MODULE 3

6.1 WHAT IS MONGODB?

MongoDB is

1. Cross-platform.
2. Open source.
3. Non-relational.
4. Distributed.
5. NoSQL.
6. Document-oriented data store.

6.2 WHY MONGODB?

Few of the major challenges with traditional RDBMS are dealing with large volumes of data, rich variety of data – particularly unstructured data, and meeting up to the scale needs of enterprise data. The need is for a database that can scale out or scale horizontally to meet the scale requirements, has flexibility with respect to schema, is fault tolerant, is consistent and partition tolerant, and can be easily distributed over a multitude of nodes in a cluster. Refer Figure 6.1.

6.2.1 Using Java Script Object Notation (JSON)

JSON is extremely expressive. MongoDB actually does not use JSON but BSON (pronounced Bee Son) – it is Binary JSON. It is an open standard. It is used to store complex data structures.

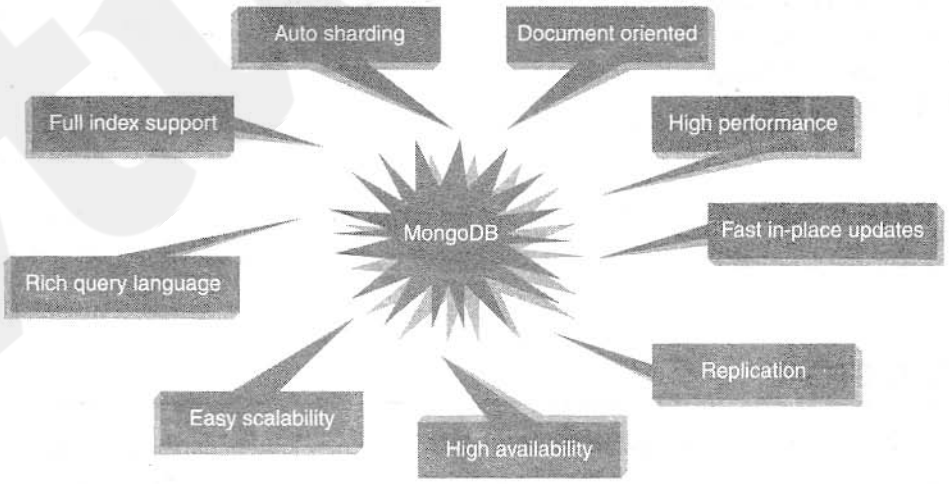


Figure 6.1 Why MongoDB?

Let us trace the journey from .csv to XML to JSON: Let us look at how data is stored in .csv file. Assume that this data is about the employees of an organization named "XYZ". As can be seen below, the column values are separated using commas and the rows are separated by a carriage return.

John, Mathews, +123 4567 8900
Andrews, Symmonds, +456 7890 1234
Mable, Mathews, +789 1234 5678

This looks good! However let us make it slightly more legible by adding column heading.

FirstName, LastName, ContactNo
John, Mathews, +123 4567 8900
Andrews, Symmonds, +456 7890 1234
Mable, Mathews, +789 1234 5678

Now assume that few employees have more than one ContactNo. It can be neatly classified as OfficeContactNo and HomeContactNo. But what if few employees have more than one OfficeContactNo and more than one HomeContactNo? Ok, so this is the first issue we need to address.

Let us look at just another piece of data that you wish to store about the employees. You need to store their email addresses as well. Here again we have the same issues, few employees have two email addresses, few have three and there are a few employees with more than three email addresses as well.

As we come across these fields or columns, we realize that it gets messy with .csv. CSV are known to store data well if it is flat and does not have repeating values.

The problem becomes even more complex when different departments maintain the details of their employees. The formats of .csv (columns, etc.) could vastly differ and it will call for some efforts before we can merge the files from the various departments to make a single file.

This problem can be solved by XML. But as the name suggests XML is highly extensible. It does not just call for defining a data format, rather it defines how you define a data format. You may be prepared to undertake this cumbersome task for highly complex and structured data; however, for simple data exchange it might just be too much work.

Enter JSON! Let us look at how it reacts to the problem at hand.

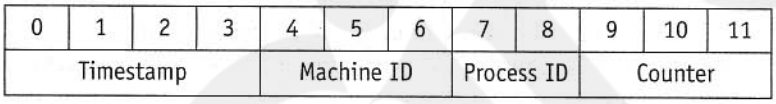
```
{
FirstName: John,
LastName: Mathews,
ContactNo: [+123 4567 8900, +123 4444 5555]
}

{
FirstName: Andrews,
LastName: Symmonds,
ContactNo: [+456 7890 1234, +456 6666 7777]
}

{
FirstName Mable,
LastName: Mathews,
ContactNo: +789 1234 5678
}
```

As you can see it is quite easy to read a JSON. There is absolutely no confusion now. One can have a list of *n* contact numbers, and they can be stored with ease.

JSON is very expressive. It provides the much needed ease to store and retrieve documents in their real form. The binary form of JSON is BSON. BSON is an open standard. In most cases it consumes less space as compared to the text-based JSON. There is yet another advantage with BSON. It is much easier and quicker to convert BSON to a programming language's native data format. There are MongoDB drivers available for a number of programming languages such as C, C++, Ruby, PHP, Python, C#, etc., and each works slightly differently. Using the basic binary format enables the native data structures to be built quickly for each language without going through the hassle of first processing JSON.

6.2.2 Creating or Generating a Unique Key

Each JSON document should have a unique identifier. It is the _id key. It is similar to the primary key in relational databases. This facilitates search for documents based on the unique identifier. An index is automatically built on the unique identifier. It is your choice to either provide unique values yourself or have the mongo shell generate the same.



6.2.2.1 Database

It is a collection of collections. In other words, it is like a container for collections. It gets created the first time that your collection makes a reference to it. This can also be created on demand. Each database gets its own set of files on the file system. A single MongoDB server can house several databases.

6.2.2.2 Collection

A collection is analogous to a table of RDBMS. A collection is created on demand. It gets created the first time that you attempt to save a document that references it. A collection exists within a single database. A collection holds several MongoDB documents. A collection does not enforce a schema. This implies that documents within a collection can have different fields. Even if the documents within a collection have same fields, the order of the fields can be different.

6.2.2.3 Document

A document is analogous to a row/record/tuple in an RDBMS table. A document has a dynamic schema. This implies that a document in a collection need not necessarily have the same set of fields/key–value pairs. Shown in Figure 6.2 is a collection by the name "students" containing three documents.

6.2.3 Support for Dynamic Queries

MongoDB has extensive support for dynamic queries. This is in keeping with traditional RDBMS wherein we have static data and dynamic queries. CouchDB, another document-oriented, schema-less NoSQL database and MongoDB's biggest competitor, works on quite the reverse philosophy. It has support for dynamic data and static queries.

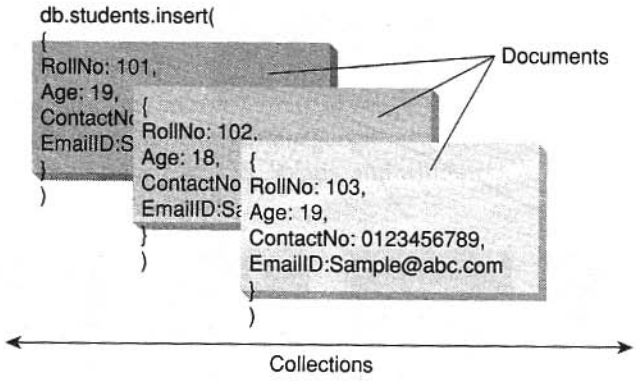


Figure 6.2 A collection "students" containing 3 documents.

6.2.4 Storing Binary Data

MongoDB provides GridFS to support the storage of binary data. It can store up to 4 MB of data. This usually suffices for photographs (such as a profile picture) or small audio clips. However, if one wishes to store movie clips, MongoDB has another solution.

It stores the metadata (data about data along with the context information) in a collection called "file". It then breaks the data into small pieces called chunks and stores it in the "chunks" collection. This process takes care about the need for easy scalability.

6.2.5 Replication

Why replication? It provides data redundancy and high availability. It helps to recover from hardware failure and service interruptions. In MongoDB, the replica set has a single primary and several secondaries. Each write request from the client is directed to the primary. The primary logs all write requests into its Oplog (operations log). The Oplog is then used by the secondary replica members to synchronize their data. This way there is strict adherence to consistency. Refer Figure 6.3. The clients usually read from the primary. However, the client can also specify a read preference that will then direct the read operations to the secondary.

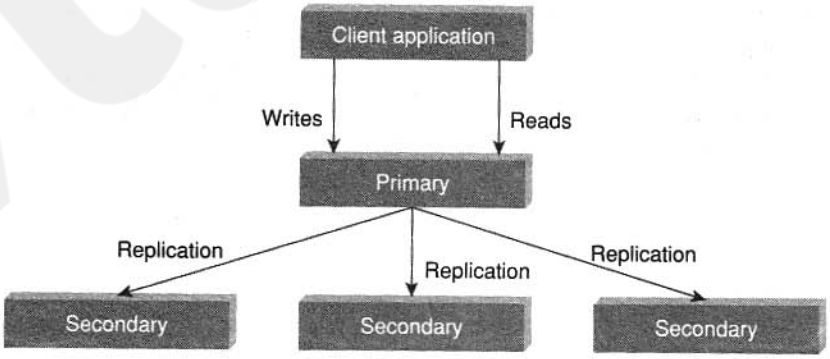


Figure 6.3 The process of **REPLICATION** in MongoDB.

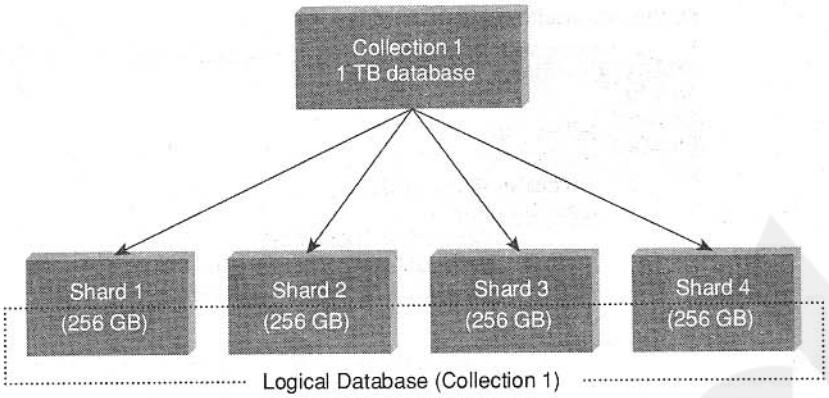


Figure 6.4 The process of **SHARDING** in MongoDB.

