.collection.find( {"field": { $not: { exp1 }}});
```

MongoDB Aggregation:

Aggregation operations process data records and return computed results. MongoDB provides three ways to perform aggregation: the aggregation pipeline, the map-reduce function, and single-purpose aggregation methods.

- `$match`: This is a filter operation. It filters the documents to pass only documents that match the specified condition(s) to the next pipeline stage.

```
db.collection.aggregate([
  { $match: { "field": value } }
]);

db.collection.find({ "field": value })
```

- `$project`: This operation can add new fields, remove existing fields, or reshape a document's data.

```
db.collection.aggregate([
  { $project: { _id: 0, "field": 1, "field": 0 } }
]);

db.collection.find({},{"_id": 0, "field": 1, "field": 1});
```

- `$limit`: This operation can restricts the number of documents that can pass through the pipeline.

```
db.collection.aggregate([
  { $limit: 10 },  // Limits the output to the first 10 docum
]);

db.collection.find().limit(number)
```

- `$skip`: This operation can used to skip a specified number of documents in the pipeline.

```
db.collection.aggregate([
  { $skip: 5 },  // Skips the first 5 documents
]);

db.collection.find().skip(number);
```

- `$count`: This operation can used to counts total amount of documents passed from the previous stage in the pipeline.

```
db.collection.aggregate([
  { $count: "field" },
]);

db.collection.find().count();
```

- `$sort`: This operation can used to sort all documents in the specified order.

```
db.collection.aggregate([
  { $sort: { "field": 1 } },
]);

db.collection.find().sort({"field": 1});
```

```
// 1 - Ascending Order
// -1 - Descending Order
```

- `$group`: This operation can used to group documents based on a specified key or keys and perform various aggregate operations on the grouped data.

```
db.collection.aggregate([
  {
            $group: {
        _id: "$field",
                total: { $sum: "$field" }
            },
  },
]);
```

- `$lookup`: This operation can used to perform a left outer join between documents from the current collection and documents from another collection. It allows you to combine documents from different collections based on a common field or expression.
  - **from**: The collection to use for lookup in the same database
  - **localField**: The field in the primary collection that can be used as a unique identifier in the from collection.
  - **foreignField**: The field in the form a collection that can be used as a unique identifier in the primary collection.
  - **as**: The name of the new field that will contain the matching documents from the from collection.

```
db.collection.aggregate([
  {
            $lookup: {
```

```
                    from: "",
                            localField: "",
                            foreignField: "",
                            as: ""
                },
    },
]);
```

- `$unwind`: This operation is used to separate the elements of an array field from the input documents, generating a new document for each element.

```
db.students.aggregate([
  { $unwind: "$field" }
]);
```