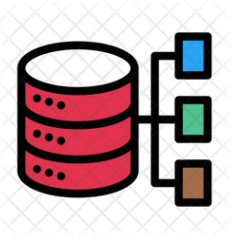
MongoDB

Tushar B. Kute,
http://tusharkute.com



MongoDB

- MongoDB is an open-source document database that provides high performance, high availability, and automatic scaling.
- In simple words, you can say that - Mongo DB is a document-oriented database. It is an open source product, developed and supported by a company named 10gen.
- MongoDB is available under General Public license for free, and it is also available under Commercial license from the manufacturer.
- The manufacturing company 10gen has defined MongoDB as:
- "MongoDB is a scalable, open source, high performance, document-oriented database." - 10gen
- MongoDB was designed to work with commodity servers. Now it is used by the company of all sizes, across all industry.

MongoDB: Document

- MongoDB is a document-oriented database. It is a key feature of MongoDB. It offers a document-oriented storage. It is very simple you can program it easily.
- MongoDB stores data as documents, so it is known as document-oriented database.
  - FirstName = "Tushar",
  - Address = "Pune",
  - Spouse = [{Name: "Rashmi"}].
  - FirstName ="Tushar",
  - Address = "Kute"

MongoDB: Advantages over RDBMS

- MongoDB is schema less. It is a document database in which one collection holds different documents.
- There may be difference between number of fields, content and size of the document from one to other.
- Structure of a single object is clear in MongoDB.
- There are no complex joins in MongoDB.
- MongoDB provides the facility of deep query because it supports a powerful dynamic query on documents.
- It is very easy to scale.
- It uses internal memory for storing working sets and this is the reason of its fast access.

Data Types

- String
  - String is the most commonly used datatype. It is used to store data. A string must be UTF 8 valid in mongodb.
- Integer
  - Integer is used to store the numeric value. It can be 32 bit or 64 bit depending on the server you are using.
- Boolean
  - This datatype is used to store boolean values. It just shows YES/NO values.
- Double
  - Double datatype stores floating point values.
- Min/Max Keys
  - This datatype compare a value against the lowest and highest bson elements.

Data Types

- Arrays
  - This datatype is used to store a list or multiple values into a single key.
- Object
  - Object datatype is used for embedded documents.
- Null
  - It is used to store null values.
- Symbol
  - It is generally used for languages that use a specific type.
- Date
  - This datatype stores the current date or time in unix time format. It makes you possible to specify your own date time by creating object of date and pass the value of date, month, year into it.

Installation on Ubuntu Linux

```
sudo apt-get update
sudo apt-get install mongodb
```

- To start the mongodb type:

```
mongo
```

MongoDB Compass

- MongoDB compass is nothing but a graphical user interface that can be connected to the MongoDB database and used to find, analyze, modify, and visualize the data stored in the database without requiring any knowledge of queries.
- MongoDB acts as an alternative to Mongo Shell. Mongo Shell can also perform all the aforementioned tasks but requires a lot of technical expertise.
- MongoDB is GUI-based whereas Shell is more technicality based i.e it uses specific commands and queries. MongoDB Compass is present in the Github repo since it is an open-source tool.

MongoDB Compass

- All the data stored in the database can be visualized and explored.
- Data stored in the database can be modified, created, updated, or deleted.
- Shows the Server Statistics in Real-time.
- The Visual representations clearly depict the performance issues and recommend plans.
- All the indexes can be managed.
- JSON schema validation rules can be sued to validate data.
- A wide range of plugins is available.

MongoDB Compass: Installation

- Directly download the required files using the command mentioned below(using wget command). To open the terminal click on ctrl+alt+T.

```
wget https://downloads.mongodb.com/compass/mongodb-compass_1.30.1_amd64.deb
```