6.2.6 Sharding

Sharding is akin to horizontal scaling. It means that the large dataset is divided and distributed over multiple servers or shards. Each shard is an independent database and collectively they would constitute a logical database.

The prime advantages of sharding are as follows:

1. Sharding reduces the amount of data that each shard needs to store and manage. For example, if the dataset was 1 TB in size and we were to distribute this over four shards, each shard would house just 256 GB data. Refer Figure 6.4. As the cluster grows, the amount of data that each shard will store and manage will decrease.
2. Sharding reduces the number of operations that each shard handles. For example, if we were to insert data, the application needs to access only that shard which houses that data.

6.2.7 Updating Information In-Place

MongoDB updates the information in-place. This implies that it updates the data wherever it is available. It does not allocate separate space and the indexes remain unaltered.

MongoDB is all for lazy-writes. It writes to the disk once every second. Reading and writing to disk is a slow operation as compared to reading and writing from memory. The fewer the reads and writes that we perform to the disk, the better is the performance. This makes MongoDB faster than its other competitors who write almost immediately to the disk. However, there is a tradeoff. MongoDB makes no guarantee that data will be stored safely on the disk.

Guess Me

A. Who am I?

- I am blindingly fast
- I am massively scalable
- I am easy to use
- I work with documents rather than rows

B. Who am I?

- I am not for everyone
- I am good with complex data structures such as blog posts and comments
- I am good with analytics such as a real time google analytics
- I am comfortable with Linux, Mac OS, Solaris, and windows

C. Who am I?

- I have support for transactions
- I have static data
- I allow dynamic queries to be run on me

D. Who am I?

- I am one of the biggest competitor for MongoDB
- I have dynamic data
- Only static queries can be run on me
- I am document-oriented too

Answers:

A. MongoDB
B. MongoDB
C. Traditional RDBMS
D. CouchDB

6.3 TERMS USED IN RDBMS AND MONGODB

RDBMS	MongoDB
Database	Database
Table	Collection
Record	Document
Columns	Fields / Key Value pairs
Index	Index
Joins	Embedded documents
Primary Key	Primary key (_id is a identifier)

	MySQL	Oracle	MongoDB
Database Server	MySqld	Oracle	Mongod
Database Client	MySql	SQL Plus	mongo

6.3.1 Create Database

The syntax for creating database is as follows:

use DATABASE_Name

To create a database by the name "myDB" the syntax is

use myDB

```
> use myDB;
switched to db myDB
>
```

To confirm the existence of your database, type the command at the MongoDB shell:

db

```
> db;
myDB
>
```

To get a list of all databases, type the below command:

show dbs

```
> show dbs;
admin  (empty)
local  0.078GB
test   0.078GB
>
```

Notice that the newly created database, "myDB" does not show up in the list above. The reason is that the database needs to have at least one document to show up in the list.

The default database in MongoDB is test. If one does not create any database, all collections are by default stored in the test database.

6.3.2 Drop Database

The syntax to drop database is as follows:

db.dropDatabase();

To drop the database, "myDB", first ensure that you are currently placed in "myDB" database and then use the db.dropDatabase() command to drop the database.

use myDB;
db.dropDatabase();

Confirm if the database "myDB" has been dropped.

```
> db.dropDatabase();
{ "dropped" : "myDB", "ok" : 1 }
>
```

If no database is selected, the default database "test" is dropped.

6.4 DATA TYPES IN MONGODB

The following are various data types in MongoDB.

String	Must be UTF-8 valid. Most commonly used data type.
Integer	Can be 32-bit or 64-bit (depends on the server).
Boolean	To store a true/false value.
Double	To store floating point (real values).
Min/Max keys	To compare a value against the lowest or highest BSON elements.
Arrays	To store arrays or list or multiple values into one key.
Timestamp	To record when a document has been modified or added.
Null	To store a NULL value. A NULL is a missing or unknown value.
Date	To store the current date or time in Unix time format. One can create object of date and pass day, month and year to it.
Object ID	To store the document's id.
Binary data	To store binary data (images, binaries, etc.).
Code	To store javascript code into the document.
Regular expression	To store regular expression.

A few commands worth looking at are as follows (try them!!!).

To report the name of the current database:

```
C:\windows\system32\cmd.exe - mongo
> db
test
>
```

To display the list of databases:

```
C:\windows\system32\cmd.exe - mongo
> show dbs
admin  (empty)
local  0.078GB
myDB1  0.078GB
>
```

To switch to a new database, for example, myDB1:

```
C:\windows\system32\cmd.exe - mongo
> use myDB1
switched to db myDB1
>
```

To display the list of collections (tables) in the current database:

```
C:\windows\system32\cmd.exe - mongo
> show collections
system.indexes
system.js
>
```

To display the current version of the MongoDB server:

```
C:\windows\system32\cmd.exe - mongo
> db.version()
2.6.1
>
```

To display the statistics that reflect the use state of a database:

```
C:\windows\system32\cmd.exe - mongo
> db.stats()
{
        "db" : "myDB1",
        "collections" : 3,
        "objects" : 6,
        "avgObjSize" : 122.66666666666667,
        "dataSize" : 736,
        "storageSize" : 24576,
        "numExtents" : 3,
        "indexes" : 1,
        "indexSize" : 8176,
        "fileSize" : 67108864,
        "nsSizeMB" : 16,
        "dataFileVersion" : {
                "major" : 4,
                "minor" : 5
        },
        "extentFreeList" : {
                "num" : 14,
                "totalSize" : 974848
        },
        "ok" : 1
}
>
```

Type in db.help() in the MongoDB client to get a list of commands:

```
C:\windows\system32\cmd.exe - mongo
> db.help();
DB methods:
        db.adminCommand(nameOrDocument) - switches to 'admin' db, and runs comma
nd [ just calls db.runCommand(...) ]
        db.auth(username, password)
        db.cloneDatabase(fromhost)
        db.commandHelp(name) returns the help for the command
        db.copyDatabase(fromdb, todb, fromhost)
        db.createCollection(name, { size : ..., capped : ..., max : ... } )
        db.createUser(userDocument)
        db.currentOp() displays currently executing operations in the db
        db.dropDatabase()
        db.eval(func, args) run code server-side
        db.fsyncLock() flush data to disk and lock server for backups
        db.fsyncUnlock() unlocks server following a db.fsyncLock()
        db.getCollection(cname) same as db['cname'] or db.cname
        db.getCollectionNames()
        db.getLastError() - just returns the err msg string
        db.getLastErrorObj() - return full status object
        db.getMongo() get the server connection object
        db.getMongo().setSlaveOk() allow queries on a replication slave server
        db.getName()
        db.getPrevError()
        db.getProfilingLevel() - deprecated
        db.getProfilingStatus() - returns if profiling is on and slow threshold
        db.getReplicationInfo()
        db.getSiblingDB(name) get the db at the same server as this one
        db.getWriteConcern() - returns the write concern used for any operations
 on this db, inherited from server object if set
        db.hostInfo() get details about the server's host
        db.isMaster() check replica primary status
        db.killOp(opid) kills the current operation in the db
        db.listCommands() lists all the db commands
        db.loadServerScripts() loads all the scripts in db.system.js
        db.logout()
        db.printCollectionStats()
        db.printReplicationInfo()
        db.printShardingStatus()
        db.printSlaveReplicationInfo()
        db.dropUser(username)
        db.repairDatabase()
        db.resetError()
        db.runCommand(cmdObj) run a database command.  if cmdObj is a string, tu
rns it into { cmdObj : 1 }
        db.serverStatus()
        db.setProfilingLevel(level,<slowms>) 0=off 1=slow 2=all
        db.setWriteConcern( <write concern doc> ) - sets the write concern for w
rites to the db
        db.unsetWriteConcern( <write concern doc> ) - unsets the write concern f
or writes to the db
        db.setVerboseShell(flag) display extra information in shell output
        db.shutdownServer()
        db.stats()
        db.version() current version of the server
>
```

Consider a table "Students" with the following columns:

1. StudRollNo
2. StudName
3. Grade
4. Hobbies
5. DOJ

Before we get into the details of CRUD operations in MongoDB, let us look at how the statements are written in RDBMS and MongoDB.

	RDBMS	MongoDB
Insert	Insert into Students (StudRollNo, StudName, Grade, Hobbies, DOJ) Values ('S101', 'Simon David', 'VII', 'Net Surfing', '10-Oct-2012')	db.Students.insert({_id:1, StudRollNo: 'S101', StudName: 'Simon David', Grade: 'VII', Hobbies: 'Net Surfing', DOJ: '10-Oct-2012'});
Update	Update Students set Hobbies ='Ice Hockey' where StudRollNo ='S101'	db.Students.update({StudRollNo: 'S101'}, {$set: {Hobbies : 'Ice Hockey'}})
	Update Students Set Hobbies ='Ice Hockey'	db.Students.update({},{$set: {Hobbies: 'Ice Hockey' }}, {multi:true})
Delete	Delete from Students where StudRollNo = 'S101'	db.Students.remove ({StudRollNo : 'S101'})
	Delete From Students	db.Students.remove({})
Select	Select * from Students	db.Students.find() db.Students.find().pretty()
	Select * from students where StudRollNo = 'S101'	db.Students.find({StudRollNo: 'S101'})
	Select StudRollNo, StudName, Hobbies from Students	db.Students.find({},{StudRollNo:1, StudName:1, Hobbies:1, _id:0})
	Select StudRollNo, StudName, Hobbies from Students where StudRollNo = 'S101'	db.Students.find({StudRollNo: 'S101'}, {StudRollNo : 1, StudName: 1, Hobbies : 1, _id:0})

RDBMS	MongoDB
Select StudRollNo, StudName, Hobbies From Students Where Grade ='VII' and Hobbies ='Ice Hockey'	db.Students.find({Grade: 'VII' , Hobbies: 'Ice Hockey'}, {StudRollNo : 1, StudName: 1, Hobbies : 1, _id:0})
Select StudRollNo, StudName, Hobbies From Students Where Grade ='VII' or Hobbies = 'Ice Hockey'	db.Students.find({ $or: [{Grade: 'VII' , Hobbies: 'Ice Hockey'}], StudRollNo : 1, StudName: 1, Hobbies : 1, _id:0})
Select * From Students Where StudName like 'S%'	db.Students.find({ StudName: /^S/}).pretty()

6.5 MONGODB QUERY LANGUAGE

CRUD (Create, Read Update, and Delete) operations in MongoDB

Create → Creation of data is done using insert() or update() or save() method.
Read → Reading the data is performed using the find() method.
Update → Update to data is accomplished using the update() method with UPSERT set to false.
Delete → a document is Deleted using the remove() method.

