Practical: 2

Aim: Practical of Data collection, Data curation and management for Large-scale Data system (such as MongoDB)

Data curation is management of the data lifecycle. The term is typically applied to efforts to collect and preserve historical data. The following are common elements of data curation.

Overview: Data Curation	
Type	Data
Definition	Management of the data lifecycle especially in regards to historical data.
Related Concepts	Data Governance Data Management Data Profiling Data Backup Data Lineage Retention Schedule Compliance Data Quality Data Availability Data Disposal

```
MongoDB shell version v4.2.25
connecting to: mongodb://127.0.0.1:27017/?compressors=disabled&gssapiServiceName=mongodb
Implicit session: session { "id" : UUID("b5a198bc-391d-49ae-8498-bd7c74650b24") }
MongoDB server version: 4.2.25
Welcome to the MongoDB shell.
For interactive help, type "help".
For more comprehensive documentation, see
        https://docs.mongodb.com/
Questions? Try the MongoDB Developer Community Forums
        https://community.mongodb.com
Server has startup warnings:
2024-02-14T23:33:01.972+0530 I  CONTROL  [initandlisten]
2024-02-14T23:33:01.972+0530 I  CONTROL  [initandlisten] ** WARNING: Access control is not enabled for the database.
2024-02-14T23:33:01.972+0530 I  CONTROL  [initandlisten] **          Read and write access to data and configuration is unrestricted.
2024-02-14T23:33:01.973+0530 I  CONTROL  [initandlisten]
---
Enable MongoDB's free cloud-based monitoring service, which will then receive and display
metrics about your deployment (disk utilization, CPU, operation statistics, etc).

The monitoring data will be available on a MongoDB website with a unique URL accessible to you
and anyone you share the URL with. MongoDB may use this information to make product
improvements and to suggest MongoDB products and deployment options to you.

To enable free monitoring, run the following command: db.enableFreeMonitoring()
To permanently disable this reminder, run the following command: db.disableFreeMonitoring()
---

>
```

1.MongoDB Create database:

To display the name of current database:

```
> db
test
```

```
> use tycs
switched to db tycs
```

```
> show dbs
admin   0.000GB
config  0.000GB
local   0.000GB
```

2. MongoDB Drop Database:

The syntax to drop a Database is:

```
db.dropDatabase()
```

We do not specify any database name in this command, because this command deletes the currently selected database. Let's see the steps to drop a database in MongoDB.

```
> db.dropDatabase()
{ "dropped" : "tycs", "ok" : 1 }
>
>
> show dbs
admin   0.000GB
config  0.000GB
local   0.000GB
>
```

3.MongoDB Create collection:

Method 1: Creating the Collection in MongoDB on the fly

The cool thing about MongoDB is that you need not to create collection before you insert document in it. With a single command you can insert a document in the collection and the MongoDB creates that collection on the fly.

Syntax: db.collection_name.insert({key:value, key:value…})

```
> db.tycs.insert({no:1 , name:"tycs1", grade:"O", age:20})
WriteResult({ "nInserted" : 1 })
>
```

To check whether the collection is created successfully, use the following command.

```
show collections
```

This command shows the list of all the collections in the currently selected database.

```
> show collections
tycs
```

4. MongoDB Drop collection:

To drop a collection , first connect to the database in which you want to delete collection and then type the following command to delete the collection:

```
db.collection_name.drop()
```

```
> db.tycs.drop()
true
```

5.MongoDB Insert Document:

Syntax to insert a document into the collection:

```
db.collection_name.insert()
```

Lets take an example to understand this .

MongoDB Insert Document using insert()

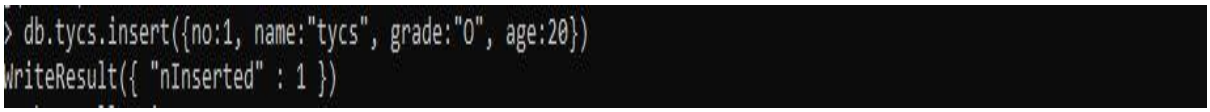
Example:

Here we are inserting a document into the collection named “tycs”. The field “no” in the example below is an array that holds the several key-value pairs.

You should see a successful write message like this:

```
WriteResult({ "nInserted" : 1 })
```

The insert() method creates the collection if it doesn't exist but if the collection is present then it inserts the document into it:



MongoDB Example: Insert Multiple Documents in collection

To insert multiple documents in collection, we define an array of documents and later we use the insert() method on the array variable as shown in the example below. Here we are inserting three documents in the collection named "tycs". This command will insert the data in "tycs" collection, if the collection is not present then it will create the collection and insert these documents.

As you can see that it shows number 3 in front of **nInserted**. this means that the 3 documents have been inserted by this command.