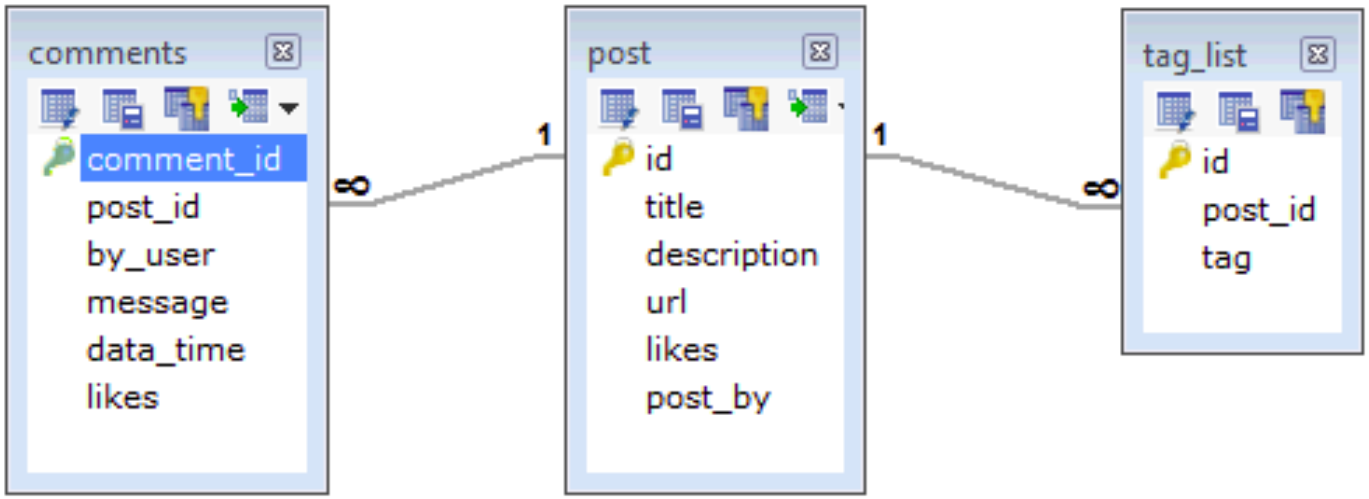
- If you have downloaded the file using the terminal use the following command to install the MongoDB Compass Ubuntu.

```
sudo dpkg -i mongodb-compass_1.30.1_amd64.deb
```

Data Model in MongoDB

- In MongoDB, data has a flexible schema. It is totally different from SQL database where you had to determine and declare a table's schema before inserting data. MongoDB collections do not enforce document structure.
- The main challenge in data modeling is balancing the need of the application, the performance characteristics of the database engine, and the data retrieval patterns.

Data Model in MongoDB



Create Database

- There is no create database command in MongoDB. Actually, MongoDB do not provide any command to create database.
- It may be look like a weird concept, if you are from traditional SQL background where you need to create a database, table and insert values in the table manually.
- Here, in MongoDB you don't need to create a database manually because MongoDB will create it automatically when you save the value into the defined collection at first time.

Create Database

- If there is no existing database, the following command is used to create a new database.
- Syntax:

```
use DATABASE_NAME
```

- If the database already exists, it will return the existing database.
- To check the database list, use the command show dbs:

```
> show dbs
```

Drop Database

- The dropDatabase command is used to drop a database. It also deletes the associated data files. It operates on the current database.
- Syntax:

```
db.dropDatabase()
```

- This syntax will delete the selected database. In the case you have not selected any database, it will delete default "test" database.

Create Collection

- In MongoDB, db.createCollection(name, options) is used to create collection. But usually you don?t need to create collection.
- MongoDB creates collection automatically when you insert some documents.
- Syntax:

```
db.createCollection(name, options)
```

- Here, Name: is a string type, specifies the name of the collection to be created.

Create Collection: Options

- Capped Boolean (Optional)
  - If it is set to true, enables a capped collection. Capped collection is a fixed size collecction that automatically overwrites its oldest entries when it reaches its maximum size. If you specify true, you need to specify size parameter also.
- AutoIndexID Boolean (Optional)
  - If it is set to true, automatically create index on ID field. Its default value is false.

Create Collection: Options

- Size Number (Optional)
  - It specifies a maximum size in bytes for a capped collection. Ifcapped is true, then you need to specify this field also.
- Max Number (Optional)
  - It specifies the maximum number of documents allowed in the capped collection.

Create Collection: Example

- Let's take an example to create collection. In this example, we are going to create a collection name mydata.

```
>use test
switched to db test
>db.createCollection("mydata")
{ "ok" : 1 }
```

- To check the created collection, use the command "show collections".

```
> show collections
```

Create Collection: Example

- MongoDB creates collections automatically when you insert some documents.
- For example: Insert a document named name into a collection named mydata. The operation will create the collection if the collection does not currently exist.

```
>db.mydata.insert({"name" : "tushar"})
>show collections
mydata
```

- If you want to see the inserted document, use the find() command.

```
db.collection_name.find()
```

Drop Collection

- In MongoDB, db.collection.drop() method is used to drop a collection from a database.
- It completely removes a collection from the database and does not leave any indexes associated with the dropped collections.
- The db.collection.drop() method does not take any argument and produce an error when it is called with an argument.
- This method removes all the indexes associated with the dropped collection.
- Syntax:

  db.COLLECTION_NAME.drop()

Insert document

- In MongoDB, the db.collection.insert() method is used to add or insert new documents into a collection in your database.
- Upsert
  - There are also two methods "db.collection.update()" method and "db.collection.save()" method used for the same purpose. These methods add new documents through an operation called upsert.
  - Upsert is an operation that performs either an update of existing document or an insert of new document if the document to modify does not exist.