We will present the various methods available in MongoDB shell to deal with data in the next few sections. The sections have been designed as follows:

Objective: What is it that we are trying to achieve here?
Input: What is the input that has been given to us to act upon?
Act: The actual statement/command to accomplish the task at hand.
Outcome: The result/output as a consequence of executing the statement.

At few places we have also provided the equivalent in RDBMS such as Oracle.

Objective: To create a collection by the name "Person". Let us take a look at the collection list prior to the creation of the new collection "Person":

```
C:\windows\system32\cmd.exe - mongo
> show collections
Students
food
system.indexes
system.js
>
```

Act: The statement to create the collection is
db.createCollection("Person")

```
C:\windows\system32\cmd.exe - mongo
> db.createCollection("Person");
{ "ok" : 1 }
>
```

Outcome: Below is the collection list after the creation of the new collection "Person":

```
C:\windows\system32\cmd.exe - mongo
> show collections;
Person
Students
food
system.indexes
system.js
>
```

Objective: To drop a collection by the name "food".
Take a look at the current collection list:

```
C:\windows\system32\cmd.exe - mongo
> show collections;
Person
Students
food
system.indexes
system.js
>
```

Act: The statement to drop the collection is

db.food.drop();

```
C:\windows\system32\cmd.exe - mongo
> db.food.drop();
true
>
```

Outcome: The collection list after the execution of the statement is as follows:

```
C:\windows\system32\cmd.exe - mongo
> show collections
Person
Students
system.indexes
system.js
>
```

6.5.1 Insert Method

We now explain the syntax of insert method.

```
db.students.insert(          <-------  Collection
{
RollNo: 101,                 <-------  Field: value
Age: 19,                     <-------  Field: value
ContactNo:0123456789,        <-------  Field: value
EmailID:Sample@abc.com       <-------  Field: value
}
)
```

Objective: To create a collection by the name "Students" and insert documents.

Input: Check the list of existing collections.

```
C:\windows\system32\cmd.exe - mongo
> show collections
system.indexes
system.js
>
```

Act: Create a collection by the name "Students" and store the following data in it.

db.Students.insert({_id:1, StudName:"Michelle Jacintha", Grade: "VII", Hobbies: "Internet Surfing"});

```
C:\windows\system32\cmd.exe - mongo
> db.Students.insert({_id:1, StudName:"Michelle Jacintha", Grade: "VII", Hobbies: "Internet Surfing"});
WriteResult({ "nInserted" : 1 })
>
```

Outcome: Check if the collection has been successfully created.

```
C:\windows\system32\cmd.exe - mongo
> show collections
Students
system.indexes
system.js
>
```

Check if the document for Student "Michelle Jacintha" has been successfully inserted into the "Students" collection.

```
C:\windows\system32\cmd.exe - mongo
> db.Students.find();
{ "_id" : 1, "StudName" : "Michelle Jacintha", "Grade" : "VII", "Hobbies" : "Internet Surfing" }
>
```

To format the result, one can add the pretty() method to the operation.

```
C:\windows\system32\cmd.exe - mongo
> db.Students.find().pretty();
{
        "_id" : 1,
        "StudName" : "Michelle Jacintha",
        "Grade" : "VII",
        "Hobbies" : "Internet Surfing"
}
>
```

Objective: Insert another document into the collection.

Input: Check the documents in the "Students" collection before proceeding.

```
C:\windows\system32\cmd.exe - mongo
> db.Students.find().pretty();
{
        "_id" : 1,
        "StudName" : "Michelle Jacintha",
        "Grade" : "VII",
        "Hobbies" : "Internet Surfing"
}
>
```

Act:

db.Students.insert({_id:2, StudName:"Mabel Mathews", Grade: "VII", Hobbies: "Baseball"});

```
C:\windows\system32\cmd.exe - mongo
> db.Students.insert({_id:2, StudName:"Mabel Mathews", Grade: "VII", Hobbies: "Baseball"});
WriteResult({ "nInserted" : 1 })
>
```

Outcome:

```
C:\windows\system32\cmd.exe - mongo
> db.Students.find().pretty();
{
        "_id" : 1,
        "StudName" : "Michelle Jacintha",
        "Grade" : "VII",
        "Hobbies" : "Internet Surfing"
}
{
        "_id" : 2,
        "StudName" : "Mabel Mathews",
        "Grade" : "VII",
        "Hobbies" : "Baseball"
}
>
```

Objective: Insert the document for "Aryan David" into the Students collection only if it does not already exist in the collection. However, if it is already present in the collection, then update the document with new values. (Update his Hobbies from "Skating" to "Chess".) Use "**Update else insert**" (if there is an existing document, it will attempt to update it, if there is no existing document then it will insert it).

Input: Check the documents in the "Students" collection before proceeding.

```
C:\windows\system32\cmd.exe - mongo
> db.Students.find().pretty();
{
        "_id" : 1,
        "StudName" : "Michelle Jacintha",
        "Grade" : "VII",
        "Hobbies" : "Internet Surfing"
}
{
        "_id" : 2,
        "StudName" : "Mabel Mathews",
        "Grade" : "VII",
        "Hobbies" : "Baseball"
}
>
```

Act:

db.Students.update({_id:3, StudName:"Aryan David", Grade: "VII"},{$set:{Hobbies: "Skating"}},{upsert:true});

```
C:\windows\system32\cmd.exe - mongo
> db.Students.update({_id:3, StudName:"Aryan David", Grade: "VII"},{$set:{Hobbies: "Skating"}},{upsert:true});
WriteResult({ "nMatched" : 0, "nUpserted" : 1, "nModified" : 0, "_id" : 3 })
>
```

Outcome: Confirm the presence of the document of "Aryan David" in the "Students" collection.

```
C:\windows\system32\cmd.exe - mongo
> db.Students.find({_id:3});
{ "_id" : 3, "Grade" : "VII", "StudName" : "Aryan David", "Hobbies" : "Skating" }
>
```

Objective: Insert the document for "Aryan David" into the Students collection only if it does not already exist in the collection. However, if it is already present in the collection, then update the document with new values. (Update his Hobbies from "Skating" to "Chess"). Use "**Update else insert**" (if there is an existing document, it will attempt to update it, if there is no existing document then it will insert it). Try the UPSERT operator by setting it first to "true" and then to "false" and observe the output.

Input: Check the documents in the "Students" collection before proceeding.

```
C:\windows\system32\cmd.exe - mongo
> db.Students.find().pretty();
{
        "_id" : 1,
        "StudName" : "Michelle Jacintha",
        "Grade" : "VII",
        "Hobbies" : "Internet Surfing"
}
{
        "_id" : 2,
        "StudName" : "Mabel Mathews",
        "Grade" : "VII",
        "Hobbies" : "Baseball"
}
{
        "_id" : 3,
        "Grade" : "VII",
        "StudName" : "Aryan David",
        "Hobbies" : "Skating"
}
>
```

Act:

```
db.Students.update({_id:3, StudName:"Aryan David", Grade: "VII"},{$set:{Hobbies: "Chess"}},{upsert:true});
```

```
C:\windows\system32\cmd.exe - mongo
> db.Students.update({_id:3, StudName:"Aryan David", Grade: "VII"},{$set:{Hobbies: "Chess"}},{upsert:true});
WriteResult({ "nMatched" : 1, "nUpserted" : 0, "nModified" : 1 })
>
```

Outcome: Confirm that the required changes have been made to the document of "Aryan David" in the "Students" collection.

```
C:\windows\system32\cmd.exe - mongo
> db.Students.find({_id:3});
{ "_id" : 3, "Grade" : "VII", "StudName" : "Aryan David", "Hobbies" : "Chess" }
>
```

Objective: To demonstrate Save method to insert a document for student "Vamsi Bapat" in the "Students" collection. Omit providing value for the _id key.

Input: Check the documents in the "Students" collection before proceeding.

```
C:\windows\system32\cmd.exe - mongo
> db.Students.find().pretty();
{
        "_id" : 1,
        "StudName" : "Michelle Jacintha",
        "Grade" : "VII",
        "Hobbies" : "Internet Surfing"
}
{
        "_id" : 2,
        "StudName" : "Mabel Mathews",
        "Grade" : "VII",
        "Hobbies" : "Baseball"
}
{
        "_id" : 3,
        "Grade" : "VII",
        "StudName" : "Aryan David",
        "Hobbies" : "Chess"
}
>
```

Act:

db.Students.save({StudName:"Vamsi Bapat",Grade:"VI"})

```
C:\windows\system32\cmd.exe - mongo
> db.Students.save({StudName:"Vamsi Bapat",Grade:"VI"})
WriteResult({ "nInserted" : 1 })
>
```

Outcome: Confirm the presence of the document of "Vamsi Bapat" in the "Students" collection.

```
C:\windows\system32\cmd.exe - mongo
> db.Students.find().pretty();
{
        "_id" : 1,
        "StudName" : "Michelle Jacintha",
        "Grade" : "VII",
        "Hobbies" : "Internet Surfing"
}
{
        "_id" : 2,
        "StudName" : "Mabel Mathews",
        "Grade" : "VII",
        "Hobbies" : "Baseball"
}
{
        "_id" : 3,
        "Grade" : "VII",
        "StudName" : "Aryan David",
        "Hobbies" : "Chess"
}
{
        "_id" : ObjectId("546dd0e0a7fba710799bb94d"),
        "StudName" : "Vamsi Bapat",
        "Grade" : "VI"
}
>
```

6.5.2 Save() Method

We now explain the save() method. The save() method will insert a new document if the document with the specified _id does not exist. However, if a document with the specified id exists, it replaces the existing document with the new one.

Objective: Insert the document of "Hersch Gibbs" into the Students collection using the Update method. First try with upsert set to false and then with upsert set to true.