6.MongoDB Query Document:

Find():

Querying all the documents in JSON format

Lets say we have a collection tycs in a database named tycs. To get all the documents we use this command:

```
db.students.find()
```

```
> db.tycs.find()
{ "_id" : ObjectId("65cc6ce63bac159ac5c22b7d"), "no" : 1, "name" : "tycs", "grade" : "O", "age" : 20 }
{ "_id" : ObjectId("65cc6d423bac159ac5c22b7e"), "no" : 2, "name" : "tycs2", "grade" : "A", "age" : 21 }
{ "_id" : ObjectId("65cc728b3bac159ac5c22b7f"), "no" : 3, "name" : "tycs3", "grade" : "B", "age" : 20 }
{ "_id" : ObjectId("65cc728b3bac159ac5c22b80"), "no" : 4, "name" : "tycs4", "grade" : "O", "age" : 19 }
{ "_id" : ObjectId("65cc728b3bac159ac5c22b81"), "no" : 5, "name" : "tycs5", "grade" : "A", "age" : 21 }
```

However, the output we get is not in any format and less-readable. To improve the readability, we can format the output in JSON format with this command:

```
db.students.find().forEach(printjson);
```

OR simply use pretty () – It does the same thing.

```
> db.tycs.find().pretty()
{
        "_id" : ObjectId("65cd055135caf8c0a42802ba"),
        "no" : 1,
        "name" : "tycs1",
        "grade" : "O",
        "age" : 20
}
{
        "_id" : ObjectId("65cd07d835caf8c0a42802bb"),
        "no" : 1,
        "name" : "tycs1",
        "grade" : "O",
        "age" : 20
}
{
        "_id" : ObjectId("65cd081a35caf8c0a42802bc"),
        "no" : 3,
        "name" : "tycs3",
        "grade" : "B",
        "age" : 20
}
{
        "_id" : ObjectId("65cd081a35caf8c0a42802bd"),
        "no" : 4,
        "name" : "tycs4",
        "grade" : "O",
        "age" : 19
}
{
        "_id" : ObjectId("65cd081a35caf8c0a42802be"),
        "no" : 5,
```

Query Document based on the criteria:

Instead of fetching all the documents from collection, we can fetch selected documents based on a criteria.

Equality Criteria:

For example: To fetch the data of “tycs3” from tycs collection. The command for this should be:

```
> db.tycs.find({name:"tycs3"}).pretty()
{
        "_id" : ObjectId("65cc728b3bac159ac5c22b7f"),
        "no" : 3,
        "name" : "tycs3",
        "grade" : "B",
        "age" : 20
}
```

Greater Than Criteria:
Syntax:

```
db.collection_name.find({"field_name":{$gt:criteria_value}}).pretty()
```

For example: To fetch the details of student having age > 20 then the query should be:

Less than Criteria:

Syntax:

```
db.collection_name.find({"field_name":{$lt:criteria_value}}).pretty()
```

Example: Find all the students having age less than 20. The command for this criteria would be:

Not Equals Criteria:

Syntax: db.collection_name.find({"field_name":{$ne:criteria_value}}).pretty()

Example:
Find all the students where age is not equal to 21. The command for this criteria would be:

```
> db.tycs.find({"age":{$ne:21}}).pretty()
{
        "_id" : ObjectId("65cc6ce63bac159ac5c22b7d"),
        "no" : 1,
        "name" : "tycs",
        "grade" : "O",
        "age" : 20
}
{
        "_id" : ObjectId("65cc728b3bac159ac5c22b7f"),
        "no" : 3,
        "name" : "tycs3",
        "grade" : "B",
        "age" : 20
}
{
        "_id" : ObjectId("65cc728b3bac159ac5c22b80"),
        "no" : 4,
        "name" : "tycs4",
        "grade" : "O",
        "age" : 19
}
```

7.MongoDB Update Document:

In MongoDB, we have two ways to update a document in a collection. 1) update() method 2) save() method. Although both the methods update an existing document, they are being used in different scenarios. The update() method is used when we need to update the values of an existing document while save() method is used to replace the existing document with the document that has been passed in it.

To update a document in MongoDB, we provide a criteria in command and the document that matches that criteria is updated.