Insert document

- Syntax

```
>db.COLLECTION_NAME.insert(document)
```

Insert document

```
db.mydata.insert(
  {
    course: "BDA",
    details: {
      duration: "6 months",
      Trainer: "Tushar Kute"
    },
    Batch: [ { size: "Small", qty: 10 }, { size: "Medium", qty: 30 } ],
    category: "Data Science"
  }
)
```

Check the document

- If the insertion is successful, you can view the inserted document by the following query.

```
> db.mydata.find()
```

- You will get the inserted document in return.

Query Document

- The find() Method:
- To query data from MongoDB collection, you need to use MongoDB's find() method.
- Syntax:
  - Basic syntax of find() method is as follows

    `db.`*`COLLECTION_NAME`*`.find()`
- find() method will display all the documents in a non structured way.

Query Document

- The pretty() Method:
- To display the results in a formatted way, you can use pretty() method.
- Syntax:

```
db.collection_name.find().pretty()
```

- Apart from find() method there is findOne() method, that reruns only one document.

RDBMS Clauses equivalent to MongoDB

- Equality:
  - Syntax: `{<key>:<value>}`
  - Example:

    ```
    db.institute.find({"by":"Mitu Research"}).pretty()
    ```

  - RDBMS equivalent:

    ```
    where by = 'Mitu Research'
    ```

- Less than:
  - Syntax: `{<key>:{$lt:<value>}}`
  - Example:

    ```
    db.institute.find({"likes":{$lt:50}}).pretty()
    ```

  - RDBMS equivalent:

    ```
    where likes < 50
    ```

RDBMS Clauses equivalent to MongoDB

- Less than or equal to:
  - Syntax: `{<key>:{$lte:<value>}}`
  - Example:

    ```
    db.institute.find({"likes":{$lte:50}}).pretty()
    ```
  - RDBMS equivalent:

    ```
    where likes <= 50
    ```
- Greater than:
  - Syntax: `{<key>:{$gt:<value>}}`
  - Example:

    ```
    db.institute.find({"likes":{$gt:50}}).pretty()
    ```
  - RDBMS equivalent:

    ```
    where likes > 50
    ```

RDBMS Clauses equivalent to MongoDB

- Greater than or equal to:
  - Syntax: `{<key>:{$gte:<value>}}`
  - Example:

    ```
    db.institute.find({"likes":{$gte:50}}).pretty()
    ```

  - RDBMS equivalent:

    ```
    where likes >= 50
    ```

- Not equal to:
  - Syntax: `{<key>:{$ne:<value>}}`
  - Example:

    ```
    db.institute.find({"likes":{$ne:50}}).pretty()
    ```

  - RDBMS equivalent:

    ```
    where likes != 50
    ```

AND and OR Operations

- In the find() method, if you pass multiple keys by separating them by ',' then MongoDB treats it AND condition. Basic syntax of AND is shown below:

```
db.mycol.find({key1:value1, key2:value2}).pretty()
```

- To query documents based on the OR condition, you need to use $or keyword. Basic syntax of OR is shown below:
- `db.mycol.find({$or: [{key1: value1}, {key2:value2}]}`

AND Operation Example

```
> db.institute.find({"by":"Mitu Research","title": "NoSQL Database"}).pretty()
{
        "_id" : ObjectId("579b0030329b317d9a0ab0bb"),
        "title" : "NoSQL Database",
        "description" : "NoSQL database doesn't have tables",
        "by" : "Mitu Research",
        "url" : "http://www.mitu.co.in",
        "tags" : [
                "mongodb",
                "database",
                "NoSQL"
        ],
        "likes" : 20
}
```

OR Operation Example

```
> db.institute.find({$or:[{"by":"Mitu Research","title": "NoSQL Database"}]}).pretty()
{
        "_id" : ObjectId("579b0030329b317d9a0ab0bb"),
        "title" : "NoSQL Database",
        "description" : "NoSQL database doesn't have tables",
        "by" : "Mitu Research",
        "url" : "http://www.mitu.co.in",
        "tags" : [
                "mongodb",
                "database",
                "NoSQL"
        ],
        "likes" : 20
}
```

Insert multiple documents

- If you want to insert multiple documents in a collection, you have to pass an array of documents to the db.collection.insert() method.
- Create an array of documents
- Define a variable named Allcourses that hold an array of documents to insert.