Input: Check the documents in the "Students" collection before proceeding.

```
C:\windows\system32\cmd.exe - mongo
> db.Students.find();
{ "_id" : 1, "StudName" : "Michelle Jacintha", "Grade" : "VII", "Hobbies" : "Internet Surfing" }
{ "_id" : 2, "StudName" : "Mabel Mathews", "Grade" : "VII", "Hobbies" : "Baseball" }
{ "_id" : 3, "Grade" : "VII", "StudName" : "Aryan David", "Hobbies" : "Chess" }
{ "_id" : ObjectId("546dd0e0a7fba710799bb94d"), "StudName" : "Vamsi Bapat", "Grade" : "VI" }
>
```

Act: *Update method with upsert set to false.*

db.Students.update({_id:4, StudName:"Hersch Gibbs", Grade: "VII"},{$set:{Hobbies: "Graffiti"}}, {upsert:false});

```
C:\windows\system32\cmd.exe - mongo
> db.Students.update({_id:4, StudName:"Hersch Gibbs", Grade: "VII"},{$set:{Hobbies: "Graffiti"}}, {upsert:false});
WriteResult({ "nMatched" : 0, "nUpserted" : 0, "nModified" : 0 })
>
```

As evident from the above display (nUpserted : 0), no document has been inserted because upsert is set to false.

Update method with upsert set to true.

db.Students.update({_id:4, StudName:"Hersch Gibbs", Grade: "VII"},{$set:{Hobbies: "Graffiti"}}, {upsert:true});

```
C:\windows\system32\cmd.exe - mongo
> db.Students.update({_id:4, StudName:"Hersch Gibbs", Grade: "VII"},{$set:{Hobbies: "Graffiti"}}, {upsert:true});
WriteResult({ "nMatched" : 0, "nUpserted" : 1, "nModified" : 0, "_id" : 4 })
>
```

nUpserted: 1 implies that a document with _id:4 has been inserted.

Outcome: Confirm the presence of the document of "Hersch Gibbs" in the "Students" collection.

```
C:\windows\system32\cmd.exe - mongo
> db.Students.find();
{ "_id" : 1, "StudName" : "Michelle Jacintha", "Grade" : "VII", "Hobbies" : "Internet Surfing" }
{ "_id" : 2, "StudName" : "Mabel Mathews", "Grade" : "VII", "Hobbies" : "Baseball" }
{ "_id" : 3, "Grade" : "VII", "StudName" : "Aryan David", "Hobbies" : "Chess" }
{ "_id" : ObjectId("546dd0e0a7fba710799bb94d"), "StudName" : "Vamsi Bapat", "Grade" : "VI" }
{ "_id" : 4, "Grade" : "VII", "StudName" : "Hersch Gibbs", "Hobbies" : "Graffiti" }
>
```

6.5.3 Adding a New Field to an Existing Document – Update Method

We now discuss the syntax of update method.

```
db.students.update(        <-------  Collection
{Age: {$gt 18}},           <-------  Update Criteria
{$set: {Status: "A" }}     <-------  Update Action
{multi:true}               <-------  Update Option
)
```

Objective: To add a new field "Location" with value "Newark" to the document (_id:4) of "Students" collection.

Input: Check the document (_id:4) in the "Students" collection before proceeding.

```
C:\windows\system32\cmd.exe - mongo
> db.Students.find({_id:4}).pretty();
{
        "_id" : 4,
        "Grade" : "VII",
        "StudName" : "Hersch Gibbs",
        "Hobbies" : "Graffiti"
}
>
```

Act:

db.Students.update({_id:4},{$set:{Location:"Newark"}});

```
C:\windows\system32\cmd.exe - mongo
> db.Students.update({_id:4},{$set:{Location:"Newark"}});
WriteResult({ "nMatched" : 1, "nUpserted" : 0, "nModified" : 1 })
>
```

Outcome: Confirm that the new field "Location" with value "Newark" has been added to document (_id:4) in the "Students" collection.

```
C:\windows\system32\cmd.exe - mongo
> db.Students.find({_id:4}).pretty();
{
        "_id" : 4,
        "Grade" : "VII",
        "StudName" : "Hersch Gibbs",
        "Hobbies" : "Graffiti",
        "Location" : "Newark"
}
>
```

6.5.4 Removing an Existing Field from an Existing Document – Remove Method

In this section we will explain the syntax of remove method.

```
db.students.remove(      <-------- Collection
{Age: {$gt 18}},         <-------- Remove Criteria
)
```

Objective: To remove the field "Location" with value "Newark" in the document (_id:4) of "Students" collection.

Input: Check the document (_id:4) in the "Students" collection before proceeding.

```
C:\windows\system32\cmd.exe - mongo
> db.Students.find({_id:4}).pretty();
{
        "_id" : 4,
        "Grade" : "VII",
        "StudName" : "Hersch Gibbs",
        "Hobbies" : "Graffiti",
        "Location" : "Newark"
}
>
```

Act:

db.Students.update({_id:4},{$unset:{Location:"Newark"}});

```
C:\windows\system32\cmd.exe - mongo
> db.Students.update({_id:4},{$unset:{Location:"Newark"}});
WriteResult({ "nMatched" : 1, "nUpserted" : 0, "nModified" : 1 })
>
```

Outcome: Confirm if the stated field ("Location") has been dropped from the document (_id:4) of the "Students" collection.

```
C:\windows\system32\cmd.exe - mongo
> db.Students.find({_id:4}).pretty();
{
        "_id" : 4,
        "Grade" : "VII",
        "StudName" : "Hersch Gibbs",
        "Hobbies" : "Graffiti"
}
>
```

6.5.5 Finding Documents based on Search Criteria – Find Method

The syntax of find method is as follows:

```
db.students.find(          <-------- Collection
{Age: {$gt 18}},           <-------- Selection Criteria
{RollNo:1,Age:1,_id:1}     <-------- Projection
).limit(10)                <-------- Cursor Modifier
```

Objective: To search for documents from the "Students" collection based on certain search criteria.

Input: Check the documents in the "Students" collection before proceeding.

```
C:\windows\system32\cmd.exe - mongo
> db.Students.find({}).pretty();
{
        "_id" : 1,
        "StudName" : "Michelle Jacintha",
        "Grade" : "VII",
        "Hobbies" : "Internet Surfing"
}
{
        "_id" : 3,
        "Grade" : "VII",
        "StudName" : "Aryan David",
        "Hobbies" : "Chess"
}
{
        "_id" : 4,
        "Grade" : "VII",
        "StudName" : "Hersch Gibbs",
        "Hobbies" : "Graffiti"
}
{
        "_id" : ObjectId("5464849889ad1ab07d489b7f"),
        "StudName" : "Vamsi Bapat",
        "Grade" : "VI"
}
{
        "_id" : 2,
        "StudName" : "Mabel Mathews",
        "Grade" : "VII",
        "Hobbies" : "Baseball"
}
>
```

Act: Find the document wherein the "StudName" has value "Aryan David".

db.Students.find({StudName:"Aryan David"});

Outcome:

```
C:\windows\system32\cmd.exe - mongo
> db.Students.find({StudName:"Aryan David"});
{ "_id" : 3, "Grade" : "VII", "StudName" : "Aryan David", "Hobbies" : "Chess" }
>
```

To format the above output, use the pretty() method:

db.Students.find({StudName:"Aryan David"}).pretty();

```
C:\windows\system32\cmd.exe - mongo
> db.Students.find({StudName:"Aryan David"}).pretty();
{
        "_id" : 3,
        "Grade" : "VII",
        "StudName" : "Aryan David",
        "Hobbies" : "Chess"
}
>
```

RDBMS equivalent:

```
Select *
    From Students
        Where StudName like 'Aryan David';
```

```
SQL> select * from Students where StudName like 'Aryan David';

STUDR STUDNAME                    GRADE HOBBIES
----- --------------------------- ----- ------------------------------
3     Aryan David                 VII   Chess

SQL>
```

Objective: To display only the StudName from all the documents of the Student's collection. The identifier "__id" should be suppressed and NOT displayed.

Act:

db.Students.find({},{StudName:1,_id:0});

Outcome:

```
C:\windows\system32\cmd.exe - mongo
> db.Students.find({},{StudName:1,_id:0});
{ "StudName" : "Michelle Jacintha" }
{ "StudName" : "Aryan David" }
{ "StudName" : "Hersch Gibbs" }
{ "StudName" : "Vamsi Bapat" }
{ "StudName" : "Mabel Mathews" }
>
```

RDBMS equivalent:

Select StudName
From Students;

```
SQL Plus
SQL> select StudName from Students;

STUDNAME
-------------------------
Michelle Jacintha
Aryan David
Mabel Mathews
Hersch Gibbs
Vamsi Bapat

SQL>
```

Objective: To display only the StudName and Grade from all the documents of the Students collection. The identifier _id should be suppressed and NOT displayed.

Act:

db.Students.find({},{StudName:1,Grade:1,_id:0});

Outcome:

```
> db.Students.find({},{StudName:1,Grade:1,_id:0});
{ "StudName" : "Michelle Jacintha", "Grade" : "VII" }
{ "Grade" : "VII", "StudName" : "Aryan David" }
{ "Grade" : "VII", "StudName" : "Hersch Gibbs" }
{ "StudName" : "Vamsi Bapat", "Grade" : "VI" }
{ "StudName" : "Mabel Mathews", "Grade" : "VII" }
>
```

RDBMS equivalent:

Select StudName, Grade
From Students;

```
SQL Plus
SQL> select StudName, Grade from Students;

STUDNAME                  GRADE
------------------------- -----
Michelle Jacintha         VII
Aryan David               VII
Mabel Mathews             VII
Hersch Gibbs              VII
Vamsi Bapat               VI

SQL>
```

Objective: To display the StudName, Grade as well the identifier, _id from the document of the Students collection where the _id column is 1.

Act:

db.Students.find({_id:1},{StudName:1,Grade:1});

Outcome:

```
C:\windows\system32\cmd.exe - mongo
> db.Students.find({_id:1},{StudName:1,Grade:1});
{ "_id" : 1, "StudName" : "Michelle Jacintha", "Grade" : "VII" }
>
```

RDBMS equivalent:

Select StudRollNo, StudName, Grade
From Students
Where StudRollNo = '1';

```
SQL> select StudRollNo, StudName, Grade from Students where StudRollNo = '1';
STUDR STUDNAME                    GRADE
----- --------------------------- -----
1     Michelle Jacintha           VII
SQL>
```

Objective: To display the StudName and Grade from the document of the Students collection where the _id column is 1. The _id field should NOT be displayed.

