Database- Day -3: MongoDB:

"Why mongodb?

MongoDB is a NoSQL, **document-oriented database** that provides a **flexible and scalable way to store, retrieve, and manage data.**

It is particularly well-suited for handling large amounts of **unstructured or semi-structured data**. MongoDB stores data in flexible, **JSON-like documents**, allowing for easy modification and adaptation of the database schema.

What is document?

a document is the basic unit of **data storage and retrieval**. It is a JSON-like **(BSON, Binary JSON)** data structure that contains **key-value pairs**.

A document represents a **single record or entry** in a MongoDB collection that documents within a collection can have **different structures**. Unlike traditional relational databases.

Eg:

{ "name": "John Doe",

"age": 30,

"city": "New York" }

{ "name": "suresh",

"age": 35,

"city": "xxxx",

phonenum:155521241445 }

What is collection?

a collection is a **grouping of MongoDB documents**. It is the **equivalent of an RDBMS table**.

A collection exists within a single MongoDB database and **holds sets of documents**, which are JSON-like BSON objects.

Mongodb vs MySQL:

Feature	MongoDB	MySQL
Database Structure	Document-Oriented (JSON-like documents)	Relational (Tables with rows and columns)
Schema	Schema-less (Flexible, dynamic schema)	Schema-based (Structured, predefined schema)
Query Language	MongoDB Query Language (document-focused)	SQL (Structured Query Language)
Use Cases	Big data applications, mobile apps eg:Real-Time Analytics	Traditional business applications, e-commerce

Performance	Read and write optimized for large data sets	Efficient for complex queries, well-designed

Example Explanation: [for below three titles]

[creation of database, collections, documents

use of operators in find() query"

find - query & projection]

Eg:

1.creation of database, collections, documents:

Basic Create Read Update Delete (CRUD) Operations

Assume you have a MongoDB database with a "products" collection containing documents like this:

Eg:

```
[
  {  "name": "Laptop", "price": 999, "stock": 10 },
  {  "name": "Smartphone", "price": 599, "stock": 20 },
  {  "name": "Tablet", "price": 299, "stock": 15 }
]
```

create database command

test> **use mydb;**

switched to db mydb

mydb>

This command **does not create the database immediately**. The database is actually created as soon as y**ou insert the first document into a collection** within that database.

create Collection & insert data into the collection.

Insert Document: (Create)

Eg:

db.products.insertOne({"name":"smartphone","price":599,"stock":20});

db.products.insertMany([{"name":"Tablet","price":299,"stock":15},{"name":"headphone","price":99 ,"stock":30}]);

Find: select or read document from the collection

db.products.findOne({"name":"Laptop"});

db.products.find();

2.use of operators in find() query :

some more examples using operators:

$gt, $lt, $gte, $lte,$eq ,$ne

comparison operators:

In MongoDB, operators like **$gt and $lt** are used in query operations to compare values in documents against specified criteria.

you can use the $gte (greater than or equal to) and $lte (less than or equal to) operators in your query.

Eg:

db.products.find({"price":{$gt:300}});

db.products.find({"price":{$lt:300}});

db.products.find({"price":{$lte:299}});

db.products.find({'price':{$gte:599}});

db.products.find({"price":{$gt:400,$lt:600}});

#eq , $ne:

Eg:

db.products.find({"price":{$eq:500}});

db.products.find({"price":{$ne:500}});

$and $or Operator:

$and Operator: This operator combines **multiple conditions** and requires that all of them be true for a document to match.

Syntax:

db.collection.find({ $and:[{},{}]})

Eg:

```
db.products.find({$and:[{"price":{$gt:400}},{"stock":{$lte:20}}]});
```

$or Operator: This operator combines multiple conditions and requires that **at least one** of them be true for a document to match.

Eg:

```
db.products.find({$or:[{"price":{$gt:600}},{"stock":{$eq:20}}]});
```

Update:

Syntax:

```
db.products.updateOne({field:value},{$set:{field:value}})
```

Eg:

```
db.products.updateOne({"name":"Laptop"},{$set:{"stock":25}});

db.products.updateOne(
{_id:ObjectId("65df6fd92a87d72558dbb208")},

   {$set:{'name':'Headphones'}});

db.products.updateMany({'price':{$lt:500}},{$set:{'stock':50}});
```

note: use carefully

Delete:

Eg:

```
db.products.deleteOne({_id:ObjectId("65df6fd92a87d72558dbb207")}
);

db.products.deleteOne({"name":"Laptop"});

db.products.deleteMany({"price":{$lt:400}}); // don't use it
```

Count:

method in MongoDB to count the **number of documents** that match a specific query condition.

Eg:

```
db.products.countDocuments({"price":{$gt:500,$lt:1000}});

db.products.countDocuments();
```

3.find - query & projection:

$project Stage: (0,1) excluding:0 including:1

```
db.products.find({},{'name':1,'price':1});
```

```
db.products.find({},{'_id':0,'name':1,'price':1});
```

```
db.products.find({},{'stock':0});
```

```
db.products.find({},{'_id':0,'stock':0});
```

db.products.find({},{_id:0,stock:1, price:1, name:0});

MongoServerError[Location31254]: Cannot do exclusion on field name in inclusion projection

We can use _id:0 in inclusion fields but not other fields

We could not exclusion(0) any one field inclusion(1)

Practise: any 5 questions

Task Questions: (have a discussion)

1. Find all the information about each products
2. Find the product price which are between 400 to 800
3. Find the product price which are not between 400 to 600
4. List the four product which are greater than 500 in price
5. Find the product name and product material of each products
6. Find the product with a row id of 10
7. Find only the product name and product material
8. Find all products which contain the value of soft in product material
9. Find products which contain product color indigo and product price 492.00
10. Delete the products which product price value are same