Insert multiple documents

```
var Allcourses =
   [
    {
      Course: "Java",
      details: { Duration: "6 months", Trainer: "Tushar Kute" },
      Batch: [ { size: "Medium", qty: 25 } ],
      category: "Programming Language"
    },
    {
      Course: "Python",
      details: { Duration: "3 months", Trainer: "Rashmi Thorave" },
      Batch: [ { size: "Small", qty: 10 }, { size: "Large", qty: 30 } ],
      category: "Programming Language"
    }
   ];
```

Insert multiple documents

- Inserts the documents

- Pass this Allcourses array to the db.collection.insert() method to perform a bulk insert.

```
> db.mydata.insert( Allcourses );
```

Update Documents

- In MongoDB, update() method is used to update or modify the existing documents of a collection.

- Syntax:

  db.COLLECTION_NAME.update(SELECTIOIN_CRITERIA, UPDATED_DATA)

Update Documents

- Update the existing course "java" into "python":

  > db.mydata.update({'course':'java'},{$set: {'course':'python'}})

- Check the updated document in the collection:

  > db.mydata.find()

Update Documents

- By default mongodb will update only single document, to update multiple you need to set a paramter 'multi' to true.
- `db.institute.update({"by":"Mitu Research"}, {$set:{"title":"MongoDB Tutorial"}}, {multi:true})`

```
> db.institute.update({"by":"Mitu Research"},{$set:{"title":"MongoDB Tutorial"}})
> db.institute.find()
{ "_id" : ObjectId("5798d3b6e102bcdf8d057377"), "name" : "tushar kute" }
{ "_id" : ObjectId("579b0030329b317d9a0ab0ba"), "title" : "MongoDB Overview", "description" : "MongoDB is no s
ql database", "by" : "Mitu Skillologies", "url" : "http://www.mitu.co.in", "tags" : [  "mongodb",  "database",
  "NoSQL" ], "likes" : 1000 }
{ "_id" : ObjectId("5798d311e102bcdf8d057376"), "name" : "tushar kute", "title" : "MongoDB Tutorial" }
{ "_id" : ObjectId("579b0030329b317d9a0ab0bb"), "by" : "Mitu Research", "description" : "NoSQL database doesn'
t have tables", "likes" : 20, "tags" : [  "mongodb",  "database",  "NoSQL" ], "title" : "MongoDB Tutorial", "u
rl" : "http://www.mitu.co.in" }
```

Update Documents

- MongoDB Save() Method:

  The save() method replaces the existing document with the new document passed in save() method
- Syntax:

```
db.COLLECTION_NAME.save({_id:ObjectId(),NEW_DATA})
```

Update Documents - Examples

- db.institute.save({"_id":ObjectId("5798d311e102bcdf8d057376"),"name":"rashmi thorave"})

```
> db.institute.find()
{ "_id" : ObjectId("579b0030329b317d9a0ab0ba"), "title" : "MongoDB Overview", "description" : "MongoDB is no sql database", "by" : "Mitu Skillologies", "url" : "http://www.mitu.co.in", "tags" : [ "mongodb", "database", "NoSQL" ], "likes" : 1000 }
{ "_id" : ObjectId("5798d311e102bcdf8d057376"), "name" : "tushar kute", "title" : "MongoDB Tutorial" }
{ "_id" : ObjectId("579b0030329b317d9a0ab0bb"), "by" : "Mitu Research", "description" : "NoSQL database doesn't have tables", "likes" : 20, "tags" : [ "mongodb", "database", "NoSQL" ], "title" : "MongoDB Tutorial", "url" : "http://www.mitu.co.in" }
{ "_id" : ObjectId("5798d3b6e102bcdf8d057377"), "name" : "tushar kute", "title" : "MongoDB Tutorial" }
> db.institute.save({"_id":ObjectId("5798d311e102bcdf8d057376"),"name":"rashmi thorave"})
> db.institute.find()
{ "_id" : ObjectId("579b0030329b317d9a0ab0ba"), "title" : "MongoDB Overview", "description" : "MongoDB is no sql database", "by" : "Mitu Skillologies", "url" : "http://www.mitu.co.in", "tags" : [ "mongodb", "database", "NoSQL" ], "likes" : 1000 }
{ "_id" : ObjectId("5798d311e102bcdf8d057376"), "name" : "rashmi thorave" }
{ "_id" : ObjectId("579b0030329b317d9a0ab0bb"), "by" : "Mitu Research", "description" : "NoSQL database doesn't have tables", "likes" : 20, "tags" : [ "mongodb", "database", "NoSQL" ], "title" : "MongoDB Tutorial", "url" : "http://www.mitu.co.in" }
{ "_id" : ObjectId("5798d3b6e102bcdf8d057377"), "name" : "tushar kute", "title" : "MongoDB Tutorial" }
```

Delete Documents

- The remove() Method:
- MongoDB's remove() method is used to remove document from the collection. remove() method accepts two parameters. One is deletion criteria and second is justOne flag

  1. deletion criteria : (Optional) deletion criteria according to documents will be removed.