Act:

db.Students.find({_id:1},{StudName:1,Grade:1,_id:0});

Outcome:

```
C:\windows\system32\cmd.exe - mongo
> db.Students.find({_id:1},{StudName:1,Grade:1,_id:0});
{ "StudName" : "Michelle Jacintha", "Grade" : "VII" }
>
```

RDBMS equivalent:

Select StudName, Grade
From Students
Where StudRollNo like '1';

```
SQL Plus
SQL> select StudName, Grade from Students where StudRollNo like '1';
STUDNAME                  GRADE
------------------------- -----
Michelle Jacintha         VII
SQL>
```

Relational operators available to use in the search criteria:

$eq	→ equal to
$ne	→ not equal to
$gte	→ greater than or equal to
$lte	→ less than or equal to
$gt	→ greater than
$lt	→ less than

Objective: To find those documents where the Grade is set to 'VII'.

Act:

db.Students.find({Grade:{$eq:'VII'}}).pretty();

Outcome:

```
C:\windows\system32\cmd.exe - mongo
> db.Students.find({Grade:{$eq:'VII'}}).pretty();
{
        "_id" : 1,
        "StudName" : "Michelle Jacintha",
        "Grade" : "VII",
        "Hobbies" : "Internet Surfing"
}
{
        "_id" : 3,
        "Grade" : "VII",
        "StudName" : "Aryan David",
        "Hobbies" : "Chess"
}
{
        "_id" : 4,
        "Grade" : "VII",
        "StudName" : "Hersch Gibbs",
        "Hobbies" : "Graffiti"
}
{
        "_id" : 2,
        "StudName" : "Mabel Mathews",
        "Grade" : "VII",
        "Hobbies" : "Baseball"
}
>
```

RDBMS Equivalent:

Select *

From Students

Where Grade like 'VII';

```
SQL Plus
SQL> select * from Students where Grade like 'VII';

STUDR STUDNAME                    GRADE HOBBIES
----- --------------------------- ----- --------------------------------
1     Michelle Jacintha           VII   Internet surfing
3     Aryan David                 VII   Chess
2     Mabel Mathews               VII   Baseball
4     Hersch Gibbs                VII   Graffiti

SQL>
```

Objective: To find those documents where the Grade is NOT set to 'VII'.

Act:

db.Students.find({Grade:{$ne:'VII'}}).pretty();

Outcome:

```
C:\windows\system32\cmd.exe - mongo
> db.Students.find({Grade:{$ne:'VII'}}).pretty();
{
        "_id" : ObjectId("5464849889ad1ab07d489b7f"),
        "StudName" : "Vamsi Bapat",
        "Grade" : "VI"
}
>
```

There is just one document that meets the above criteria of Grade NOT EQUAL to 'VII'.

RDBMS Equivalent:

Select *

From Students

Where Grade <> 'VII';

```
SQL Plus
SQL> select * from Students where Grade <> 'VII';

STUDR STUDNAME                    GRADE HOBBIES
----- --------------------------- ----- --------------------------------
      Vamsi Bapat                 VI

SQL>
```

OR

```
SQL Plus
SQL> select * from Students where Grade != 'VII';
STUDR STUDNAME                    GRADE HOBBIES
----- --------------------------- ----- ------------------------------
      Vamsi Bapat                 VI
SQL>
```

Objective: To find those documents from the Students collection where the Hobbies is set to either 'Chess' or is set to 'Skating'.

Act:

db.Students.find ({Hobbies :{ $in: ['Chess','Skating']}}).pretty ();

Outcome:

```
C:\windows\system32\cmd.exe - mongo
> db.Students.find ({Hobbies :{ $in: ['Chess','Skating']}}).pretty ();
{
        "_id" : 3,
        "Grade" : "VII",
        "StudName" : "Aryan David",
        "Hobbies" : "Chess"
}
>
```

RDBMS Equivalent:

Select *

From Students

Where Hobbies in ('Chess', 'Skating');

```
SQL Plus
SQL> select * from Students where Hobbies in ('Chess', 'Skating');
STUDR STUDNAME                    GRADE HOBBIES
----- --------------------------- ----- ------------------------------
3     Aryan David                 VII   Chess
SQL>
```

Objective: To find those documents from the Students collection where the Hobbies is set neither to 'Chess' nor is set to 'Skating'.

Act:

db.Students.find ({Hobbies :{ $nin: ['Chess','Skating']}}).pretty ();

Outcome:

```
C:\windows\system32\cmd.exe - mongo
> db.Students.find ({Hobbies :{ $nin: ['Chess','Skating']}}).pretty ();
{
        "_id" : 1,
        "StudName" : "Michelle Jacintha",
        "Grade" : "VII",
        "Hobbies" : "Internet Surfing"
}
{
        "_id" : 4,
        "Grade" : "VII",
        "StudName" : "Hersch Gibbs",
        "Hobbies" : "Graffiti"
}
{
        "_id" : ObjectId("5464849889ad1ab07d489b7f"),
        "StudName" : "Vamsi Bapat",
        "Grade" : "VI"
}
{
        "_id" : 2,
        "StudName" : "Mabel Mathews",
        "Grade" : "VII",
        "Hobbies" : "Baseball"
}
>
```

RDBMS Equivalent:

```
Select *
        From Students
                Where Hobbies not in ('Chess', 'Skating');
```

```
SQL> select * from Students where Hobbies not in ('Chess','Skating');

STUDR STUDNAME                    GRADE HOBBIES
----- --------------------------- ----- ------------------------------
1     Michelle Jacintha           VII   Internet surfing
2     Mabel Mathews               VII   Baseball
4     Hersch Gibbs                VII   Graffiti

SQL>
```

Objective: To find those documents from the Students collection where the Hobbies is set to 'Graffiti' and the StudName is set to 'Hersch Gibbs' (AND condition).

Act:

```
db.Students.find({Hobbies:'Graffiti', StudName: 'Hersch Gibbs'}).pretty();
```

Outcome:

```
> db.Students.find({Hobbies:'Graffiti', StudName: 'Hersch Gibbs'}).pretty();
{
        "_id" : 4,
        "Grade" : "VII",
        "StudName" : "Hersch Gibbs",
        "Hobbies" : "Graffiti"
}
>
```

RDBMS Equivalent:

```
Select *
        From Students
                Where Hobbies like 'Graffiti' and StudName like 'Hersch Gibbs';
```

```
SQL> select * from Students where Hobbies like 'Graffiti' and StudName like 'Her
sch Gibbs';

STUDR STUDNAME                    GRADE HOBBIES
----- --------------------------- ----- ------------------------------
4     Hersch Gibbs                VII   Graffiti

SQL>
```

Objective: To find documents from the Students collection where the StudName begins with "M".

Act:

```
db.Students.find({StudName:/^M/}).pretty();
```

Outcome:

```
> db.Students.find({StudName:/^M/}).pretty();
{
        "_id" : 1,
        "StudName" : "Michelle Jacintha",
        "Grade" : "VII",
        "Hobbies" : "Internet Surfing"
}
{
        "_id" : 2,
        "StudName" : "Mabel Mathews",
        "Grade" : "VII",
        "Hobbies" : "Baseball"
}
>
```

RDBMS Equivalent:

```
Select *
        From Students
                Where StudName like 'M%';
```

```
SQL> select * from Students where StudName like 'M%';
STUDR STUDNAME                    GRADE HOBBIES
----- --------------------------- ----- ------------------------------
1     Michelle Jacintha           VII   Internet surfing
2     Mabel Mathews               VII   Baseball
SQL>
```

Objective: To find documents from the Students collection where the StudName ends in "s".

Act:

```
db.Students.find({StudName:/s$/}).pretty();
```

Outcome:

```
> db.Students.find({StudName:/s$/}).pretty();
{
        "_id" : 4,
        "Grade" : "VII",
        "StudName" : "Hersch Gibbs",
        "Hobbies" : "Graffiti"
}
{
        "_id" : 2,
        "StudName" : "Mabel Mathews",
        "Grade" : "VII",
        "Hobbies" : "Baseball"
}
>
```

RDBMS Equivalent:

```
Select *
        From Students
                Where StudName like '%s';
```

```
SQL> select * from Students where StudName like '%s';
STUDR STUDNAME                    GRADE HOBBIES
----- --------------------------- ----- ------------------------------
2     Mabel Mathews               VII   Baseball
4     Hersch Gibbs                VII   Graffiti
SQL>
```

Objective: To find documents from the Students collection where the StudName has an "e" in any position.

Act:

```
db.Students.find({StudName:/e/}).pretty();
```

OR

```
db.Students.find({StudName:/.*e.*/}).pretty();
```

OR

```
db.Students.find({StudName:{$regex:"e"}}).pretty();
```

Outcome:

```
C:\windows\system32\cmd.exe - mongo
> db.Students.find({StudName:/e/}).pretty();
{
        "_id" : 1,
        "StudName" : "Michelle Jacintha",
        "Grade" : "VII",
        "Hobbies" : "Internet Surfing"
}
{
        "_id" : 4,
        "Grade" : "VII",
        "StudName" : "Hersch Gibbs",
        "Hobbies" : "Graffiti"
}
{
        "_id" : 2,
        "StudName" : "Mabel Mathews",
        "Grade" : "VII",
        "Hobbies" : "Baseball"
}
>
```

RDBMS Equivalent:

Select *

From Students

Where StudName like '%e%';

```
SQL Plus
SQL> select * from Students where StudName like '%e%';
STUDR STUDNAME                    GRADE HOBBIES
----- --------------------------- ----- --------------------------------
1     Michelle Jacintha           VII   Internet surfing
2     Mabel Mathews               VII   Baseball
4     Hersch Gibbs                VII   Graffiti
SQL>
```

Objective: To find documents from the Students collection where the StudName ends in "a".

Act:

db.Students.find({StudName:{$regex:"a$"}}).pretty();

Outcome:

```
C:\windows\system32\cmd.exe - mongo
> db.Students.find({StudName:{$regex:"a$"}}).pretty();
{
        "_id" : 1,
        "StudName" : "Michelle Jacintha",
        "Grade" : "VII",
        "Hobbies" : "Internet Surfing"
}
>
```

RDBMS Equivalent:

Select *

From Students

Where StudName like '%a';

```
SQL Plus
SQL> select * from Students where StudName like '%a';
STUDR STUDNAME                    GRADE HOBBIES
----- --------------------------- ----- --------------------------------
1     Michelle Jacintha           VII   Internet surfing
SQL>
```

Objective: To find documents from the Students collection where the StudName begins with "M".