Updating Document using update() method

Syntax:

```
db.collection_name.update(criteria, update_data)
```

To update the name of “tycs2” with the name “BNB”. The command for this would be:

```
> db.tycs.update({"name":"tycs2"},{$set:{"name":"BNB"}})
WriteResult({ "nMatched" : 1, "nUpserted" : 0, "nModified" : 1 })
```

```
> db.tycs.update({"name":"tycs1"},{$set:{"name":"BNB"}})
WriteResult({ "nMatched" : 1, "nUpserted" : 0, "nModified" : 1 })
> db.tycs.find().pretty()
{
        "_id" : ObjectId("65cd055135caf8c0a42802ba"),
        "no" : 1,
        "name" : "BNB",
        "grade" : "O",
        "age" : 20
}
{
        "_id" : ObjectId("65cd07d835caf8c0a42802bb"),
        "no" : 1,
        "name" : "tycs1",
        "grade" : "O",
        "age" : 20
}
{
        "_id" : ObjectId("65cd081a35caf8c0a42802bc"),
        "no" : 3,
        "name" : "tycs3",
        "grade" : "B",
        "age" : 20
```

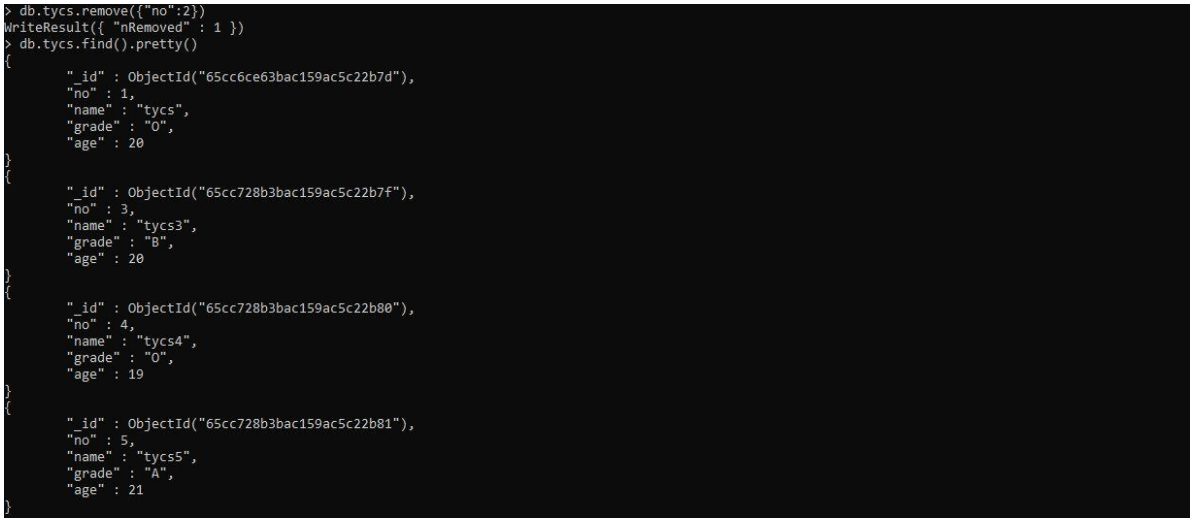
8.Delete document in MongoDB:

MongoDB Delete Document from a Collection

In this tutorial we will learn how to delete documents from a collection. The remove() method is used for removing the documents from a collection in MongoDB.

Syntax of remove() method:

```
db.collection_name.remove(delete_criteria)
```



Remove all Documents

If you want to remove all the documents from a collection but does not want to remove the collection itself then you can use remove() method like this:

```
db.collection_name.remove({})
```

9.MongoDB Projection:

This is used when we want to get the selected fields of the documents rather than all fields.

For example, we have a collection where we have stored documents that have the fields: no, name, grade, age but we want to see only the no of all the students then in that case we can use projection to get only the no of students.

Syntax: db.collection_name.find({},{field_key:1 or 0})

```
> db.tycs.find({},{"_id":0,"no":1})
{ "no" : 1 }
{ "no" : 2 }
{ "no" : 3 }
{ "no" : 4 }
{ "no" : 5 }
```

Value 1 means show that field and 0 means do not show that field. When we set a field to 1 in Projection other fields are automatically set to 0, except _id, so to avoid the _id we need to specifically set it to 0 in projection. The vice versa is also true when we set few fields to 0, other fields set to 1 automatically.

10. limit() method in MongoDB:

The limit() method in MongoDB

This method limits the number of documents returned in response to a particular query. Syntax:

```
db.collection_name.find().limit(number_of_documents)
```

```
> db.tycs.find().limit(1).pretty()
{
        "_id" : ObjectId("65cc6ce63bac159ac5c22b7d"),
        "no" : 1,
        "name" : "tycs",
        "grade" : "O",
        "age" : 20
}
```

11. MongoDB Skip() Method

The skip() method is used for skipping the given number of documents in the Query result.

To understand the use of skip() method, lets take the same example that we have seen above. In the above example we can see that by using limit(1) we managed to get only one document, which is the first document that matched the given criteria. What if you do not want the first document matching your criteria. For example we have two documents that have student_id value greater than 2002 but when we limited the result to 1 by using limit(1), we got the first document, in order to get the second document matching this criteria we can use skip(1) here which will skip the first document.

```
> db.tycs.find().limit(1).skip(1).pretty()
{
        "_id" : ObjectId("65cc6d423bac159ac5c22b7e"),
        "no" : 2,
        "name" : "BNB",
        "grade" : "A",
        "age" : 21
}
```

12.Sorting of Documents in MongoDB:

Sorting Documents using sort() method

Using sort() method, you can sort the documents in ascending or descending order based on a particular field of document.