  2. justOne : (Optional) if set to true or 1, then remove only one document.

- Syntax:

```
db.COLLECTION_NAME.remove(DELETION_CRITERIA)
```

Delete Documents

- `db.institute.remove({'title':'MongoDB Overview'})`

```
> db.institute.find()
{ "_id" : ObjectId("579b0030329b317d9a0ab0ba"), "title" : "MongoDB Overview", "description" : "MongoDB is no s
ql database", "by" : "Mitu Skillologies", "url" : "http://www.mitu.co.in", "tags" : [  "mongodb",  "database",
  "NoSQL" ], "likes" : 1000 }
{ "_id" : ObjectId("5798d311e102bcdf8d057376"), "name" : "rashmi thorave" }
{ "_id" : ObjectId("579b0030329b317d9a0ab0bb"), "by" : "Mitu Research", "description" : "NoSQL database doesn'
t have tables", "likes" : 20, "tags" : [  "mongodb",  "database",  "NoSQL" ], "title" : "MongoDB Tutorial", "u
rl" : "http://www.mitu.co.in" }
{ "_id" : ObjectId("5798d3b6e102bcdf8d057377"), "name" : "tushar kute", "title" : "MongoDB Tutorial" }
> db.institute.remove({'title':'MongoDB Overview'})
> db.institute.find()
{ "_id" : ObjectId("5798d311e102bcdf8d057376"), "name" : "rashmi thorave" }
{ "_id" : ObjectId("579b0030329b317d9a0ab0bb"), "by" : "Mitu Research", "description" : "NoSQL database doesn'
t have tables", "likes" : 20, "tags" : [  "mongodb",  "database",  "NoSQL" ], "title" : "MongoDB Tutorial", "u
rl" : "http://www.mitu.co.in" }
{ "_id" : ObjectId("5798d3b6e102bcdf8d057377"), "name" : "tushar kute", "title" : "MongoDB Tutorial" }
```

Delete Documents

- Remove only one
  - If there are multiple records and you want to delete only first record, then set justOne parameter in remove() method

    ```
    db.COLLECTION_NAME.remove(DELETION_CRITERIA,1)
    ```
- Remove All documents
  - If you don't specify deletion criteria, then mongodb will delete whole documents from the collection. This is equivalent of SQL's truncate command.

    ```
    db.mycol.remove()
    ```

    ```
    db.mycol.find()
    ```

Projection

- In mongodb projection meaning is selecting only necessary data rather than selecting whole of the data of a document. If a document has 5 fields and you need to show only 3, then select only 3 fields from them.
- The find() Method
- MongoDB's find() method accepts second optional parameter that is list of fields that you want to retrieve. In MongoDB when you execute find() method, then it displays all fields of a document.
- To limit this you need to set list of fields with value 1 or 0. 1 is used to show the filed while 0 is used to hide the field.
- Syntax:

```
db.COLLECTION_NAME.find({},{KEY:1})
```

Projection – Example

```
> db.institute.find({},{"title":1,_id:0})
{  }
{ "title" : "MongoDB Tutorial" }
{ "title" : "MongoDB Tutorial" }
```

Limit Records

- The limit() Method:

  To limit the records in MongoDB, you need to use limit() method. limit() method accepts one number type argument, which is number of documents that you want to displayed.

- Syntax:

```
db.COLLECTION_NAME.find().limit(NUMBER)
```

Limit Records

```
> db.institute.find({},{"title":1,_id:0})
{  }
{ "title" : "MongoDB Tutorial" }
{ "title" : "MongoDB Tutorial" }
> db.institute.find({},{"title":1,_id:0}).limit(2)
{  }
{ "title" : "MongoDB Tutorial" }
```

Skip Records

- The skip() method:

  Apart from limit() method there is one more method skip() which also accepts number type argument and used to skip number of documents.

- Syntax:

```
db.COLLECTION_NAME.find().limit(NUMBER).skip(NUMBER)
```

Skip Records

```
> db.institute.find({},{"title":1,_id:0})
{  }
{ "title" : "MongoDB Tutorial" }
{ "title" : "MongoDB Tutorial" }
> db.institute.find({},{"title":1,_id:0}).limit(2).skip(2)
{ "title" : "MongoDB Tutorial" }
```

Sort Documents

- The sort() Method:

  To sort documents in MongoDB, you need to use sort() method. sort() method accepts a document containing list of fields along with their sorting order. To specify sorting order 1 and -1 are used. 1 is used for ascending order while -1 is used for descending order.