Act:

```
db.Students.find({StudName:{$regex:"^M"}}).pretty();
```

Outcome:

```
> db.Students.find({StudName:{$regex:"^M"}}).pretty();
{
        "_id" : 1,
        "StudName" : "Michelle Jacintha",
        "Grade" : "VII",
        "Hobbies" : "Internet Surfing"
}
{
        "_id" : 2,
        "StudName" : "Mabel Mathews",
        "Grade" : "VII",
        "Hobbies" : "Baseball"
}
>
```

RDBMS Equivalent:

```
Select *
        From Students
                Where StudName like 'M%';
```

```
SQL> select * from Students where StudName like 'M%';

STUDR STUDNAME                    GRADE HOBBIES
----- --------------------------- ----- ------------------------------
1     Michelle Jacintha           VII   Internet surfing
2     Mabel Mathews               VII   Baseball

SQL>
```

6.5.6 Dealing with NULL Values

Objective: To add a new field with null value in existing documents (_id:3 and _id:4) of Students collection. A NULL is a missing or unknown value. When we place NULL as a value for a field, it implies that currently we do not know the value or the value is missing. We can always update the value of the field once we know it.

Input: Before we execute the commands to update documents with a null value in a column, let us first view the two documents.

```
db.Students.find({$or:[{_id:3},{_id:4}]})
```

```
> db.Students.find({$or:[{_id:3},{_id:4}]});
{ "_id" : 3, "Grade" : "VII", "StudName" : "Aryan David", "Hobbies" : "Chess" }
{ "_id" : 4, "Grade" : "VII", "StudName" : "Hersch Gibbs", "Hobbies" : "Graffiti" }
>
```

Act: Update the documents with NULL values in the "Location" column.

```
db.Students.update({_id:3},{$set:{Location:null}});
db.Students.update({_id:4},{$set:{Location:null}});
```

```
> db.Students.update({_id:3},{$set:{Location:null}});
WriteResult({ "nMatched" : 1, "nUpserted" : 0, "nModified" : 1 })
> db.Students.update({_id:4},{$set:{Location:null}});
WriteResult({ "nMatched" : 1, "nUpserted" : 0, "nModified" : 1 })
>
```

RDBMS Equivalent:

Update Students
    Set Location = null
        Where StudRollNo in ('3','4');

```
SQL> update Students set Location = null where StudRollNo in ('3','4');
2 rows updated.
SQL>
```

Outcome: To search for NULL values in Location column.

db.Students.find({Location:{$eq:null}});

```
> db.Students.find({Location:{$eq:null}});
{ "_id" : 1, "StudName" : "Michelle Jacintha", "Grade" : "VII", "Hobbies" : "Internet Surfing" }
{ "_id" : 3, "Grade" : "VII", "StudName" : "Aryan David", "Hobbies" : "Chess", "Location" : null }
{ "_id" : 4, "Grade" : "VII", "StudName" : "Hersch Gibbs", "Hobbies" : "Graffiti", "Location" : null }
{ "_id" : ObjectId("5464849889ad1ab07d489b7f"), "StudName" : "Vamsi Bapat", "Grade" : "VI" }
{ "_id" : 2, "StudName" : "Mabel Mathews", "Grade" : "VII", "Hobbies" : "Baseball" }
>
```

The above statement displays documents which either have NULL values in the location column or do not have the location column at all.

RDBMS Equivalent:

Select *
    From Students
        Where Location is Null;

```
SQL> select * from Students where Location is NULL;
STUDR STUDNAME                    GRADE HOBBIES                        LOCATION
----- --------------------------- ----- ------------------------------ ----------
---------
1     Michelle Jacintha           VII   Internet surfing
3     Aryan David                 VII   Chess
2     Mabel Mathews               VII   Baseball
4     Hersch Gibbs                VII   Graffiti
      Vamsi Bapat                 VI

SQL>
```

Objective: To remove "Location" field having "NULL" values from the documents (_id:3 and _id:4) from the Students collection.

Input: Document from the "Students" collection having "NULL" values in the "Location" column.

```
> db.Students.find({Location:{$eq:null}});
{ "_id" : 1, "StudName" : "Michelle Jacintha", "Grade" : "VII", "Hobbies" : "Internet Surfing" }
{ "_id" : 3, "Grade" : "VII", "StudName" : "Aryan David", "Hobbies" : "Chess", "Location" : null }
{ "_id" : 4, "Grade" : "VII", "StudName" : "Hersch Gibbs", "Hobbies" : "Graffiti", "Location" : null }
{ "_id" : ObjectId("5464849889ad1ab07d489b7f"), "StudName" : "Vamsi Bapat", "Grade" : "VI" }
{ "_id" : 2, "StudName" : "Mabel Mathews", "Grade" : "VII", "Hobbies" : "Baseball" }
>
```

Act:

db.Students.update({_id:3},{$unset:{Location:null}});
db.Students.update({_id:4},{$unset:{Location:null}});

```
> db.Students.update({_id:3},{$unset:{Location:null}});
WriteResult({ "nMatched" : 1, "nUpserted" : 0, "nModified" : 0 })
> db.Students.update({_id:4},{$unset:{Location:null}});
WriteResult({ "nMatched" : 1, "nUpserted" : 0, "nModified" : 0 })
>
```

Outcome: Let us confirm if the changes have been made by running find method on the Students collection.

```
> db.Students.find({})
{ "_id" : 1, "StudName" : "Michelle Jacintha", "Grade" : "VII", "Hobbies" : '
{ "_id" : 3, "Grade" : "VII", "StudName" : "Aryan David", "Hobbies" : "Chess'
{ "_id" : 4, "Grade" : "VII", "StudName" : "Hersch Gibbs", "Hobbies" : "Graf'
{ "_id" : ObjectId("5464849889ad1ab07d489b7f"), "StudName" : "Vamsi Bapat",
```

6.5.7 Count, Limit, Sort, and Skip

Objective: To find the number of documents in the Students collection.

Act:

db.Students.count()

Outcome:

```
> db.Students.count()
5
>
```

Objective: To find the number of documents in the Students collection wherein the Grade is VII.

Act:

db.Students.count({Grade:"VII"});

Outcome:

```
> db.Students.count({Grade:"VII"});
4
>
```

Objective: To retrieve the first 3 documents from the Students collection wherein the Grade is VII.

Act:

db.Students.find({Grade:"VII"}).limit(3).pretty();

Outcome:

```
> db.Students.find({Grade:"VII"}).limit(3).pretty();
{
        "_id" : 1,
        "StudName" : "Michelle Jacintha",
        "Grade" : "VII",
        "Hobbies" : "Internet Surfing"
}
{
        "_id" : 3,
        "Grade" : "VII",
        "StudName" : "Aryan David",
        "Hobbies" : "Chess"
}
{
        "_id" : 4,
        "Grade" : "VII",
        "StudName" : "Hersch Gibbs",
        "Hobbies" : "Graffiti"
}
>
```

RDBMS Equivalent:

Select *

From Students

Where Grade like 'VII' and rownum < 4;

```
SQL> select * from Students where Grade like 'VII' and rownum < 4;
STUDR STUDNAME                    GRADE HOBBIES                          LOCATION
----- --------------------------- ----- -------------------------------- ----------
1     Michelle Jacintha           VII   Internet surfing
2     Aryan David                 VII   Chess
3     Mabel Mathews               VII   Baseball
SQL>
```

Objective: To sort the documents from the Students collection in the ascending order of StudName.

Act:

db.Students.find().sort({StudName:1}).pretty();

Outcome:

```
> db.Students.find().sort({StudName:1}).pretty();
{
        "_id" : 3,
        "Grade" : "VII",
        "StudName" : "Aryan David",
        "Hobbies" : "Chess"
}
{
        "_id" : 4,
        "Grade" : "VII",
        "StudName" : "Hersch Gibbs",
        "Hobbies" : "Graffiti"
}
{
        "_id" : 2,
        "StudName" : "Mabel Mathews",
        "Grade" : "VII",
        "Hobbies" : "Baseball"
}
{
        "_id" : 1,
        "StudName" : "Michelle Jacintha",
        "Grade" : "VII",
        "Hobbies" : "Internet Surfing"
}
{
        "_id" : ObjectId("5464849889ad1ab07d489b7f"),
        "StudName" : "Vamsi Bapat",
        "Grade" : "VI"
}
>
```

RDBMS Equivalent:

Select *

From Students

Order by StudName asc;

```
SQL> select * from Students order by StudName asc;
STUDR STUDNAME                    GRADE HOBBIES                          LOCATION
----- --------------------------- ----- -------------------------------- ----------
3     Aryan David                 VII   Chess
4     Hersch Gibbs                VII   Graffiti
2     Mabel Mathews               VII   Baseball
1     Michelle Jacintha           VII   Internet surfing
      Vamsi Bapat                 VI
SQL>
```

Objective: To sort the documents from the Students collection in the descending order of StudName.

Act:

db.Students.find().sort({StudName:-1}).pretty();

Outcome:

```
> db.Students.find().sort({StudName:-1}).pretty();
{
        "_id" : ObjectId("5464849889ad1ab07d489b7f"),
        "StudName" : "Vamsi Bapat",
        "Grade" : "VI"
}
{
        "_id" : 1,
        "StudName" : "Michelle Jacintha",
        "Grade" : "VII",
        "Hobbies" : "Internet Surfing"
}
{
        "_id" : 2,
        "StudName" : "Mabel Mathews",
        "Grade" : "VII",
        "Hobbies" : "Baseball"
}
{
        "_id" : 4,
        "Grade" : "VII",
        "StudName" : "Hersch Gibbs",
        "Hobbies" : "Graffiti"
}
{
        "_id" : 3,
        "Grade" : "VII",
        "StudName" : "Aryan David",
        "Hobbies" : "Chess"
}
```

RDBMS Equivalent:

Select *

From Students

Order by StudName desc;

```
SQL> select * from Students order by StudName desc;

STUDR STUDNAME                    GRADE HOBBIES                        LOCATION
----- --------------------------- ----- ------------------------------ ----------
---------
      Vamsi Bapat                 VI
1     Michelle Jacintha           VII   Internet surfing
2     Mabel Mathews               VII   Baseball
4     Hersch Gibbs                VII   Graffiti
3     Aryan David                 VII   Chess

SQL>
```

Objective: To sort the documents from the Students collection first on Grade in ascending order and then on Hobbies in descending order.

