**Non SQL/ NoSQL database**

1. Key-value database
2. No rows and columns, No tables

Ex: Mongo DB

**Mongo DB**

Note:

1. Not RDBMS
2. Not Joins

**Database** – collections of collection

**Collection** – Group of document (relate to table, no schema structure)

Can have any number of document

Documents can have dynamic schema; documents can be same or different

User defined scheme, it is not static or fixed

**Document** – set of key-value pair, each document unique key \_id

When a document is inserted, automatically a unique id

Ex: - 1 {“first name”:” Trupthi”,” last name”:” kale”}

Ex: - 2 {“first name”:” Trupthi”}

These are 2 documents

**Data Type**

1. JSON
2. Integer
3. Boolean
4. Double
5. Arrays
6. Object
7. Null
8. Date
9. Timestamp
10. Object Id
11. Code
12. BSON

Mongo DB data type. Extended version of JSON. Binary encoded JSON. It has extended data types which are not supported by JSON.

Like: - Date

Timestamp

Object Id

**Commands**

Commands should be opened in ‘C:\Program Files\MongoDB\Server\5.0\bin>’ folder.

mongo.exe

To see database – **show databases;**

To create or use database - **use database-name;**

Once the database is created, it won’t be shown, collection should be created to see it.

To see which database we are working on – **db;**

To delete database - **db.dropDatabase();**

To create the collection – **db.createCollection("collection-name");**

To delete the collection – **db.collection-name.drop();**

To see all collections - **show collections;**

Inserting the data/document in collection

1. Inserting single document - **db.collection-name.insert({"key”: “value"});**

Ex: - db.Contacts.insert({"name”: “Trupthi"});

1. Inserting multiple documents - **db. collection-name.insertMany(**

**[**

**{**

**"key1":"value1"**

**}, //document1**

**{**

**"key1":"value1",**

**"key2":"value2"**

**} //document2**

**]**

**);**

Ex: - db.Contacts.insertMany([{"name":"abc"},{"first name":"abc","last name":"xyz"}]);

Every Document we insert will be unique key “\_id”, it will be 24 character. It is just like primary key.

We can change the value of “\_id”? Yes, but it is suggested not to do.

Updating the documents

1. Update single document

**db.Contacts.update(**

**{"existing-key":"existing-value"}, // just like where condition**

**{**

**$set:**

**{"key": “value”} //what has to be updated**

**}**

**);**

Ex: - db.Contacts.update({"name":"Trupthi"},{$set:{"isActive":false}});

1. Update multiple document

**db.Contacts.updateMany(**

**{"existing-key":"existing-value"}, // just like where condition**

**{**

**$set:**

**{"key": “value”} //what has to be updated**

**}**

**);**

Ex: - db.Contacts.updateMany({"name":"abc"},{$set:{"isActive":false}});

Read data

1. To find all the documents - **db. collection-name.find();**
2. To find first record - **db. collection-name.findOne();**
3. To find with passing condition - **db.collection-name.find({"key":"value"}); // you can pass single key & value pair or multiple key & value pair for condition**
4. To find document and replace –

**db.Contacts.findOneAndReplace(**

**{"existing key ":"existing value"}, // it will search the document**

**{"key":"value"} // complete document will be replaced**

**);**

Ex: - db.Contacts.findOneAndReplace({"name":"abc"},{"name":"abc xyz"});

1. To find document and delete - **db.Contacts.findOneAndDelete({"existing key":"existing value"}); // it will search the document with condition and delete it**

Ex: - db.Contacts.findOneAndDelete({"name":"abc xyz"});

Deleting the document in the collection

1. To delete only one record that matches the condition – **db.Contacts.deleteOne({"existing key":"existing value"});**

Ex: - db.Contacts.deleteOne({"name":"Trupthi"});

1. To delete multiple records that matches the condition - **db.Contacts.deleteMany({"existing key":"existing value "});**

Ex: - db.Contacts.deleteMany({"name":"Trupthi"});

Using find methods with operators:

1. eq – Equality
2. lt – Less than
3. lte – Less than Equal
4. gt – Greater than
5. gte – Greater than Equal
6. $and
7. $or

Usage: ex: - **db.Contacts.find({"salary":{$lte:18000}});**

**db.Contacts.find({$and : [{"salary":{$gt:15000}},{"salary":{$lt:19000}}]});**

If you want to restrict any key and values pair you can do it as below(Projection)

**db.Contacts.find({}, {"salary":0});** // 0 means don’t display, 1 means display

**Aggregate Stages:**

1. $count
2. $group
3. $limit
4. $lookup
5. $match
6. $merge
7. $sort
8. $project
9. $unwind
10. $unset

Aggregate is a function that takes list as an argument. It is used just like find method. To the list we can pass key and values pairs. Key will be any of these stages, value will be condition. This condition will also have key and value pair.

Ex: - **Sort**

var pipeline = [

{

$sort:

{"salary":1} // Here 1 means ascending order. In the output salary will be sorted in ascending order. -1 means descending order.

}

];

db.collection-name.aggregate(pipeline);

Ex: - **limit**

var pipeline = [{$limit:4}];

db.collection-name.aggregate(pipeline);

Limit and Skip

**Limit** – It limits the number of records. Only specified number of records will be displayed in the output

Ex: - db.collection-name.find().limit(4);

It the output only 4 records will be displayed.

**Skip** – It skips the records sequentially. Specified number of records will be skipped in the output sequentially.

Ex: - db.collection-name.find().skip(3);

It first 3 records from the output.

**Sort** - It is used to sort the output in ascending (1), descending (-1) order.

Ex: - db.collection-name.find().sort({"salary":-1});

Sort is a function that takes key and value pair as argument. Value will be 1 or -1.

**Indexing** – It increases the speed of retrieval of data.