- Syntax:

```
db.COLLECTION_NAME.find().sort({KEY:1})
```

Sort Documents

- `db.institute.find().sort({"title":-1})`

```
> db.institute.find().sort({"title":-1})
{ "_id" : ObjectId("579b0030329b317d9a0ab0bb"), "by" : "Mitu Research", "description" : "NoSQL database doesn'
t have tables", "likes" : 20, "tags" : [  "mongodb",  "database",  "NoSQL" ], "title" : "MongoDB Tutorial", "u
rl" : "http://www.mitu.co.in" }
{ "_id" : ObjectId("5798d3b6e102bcdf8d057377"), "name" : "tushar kute", "title" : "MongoDB Tutorial" }
{ "_id" : ObjectId("5798d311e102bcdf8d057376"), "name" : "rashmi thorave" }
```

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 sql count(*) and with group by is an equivalent of mongodb aggregation.
- The aggregate() Method:

  Syntax:

  ```
  db.COLLECTION_NAME.aggregate(AGGREGATE_OPERATION)
  ```

Aggregation

```
> db.institute.insert(
... {
... 'title': 'MongoDB Overview',
... 'description': 'MongoDB is no sql database',
... 'by_user': 'rashmi thorave',
... 'url': 'http://www.mitu.co.in',
... 'tags': ['mongodb', 'database', 'NoSQL'],
... 'likes': 100
... },
... {
... title: 'NoSQL Overview',
... description: 'No sql database is very fast',
... by_user: 'tushar kute',
... url: 'http://www.tusharkute.com',
... tags: ['mongodb', 'database', 'NoSQL'],
... likes: 10
... },
... {
... title: 'Neo4j Overview',
... description: 'Neo4j is no sql database',
... by_user: 'Neo4j',
... url: 'http://www.neo4j.com',
... tags: ['neo4j', 'database', 'NoSQL'],
... likes: 750
... })
```

Aggregation

```
> db.institute.aggregate([{$group : {_id : "$by_user", num_tutorial : {$sum : 1}}}])
{
        "result" : [
                {
                        "_id" : "rashmi thorave",
                        "num_tutorial" : 1
                }
        ],
        "ok" : 1
}
```

Sql equivalent query for the above use case will be

```
select by_user, count(*) from mycol group by by_user
```

Aggregation functions

- `db.institute.aggregate([{$group : {_id: "$by_user", num_tutorial : {$sum :"$likes"}}}])`

Expression	Description	Example
$sum	Sums up the defined value from all documents in the collection.	db.mycol.aggregate([{$group : {_id : "$by_user", num_tutorial : {$sum : "$likes"}}}])
$avg	Calculates the average of all given values from all documents in the collection.	db.mycol.aggregate([{$group : {_id : "$by_user", num_tutorial : {$avg : "$likes"}}}])
$min	Gets the minimum of the corresponding values from all documents in the collection.	db.mycol.aggregate([{$group : {_id : "$by_user", num_tutorial : {$min : "$likes"}}}])
$max	Gets the maximum of the corresponding values from all documents in the collection.	db.mycol.aggregate([{$group : {_id : "$by_user", num_tutorial : {$max : "$likes"}}}])

Distinct

- Finds the distinct values for a specified field across a single collection.
- distinct returns a document that contains an array of the distinct values.
- The return document also contains an embedded document with query statistics and the query plan.

Distinct

- The following example returns the distinct values for the field dept from all documents in the mydata collection:

- db.runCommand ({ distinct: "mydata", key: "dept" })

- db.runCommand(
  {
  distinct: "restaurants",
  key: "rating",
  query: { cuisine: "italian" },
  }
)

Upsert

- Upsert is a method that is used to insert and update the value in any operation. In other words, the MongoDB upsert method is a combination of insert and update (insert + update = upsert).
- By default, the upsert method's value is always false. If the document matches the specified query and the method's value is set to true, the update operation will update the matching documents.
- If the document does not match the specified query and the method's value is set to true, this method inserts a new document in the collection. This new document contains the fields that indicate to the operation.

Upsert

- The user can use the upsert option in conjunction with the findAndModify() function. In this function, the default value of the option is false. If the value of this option is set to true, the function performs one of the following operations:
  - If a document is found that matches the given query criteria, the findAndModify() function updates the document.
  - If no document matches the given query criteria, the findAndModify() function inserts a new document into the collection.