Act:

db.Students.find().sort({Grade:1, Hobbies:-1}).pretty();

Outcome:

```
> db.Students.find().sort({Grade:1, Hobbies:-1}).pretty();
{
        "_id" : ObjectId("5464849889ad1ab07d489b7f"),
        "StudName" : "Vamsi Bapat",
        "Grade" : "VI"
}
{
        "_id" : 1,
        "StudName" : "Michelle Jacintha",
        "Grade" : "VII",
        "Hobbies" : "Internet Surfing"
}
{
        "_id" : 4,
        "Grade" : "VII",
        "StudName" : "Hersch Gibbs",
        "Hobbies" : "Graffiti"
}
{
        "_id" : 3,
        "Grade" : "VII",
        "StudName" : "Aryan David",
        "Hobbies" : "Chess"
}
{
        "_id" : 2,
        "StudName" : "Mabel Mathews",
        "Grade" : "VII",
        "Hobbies" : "Baseball"
}
```

RDBMS Equivalent:

Select *

From Students

Order by Grade asc, hobbies desc;

```
SQL Plus
SQL> select * from Students order by Grade asc, Hobbies desc;
STUDR STUDNAME                    GRADE HOBBIES                          LOCATION
----- --------------------------- ----- -------------------------------- ----------
---------
      Vamsi Bapat                 VI
1     Michelle Jacintha           VII   Internet surfing
4     Hersch Gibbs                VII   Graffiti
3     Aryan David                 VII   Chess
2     Mabel Mathews               VII   Baseball
SQL>
```

Objective: To sort the documents from the Students collection first on Grade in ascending order and then on Hobbies in ascending order.

Act:

db.Students.find().sort({Grade:1, Hobbies:1}).pretty();

Outcome:

```
C:\windows\system32\cmd.exe - mongo
> db.Students.find().sort({Grade:1, Hobbies:1}).pretty();
{
        "_id" : ObjectId("5464849889ad1ab07d489b7f"),
        "StudName" : "Vamsi Bapat",
        "Grade" : "VI"
}
{
        "_id" : 2,
        "StudName" : "Mabel Mathews",
        "Grade" : "VII",
        "Hobbies" : "Baseball"
}
{
        "_id" : 3,
        "Grade" : "VII",
        "StudName" : "Aryan David",
        "Hobbies" : "Chess"
}
{
        "_id" : 4,
        "Grade" : "VII",
        "StudName" : "Hersch Gibbs",
        "Hobbies" : "Graffiti"
}
{
        "_id" : 1,
        "StudName" : "Michelle Jacintha",
        "Grade" : "VII",
        "Hobbies" : "Internet Surfing"
}
>
```

RDBMS Equivalent:

Select *

From Students

Order by Grade asc, Hobbies asc;

```
SQL Plus
SQL> select * from Students order by Grade asc, Hobbies asc;
STUDR STUDNAME                    GRADE HOBBIES                          LOCATION
----- --------------------------- ----- -------------------------------- ----------
---------
      Vamsi Bapat                 VI
2     Mabel Mathews               VII   Baseball
3     Aryan David                 VII   Chess
4     Hersch Gibbs                VII   Graffiti
1     Michelle Jacintha           VII   Internet surfing
SQL>
```

Objective: To skip the first 2 documents from the Students collection.

Act:

```
db.Students.find().skip(2).pretty();
```

Outcome:

```
> db.Students.find().skip(2).pretty();
{
        "_id" : 4,
        "Grade" : "VII",
        "StudName" : "Hersch Gibbs",
        "Hobbies" : "Graffiti"
}
{
        "_id" : ObjectId("5464849889ad1ab07d489b7f"),
        "StudName" : "Vamsi Bapat",
        "Grade" : "VI"
}
{
        "_id" : 2,
        "StudName" : "Mabel Mathews",
        "Grade" : "VII",
        "Hobbies" : "Baseball"
}
>
```

RDBMS Equivalent:

```
Select StudRollNo, StudName, Grade, Hobbies
        From (Select StudRollNo, StudName, Grade, Hobbies, RowNum as TheRowNum
                From Students)
                        Where TheRowNum > 2;
```

```
SQL> Select StudRollNo, StudName, Grade, Hobbies from (Select StudRollno, StudNa
me, Grade, Hobbies, Rownum as therownum from Students) where theRownum > 2;

STUDR STUDNAME                 GRADE HOBBIES
----- ------------------------ ----- --------------------------------
2     Mabel Mathews            VII   Baseball
4     Hersch Gibbs             VII   Graffiti
      Vamsi Bapat              VI

SQL>
```

Objective: To sort the documents from the Students collection and skip the first document from the output.

Act:

```
db.Students.find().skip(1).pretty().sort({StudName:1});
```

Outcome:

```
> db.Students.find().skip(1).pretty().sort({StudName:1});
{
        "_id" : 4,
        "Grade" : "VII",
        "StudName" : "Hersch Gibbs",
        "Hobbies" : "Graffiti"
}
{
        "_id" : 2,
        "StudName" : "Mabel Mathews",
        "Grade" : "VII",
        "Hobbies" : "Baseball"
}
{
        "_id" : 1,
        "StudName" : "Michelle Jacintha",
        "Grade" : "VII",
        "Hobbies" : "Internet Surfing"
}
{
        "_id" : ObjectId("5464849889ad1ab07d489b7f"),
        "StudName" : "Vamsi Bapat",
        "Grade" : "VI"
}
>
```

RDBMS Equivalent:

```
Select StudRollNo, StudName, Grade, Hobbies
    From (Select StudRollNo, StudName, Grade, Hobbies, RowNum as TheRowNum
        From Students)
            Where TheRowNum > 1
                Order by StudName;
```

```
SQL> Select StudRollNo, StudName, Grade, Hobbies from (Select StudRollno, StudNa
me, Grade, Hobbies, Rownum as therownum from Students) where theRownum >1 order
by StudName;

STUDR STUDNAME                    GRADE HOBBIES
----- --------------------------- ----- ------------------------------
3     Aryan David                 VII   Chess
4     Hersch Gibbs                VII   Graffiti
2     Mabel Mathews               VII   Baseball
      Vamsi Bapat                 VI

SQL>
```

Objective: To display the last 2 records from the Students collection.

Act:

db.Students.find().pretty().skip(db.Students.count()-2);

Outcome:

```
C:\windows\system32\cmd.exe - mongo
> db.Students.find().pretty().skip(db.Students.count()-2);
{
        "_id" : ObjectId("5464849889ad1ab07d489b7f"),
        "StudName" : "Vamsi Bapat",
        "Grade" : "VI"
}
{
        "_id" : 2,
        "StudName" : "Mabel Mathews",
        "Grade" : "VII",
        "Hobbies" : "Baseball"
}
>
```

Objective: To retrieve the third, fourth, and fifth document from the Students collection.

Act:

db.Students.find().pretty().skip(2).limit(3);

Outcome:

```
C:\windows\system32\cmd.exe - mongo
> db.Students.find().pretty().skip(2).limit(3);
{
        "_id" : 4,
        "Grade" : "VII",
        "StudName" : "Hersch Gibbs",
        "Hobbies" : "Graffiti"
}
{
        "_id" : ObjectId("5464849889ad1ab07d489b7f"),
        "StudName" : "Vamsi Bapat",
        "Grade" : "VI"
}
{
        "_id" : 2,
        "StudName" : "Mabel Mathews",
        "Grade" : "VII",
        "Hobbies" : "Baseball"
}
>
```

6.5.8 Arrays

Objective: To create a collection by the name "food" and then insert documents into the "food" collection. Each document should have a "fruits" array.

Act:

```
db.food.insert({_id:1,fruits:[ 'banana','apple','cherry' ] })
db.food.insert({_id:2,fruits:[ 'orange','butterfruit','mango' ]})
db.food.insert({_id:3,fruits:[ 'pineapple','strawberry','grapes']});
db.food.insert({_id:4,fruits:[ 'banana','strawberry','grapes']});
db.food.insert({_id:5,fruits:[ 'orange','grapes']});
```

```
C:\windows\system32\cmd.exe - mongo
> db.food.insert({_id:1,fruits:[ 'banana','apple','cherry' ] })
WriteResult({ "nInserted" : 1 })
> db.food.insert({_id:2,fruits:[ 'orange','butterfruit','mango' ]})
WriteResult({ "nInserted" : 1 })
> db.food.insert({_id:3,fruits:[ 'pineapple','strawberry','grapes']});
WriteResult({ "nInserted" : 1 })
> db.food.insert({_id:4,fruits:[ 'banana','strawberry','grapes']});
WriteResult({ "nInserted" : 1 })
> db.food.insert({_id:5,fruits:[ 'orange','grapes']});
WriteResult({ "nInserted" : 1 })
>
```

Outcome: Let us check if these documents are now in the "food" collection.

```
db.food.find({})
```

```
C:\windows\system32\cmd.exe - mongo
> db.food.find({})
{ "_id" : 1, "fruits" : [ "banana", "apple", "cherry" ] }
{ "_id" : 2, "fruits" : [ "orange", "butterfruit", "mango" ] }
{ "_id" : 3, "fruits" : [ "pineapple", "strawberry", "grapes" ] }
{ "_id" : 4, "fruits" : [ "banana", "strawberry", "grapes" ] }
{ "_id" : 5, "fruits" : [ "orange", "grapes" ] }
>
```

Objective: To find those documents from the "food" collection which has the "fruits array" constituted of "banana", "apple" and "cherry".

Act:

```
db.food.find({fruits:['banana','apple','cherry']}).pretty()
```

Outcome:

```
C:\windows\system32\cmd.exe - mongo
> db.food.find({fruits:['banana','apple','cherry']}).pretty()
{ "_id" : 1, "fruits" : [ "banana", "apple", "cherry" ] }
>
```

Objective: To find those documents from the "food" collection which has the "fruits" array having "banana", as an element.

Act:

```
db.food.find({fruits:'banana'})
```

Outcome:

```
C:\windows\system32\cmd.exe - mongo
> db.food.find({fruits:'banana'})
{ "_id" : 1, "fruits" : [ "banana", "apple", "cherry" ] }
{ "_id" : 4, "fruits" : [ "banana", "strawberry", "grapes" ] }
>
```

Objective: To find those documents from the "food" collection which have the "fruits" array having "grapes" in the first index position. The index position begins at 0.

Act:

db.food.find({'fruits.1':'grapes'})

Outcome:

```
C:\windows\system32\cmd.exe - mongo
> db.food.find({'fruits.1':'grapes'})
{ "_id" : 5, "fruits" : [ "orange", "grapes" ] }
>
```

Objective: To find those documents from the "food" collection where "grapes" is present in the 2nd index position of the "fruits" array.