Syntax of sort() method:

```
db.collecttion_name.find().sort({field_key:1 or -1})
```

1 is for ascending order and -1 is for descending order. The default value is 1.

For example: collection tycs contains following documents:

```
> db.tycs.find().pretty()
{
        "_id" : ObjectId("65cd055135caf8c0a42802ba"),
        "no" : 1,
        "name" : "tycs1",
        "grade" : "O",
        "age" : 20
}
{
        "_id" : ObjectId("65cd07d835caf8c0a42802bb"),
        "no" : 1,
        "name" : "tycs1",
        "grade" : "O",
        "age" : 20
}
{
        "_id" : ObjectId("65cd081a35caf8c0a42802bc"),
        "no" : 3,
        "name" : "tycs3",
        "grade" : "B",
        "age" : 20
}
{
        "_id" : ObjectId("65cd081a35caf8c0a42802bd"),
        "no" : 4,
        "name" : "tycs4",
        "grade" : "O",
        "age" : 19
}
{
        "_id" : ObjectId("65cd081a35caf8c0a42802be"),
        "no" : 5,
```

To display the data of students in **ascending order** of their age:

```
> db.tycs.find({}, {"age":1,"_id":0}).sort({"age":1})
{ "age" : 19 }
{ "age" : 20 }
{ "age" : 20 }
{ "age" : 21 }
{ "age" : 21 }
>
```

Let’s display the data of students in **descending order** of their age:

```
> db.tycs.find({}, {"age":1,"_id":0}).sort({"age":-1})
{ "age" : 21 }
{ "age" : 21 }
{ "age" : 20 }
{ "age" : 20 }
{ "age" : 19 }
```

13.MongoDB Indexing:

MongoDB Indexing Tutorial with Example

An **index** in MongoDB is a special data structure that holds the data of few fields of documents on which the index is created. Indexes improve the speed of search operations in database because instead of searching the whole document, the search is performed on the indexes that holds only few fields. On the other hand, having too many indexes can hamper the performance of insert, update and delete operations because of the additional write and additional data space used by indexes.

How to create index in MongoDB

Syntax:

```
db.collection_name.createIndex({field_name: 1 or -1})
```

The value 1 is for ascending order and -1 is for descending order.

For example, I have a collection studentdata. The documents inside this collection have following fields:

student_name, student_id and student_age

its say I want to create the index on student_name field in ascending order:

```
db.studentdata.createIndex({student_name: 1})
```

Output:

```
> db.tycs.createIndex({name: 3})
{
        "createdCollectionAutomatically" : false,
        "numIndexesBefore" : 1,
        "numIndexesAfter" : 2,
        "ok" : 1
}
```

We have created the index on name which means when someone searches the document based on the name, the search will be faster because the index will be used for this search. So, this is important to create the index on the field that will be frequently searched in a collection.

MongoDB – Finding the indexes in a collection

We can use getIndexes() method to find all the indexes created on a collection. The syntax for this method is:

```
db.collection_name.getIndexes()
```

```
> db.tycs.getIndexes()
[
        {
                "v" : 2,
                "key" : {
                        "_id" : 1
                },
                "name" : "_id_",
                "ns" : "tycs.tycs"
        }
]
```

The output shows that we have two indexes in this collection. The default index created on _id and the index that we have created on student_name field.

MongoDB – Drop indexes in a collection
You can either drop a particular index or all the indexes.

Dropping a specific index:
For this purpose the dropIndex() method is used.

```
db.collection_name.dropIndex({index_name: 1})
```

Lets drop the index that we have created on tycs:

```
> db.tycs.dropIndex({name: 3})
{ "nIndexesWas" : 2, "ok" : 1 }
```

It shows how many indexes were there before this command got executed,
ok: 1: This means the command is executed successfully.

Dropping all the indexes:
To drop all the indexes of a collection, we use dropIndexes() method.
Syntax of dropIndexs() method:

```
db.collection_name.dropIndexes()
```

Lets say we want to drop all the indexes of tycs collection.

```
db.tycs.dropIndexes()
```

```
> db.tycs.dropIndex({name: 3})
{ "nIndexesWas" : 2, "ok" : 1 }
> db.tycs.dropIndexes()
{
        "nIndexesWas" : 1,
        "msg" : "non-_id indexes dropped for collection",
        "ok" : 1
}
```

The message “non-_id indexes dropped for collection” indicates that the default index _id will still remain and cannot be dropped. This means that using this method we can only drop indexes that we have created, we can’t drop the default index created on _id field.