Upsert

- 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>, ?]
- })

Upsert

- db.student.findAndModify({query:{roll:3}, update:{$set:{address:"Pune", marks:67.23}}, upsert:true})

- db.emp.update({Name: "Henry"}, // Query parameter

  {$set: {Phone Number: '9654785423 '}, // Update document

  $setOnInsert: {Gender: 'Male'}},

  {upsert: true})

MongoDB Operators

- $nin
  - The $nin operator chooses the documents where the field value is not in the specified array or does not exist.
  - Syntax:

    { field : { $nin: [<value1>, <value2>, ...] } }
  - Example:

    db.books.find ({ price: { $nin: [50, 150, 200] } })

MongoDB Operators

- $not

  The $not operator works as a logical NOT on the specified expression and chooses the documents that are not related to the expression.

- Syntax:

  ```
  { field: { $not: { <operator-expression> } } }
  ```

- Example:

  ```
  db.books.find ( { price: { $not: { $gt: 200 } } } )
  ```

MongoDB Operators

- $exists
- The exists operator matches the documents that contain the field when Boolean is true. It also matches the document where the field value is null.
- Syntax:

  { field: { $exists: <boolean> } }
- Example:

  db.books.find ({ qty: { $exists: true } })

Index

- An index is a data structure that allows us to add indexes in the existing table. It enables you to improve the faster retrieval of records on a database table.
- It creates an entry for each value of the indexed columns. We use it to quickly find the record without searching each row in a database table whenever the table is accessed.
- We can create an index by using one or more columns of the table for efficient access to the records.

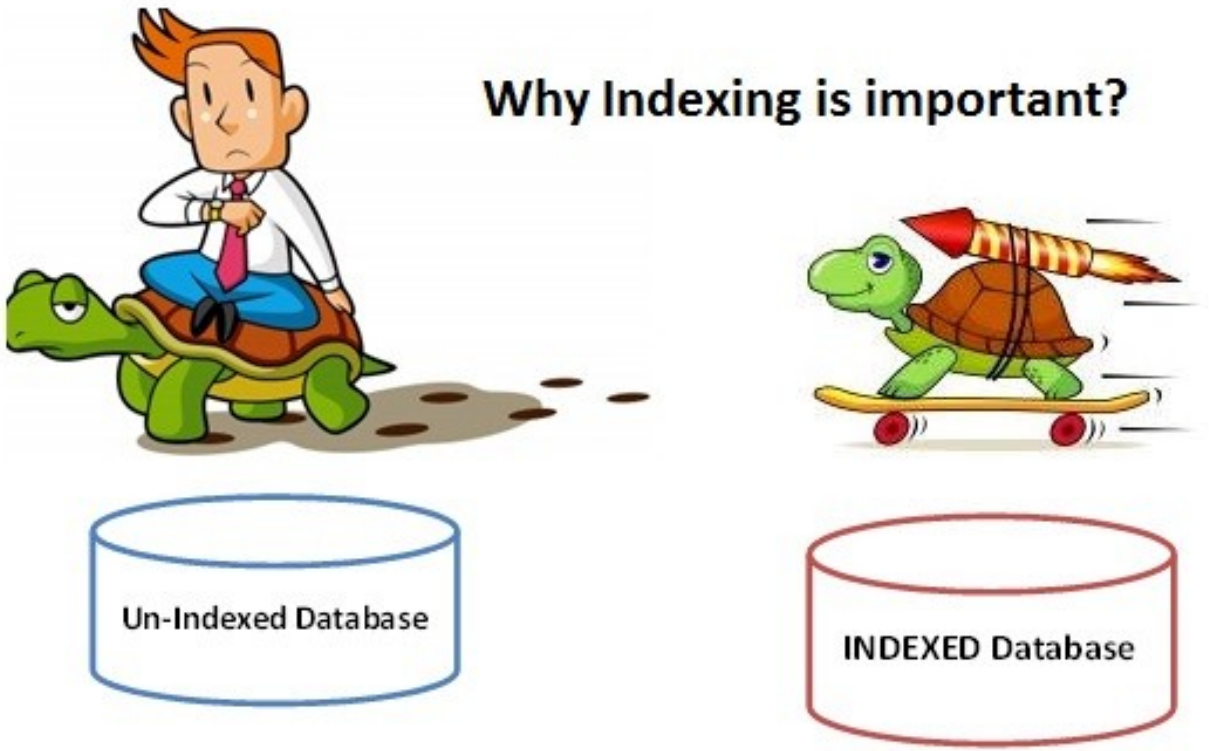
Index

- When a table is created with a primary key or unique key, it automatically creates a special index named PRIMARY.
- We called this index as a clustered index. All indexes other than PRIMARY indexes are known as a non-clustered index or secondary index.

Indexing need

- Suppose we have a contact book that contains names and mobile numbers of the user. In this contact book, we want to find the mobile number of Martin Williamson.
- If the contact book is an unordered format means the name of the contact book is not sorted alphabetically, we need to go over all pages and read every name until we will not find the desired name that we are looking for.
- This type of searching name is known as sequential searching.

Why indexing ?



Index

- Indexes support the efficient resolution of queries. Without indexes, MongoDB must scan every document of a collection to select those documents that match the query statement.
- This scan is highly inefficient and require MongoDB to process a large volume of data.
- Indexes are special data structures, that store a small portion of the data set in an easy-to-traverse form.
- The index stores the value of a specific field or set of fields, ordered by the value of the field as specified in the index.

Index

- To create an index, you need to use createIndex() method of MongoDB.
- Syntax:

  The basic syntax of createIndex() method is as follows().

  > db.COLLECTION_NAME.createIndex({KEY:1})
- Here key is the name of the field on which you want to create index and 1 is for ascending order.

Index

- To create index in descending order you need to use -1.
- Example

```
>db.mycol.createIndex({"title":1})
{
"createdCollectionAutomatically" : false,
"numIndexesBefore" : 1,
"numIndexesAfter" : 2,
"ok" : 1
}
```

Index

- In createIndex() method you can pass multiple fields, to create index on multiple fields.

```
>db.mycol.createIndex({"title":1,"description":-1})
```

Index

- You can drop a particular index using the dropIndex() method of MongoDB.
- Syntax:

  > db.COLLECTION_NAME.dropIndex({KEY:1})

- Here, "key" is the name of the file on which you want to remove an existing index. Instead of the index specification document (above syntax), you can also specify the name of the index directly as:

  dropIndex("name_of_the_index")

Index

- Example:

```
> db.mycol.dropIndex({"title":1})
{
"ok" : 0,
"errmsg" : "can't find index with key: { title:
1.0 }",
"code" : 27,
"codeName" : "IndexNotFound"
}
```

Index

- This method deletes multiple (specified) indexes on a collection.
- Syntax:

  > db.COLLECTION_NAME.dropIndexes()
- Example
- Assume we have created 2 indexes in the named mycol collection as shown below –

  > db.mycol.createIndex({"title":1,"description":-1})
- Following example removes the above created indexes of mycol –

  > db.mycol.dropIndexes({"title":1,"description":-1})

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

Index

- The getIndexes() method

- This method returns the description of all the indexes int the collection.

  Syntax:

  ```
  db.COLLECTION_NAME.getIndexes( )
  ```

MongoDB vs. SQL Similiarities

- Although we call MongoDB and MySQL a database, both of them are used to create databases.
- We create a lot of tables in MySQL, similarly, in the case of MongoDB we call it a collection, which means the equivalent thing of the table in the world of MongoDB is called a collection.
- In the case of MySQL, you will come to know a lot about the Index, and the same thing is there for MongoDB.

MongoDB vs. SQL Similiarities

- In the MySQL world, we have the primary key, and it consists of a huge impact on linking the tables. Similarly, we have the same when it comes to MongoDB. Here, each document stored in a collection requires a unique _id field. It is equivalent to what the primary key is in the case of MongoDB.
- Another key point is that in MySQL we have the concept of rows, we create a lot of rows to enter the data. Similarly the same stands for MongoDB, but it is termed as BSON documents or JSON also.
- MySQL and MongoDB support the programming languages such as Node.JS, Java, and Python.

MongoDB Language Bindings

- MongoDB loves developers. And we want to make software engineers everywhere successful with building applications—whether that's a serverless web app, a mobile game, an IoT integration, or any other application you can imagine.
- MongoDB is the most popular NoSQL databases and is widely adopted for storing and managing both structured and unstructured data.
- Data administrators, analysts, and programmers can use the programming language of their choice to optimize and manage data, and create highly performant applications.

Reference: mongodb.com

MongoDB Language Bindings

- The languages officially supported and compatible with MongoDB are:
  - C
  - C++
  - C#
  - Go
  - Java
  - Node.js
  - PHP
  - Python
  - Ruby
  - Rust
  - Scala
  - Swift

Thank you

@mitu_skillologies

@mITuSkillologies

@mitu_group

@mitu-skillologies

@MITUSkillologies

kaggle

@mituskillologies

Web Resources
https://mitu.co.in
http://tusharkute.com

@mituskillologies

contact@mitu.co.in

tushar@tusharkute.com