Act:

db.food.find({'fruits.2':'grapes'})

Outcome:

```
C:\windows\system32\cmd.exe - mongo
> db.food.find({'fruits.2':'grapes'})
{ "_id" : 3, "fruits" : [ "pineapple", "strawberry", "grapes" ] }
{ "_id" : 4, "fruits" : [ "banana", "strawberry", "grapes" ] }
>
```

Objective: To find those documents from the "food" collection where the size of the array is two. The size implies that the array holds only 2 values.

Act:

db.food.find({"fruits":{$size:2}})

Outcome:

```
C:\windows\system32\cmd.exe - mongo
> db.food.find({"fruits":{$size:2}})
{ "_id" : 5, "fruits" : [ "orange", "grapes" ] }
>
```

Objective: To find those documents from the "food" collection where the size of the array is three. The size implies that the array holds only 3 values.

Act:

db.food.find({"fruits":{$size:3}})

Outcome:

```
C:\windows\system32\cmd.exe - mongo
> db.food.find({"fruits":{$size:3}});
{ "_id" : 1, "fruits" : [ "banana", "apple", "cherry" ] }
{ "_id" : 2, "fruits" : [ "orange", "butterfruit", "mango" ] }
{ "_id" : 3, "fruits" : [ "pineapple", "strawberry", "grapes" ] }
{ "_id" : 4, "fruits" : [ "banana", "strawberry", "grapes" ] }
>
```

Objective: To find the document with (_id: 1) from the "food" collection and display the first two elements from the array "fruits".

Act:

db.food.find({_id:1},{"fruits":{$slice:2}})

Outcome:

```
C:\windows\system32\cmd.exe - mongo
> db.food.find({_id:1},{"fruits":{$slice:2}})
{ "_id" : 1, "fruits" : [ "banana", "apple" ] }
>
```

Objective: To find all documents from the "food" collection which have elements "orange" and "grapes" in the array "fruits".

Act:

db.food.find ({fruits: {$all: ["orange", "grapes"]}}).pretty ();

Outcome:

```
C:\windows\system32\cmd.exe - mongo
> db.food.find ({fruits: {$all: ["orange", "grapes"]}}).pretty ();
{ "_id" : 5, "fruits" : [ "orange", "grapes" ] }
>
```

Objective: To find those documents from the "food" collection which have the element "orange" in the 0^{th} index position in the array "fruits".

Act:

db.food.find({ "fruits.0" : "orange" }).pretty();

Outcome:

```
C:\windows\system32\cmd.exe - mongo
> db.food.find({ "fruits.0" : "orange" }).pretty();
{ "_id" : 2, "fruits" : [ "orange", "butterfruit", "mango" ] }
{ "_id" : 5, "fruits" : [ "orange", "grapes" ] }
>
```

Objective: To find the document with (_id: 1) from the "food" collection and display two elements from the array "fruits", starting with the element at 0th index position.

Act:

```
db.food.find({_id:1},{"fruits":{$slice:[0,2]}})
```

Outcome:

```
> db.food.find({_id:1},{"fruits":{$slice:[0,2]}})
{ "_id" : 1, "fruits" : [ "banana", "apple" ] }
>
```

Objective: To find the document with (_id: 1) from the "food" collection and display two elements from the array "fruits", starting with the element at 1st index position.

Act:

```
db.food.find({_id:1},{"fruits":{$slice:[1,2]}})
```

Outcome:

```
> db.food.find({_id:1},{"fruits":{$slice:[1,2]}});
{ "_id" : 1, "fruits" : [ "apple", "cherry" ] }
>
```

Objective: To find the document with (_id: 1) from the "food" collection and display three elements from the array "fruits", starting with the element at 2nd index position. Since we have only 3 elements in the array "fruits" for the document with _id:1, it displays only one element, the element at 2nd index position, that is, "cherry".

Act:

```
db.food.find({_id:1},{"fruits":{$slice:[2,3]}})
```

Outcome:

```
> db.food.find({_id:1},{"fruits":{$slice:[2,3]}});
{ "_id" : 1, "fruits" : [ "cherry" ] }
>
```

6.5.8.1 Update on the Array

Before we begin the update operations on the "fruits" array of the documents of "food" collection, let us take a look at the documents that we have in the "food" collection:

```
> db.food.find({});
{ "_id" : 1, "fruits" : [ "banana", "apple", "cherry" ] }
{ "_id" : 2, "fruits" : [ "orange", "butterfruit", "mango" ] }
{ "_id" : 3, "fruits" : [ "pineapple", "strawberry", "grapes" ] }
{ "_id" : 4, "fruits" : [ "banana", "strawberry", "grapes" ] }
{ "_id" : 5, "fruits" : [ "orange", "grapes" ] }
>
```

Objective: To update the document with "_id:4" and replace the element present in the 1st index position of the "fruits" array with "apple".

Act:

```
db.food.update({_id:4},{$set:{'fruits.1': 'apple'}})
```

```
C:\windows\system32\cmd.exe - mongo
> db.food.update({_id:4},{$set:{'fruits.1': 'apple'}})
WriteResult({ "nMatched" : 1, "nUpserted" : 0, "nModified" : 1 })
>
```

Outcome: Let us take a look at how this update has changed our document.

```
C:\windows\system32\cmd.exe - mongo
> db.food.find({_id:4});
{ "_id" : 4, "fruits" : [ "banana", "apple", "grapes" ] }
>
```

Objective: To update the document with "_id:1" and replace the element "apple" of the "fruits" array with "An apple".

Act:

```
db.food.update({_id:1, 'fruits':'apple'},{$set:{'fruits.$': 'An apple' }})
```

```
C:\windows\system32\cmd.exe - mongo
> db.food.update({_id:1, 'fruits':'apple'},{$set:{'fruits.$': 'An apple' }})
WriteResult({ "nMatched" : 1, "nUpserted" : 0, "nModified" : 1 })
>
```

Outcome: The document after update is as follows.

```
C:\windows\system32\cmd.exe - mongo
> db.food.find({_id:1});
{ "_id" : 1, "fruits" : [ "banana", "An apple", "cherry" ] }
>
```

Objective: To update the document with "_id:2" and push new key value pairs in the "fruits" array.

Act:

```
db.food.update({_id:2},{$push:{price:{orange:60,butterfruit:200,mango:120}}})
```

```
C:\windows\system32\cmd.exe - mongo
> db.food.update({_id:2},{$push:{price:{orange:60,butterfruit:200,mango:120}}});
WriteResult({ "nMatched" : 1, "nUpserted" : 0, "nModified" : 1 })
>
```

Outcome:

```
C:\windows\system32\cmd.exe - mongo
> db.food.find().pretty();
{ "_id" : 1, "fruits" : [ "banana", "An apple", "cherry" ] }
{ "_id" : 3, "fruits" : [ "pineapple", "strawberry", "grapes" ] }
{ "_id" : 4, "fruits" : [ "banana", "apple", "grapes" ] }
{ "_id" : 5, "fruits" : [ "orange", "grapes" ] }
{
        "_id" : 2,
        "fruits" : [
                "orange",
                "butterfruit",
                "mango"
        ],
        "price" : [
                {
                        "orange" : 60,
                        "butterfruit" : 200,
                        "mango" : 120
                }
        ]
}
>
```

6.5.8.2 Further Updates to the Array "fruits" ...

Before we do the updates to the documents in the food collection, let us look at the current state:

```
C:\windows\system32\cmd.exe - mongo
> db.food.find().pretty();
{ "_id" : 1, "fruits" : [ "banana", "An apple", "cherry" ] }
{ "_id" : 3, "fruits" : [ "pineapple", "strawberry", "grapes" ] }
{ "_id" : 4, "fruits" : [ "banana", "apple", "grapes" ] }
{ "_id" : 5, "fruits" : [ "orange", "grapes" ] }
{
        "_id" : 2,
        "fruits" : [
                "orange",
                "butterfruit",
                "mango"
        ],
        "price" : [
                {
                        "orange" : 60,
                        "butterfruit" : 200,
                        "mango" : 120
                }
        ]
}
>
```

Objective: To update the document with "_id:4" by adding an element "orange" to the list of elements in the array "fruits".

Act:

db.food.update({_id:4},{$addToSet:{fruits:"orange"}});

```
C:\windows\system32\cmd.exe - mongo
> db.food.update({_id:4},{$addToSet:{fruits:"orange"}});
WriteResult({ "nMatched" : 1, "nUpserted" : 0, "nModified" : 1 })
>
```

Outcome: The result after the execution of the statement is as follows.

```
C:\windows\system32\cmd.exe - mongo
> db.food.find().pretty();
{ "_id" : 1, "fruits" : [ "banana", "An apple", "cherry" ] }
{ "_id" : 3, "fruits" : [ "pineapple", "strawberry", "grapes" ] }
{ "_id" : 4, "fruits" : [ "banana", "apple", "grapes", "orange" ] }
{ "_id" : 5, "fruits" : [ "orange", "grapes" ] }
{
        "_id" : 2,
        "fruits" : [
                "orange",
                "butterfruit",
                "mango"
        ],
        "price" : [
                {
                        "orange" : 60,
                        "butterfruit" : 200,
                        "mango" : 120
                }
        ]
}
>
```

Objective: To update the document with "_id:4" by popping an element from the list of elements present in the array "fruits". The element popped is the one from the end of the array.

Act:

db.food.update({_id:4},{$pop:{fruits:1}});

```
C:\windows\system32\cmd.exe - mongo
> db.food.update({_id:4},{$pop:{fruits:1}});
WriteResult({ "nMatched" : 1, "nUpserted" : 0, "nModified" : 1 })
>
```

Outcome: The "food" collection after the execution of the statement is as follows.

```
C:\windows\system32\cmd.exe - mongo
> db.food.find().pretty();
{ "_id" : 1, "fruits" : [ "banana", "An apple", "cherry" ] }
{ "_id" : 3, "fruits" : [ "pineapple", "strawberry", "grapes" ] }
{ "_id" : 4, "fruits" : [ "banana", "apple", "grapes" ] }
{ "_id" : 5, "fruits" : [ "orange", "grapes" ] }
{
        "_id" : 2,
        "fruits" : [
                "orange",
                "butterfruit",
                "mango"
        ],
        "price" : [
                {
                        "orange" : 60,
                        "butterfruit" : 200,
                        "mango" : 120
                }
        ]
}
>
```

Objective: To update the document with "_id:4" by popping an element from the list of elements present in the array "fruits". The element popped is the one from the beginning of the array.

Act:

db.food.update({_id:4},{$pop:{fruits:-1}});

```
C:\windows\system32\cmd.exe - mongo
> db.food.update({_id:4},{$pop:{fruits:-1}});
WriteResult({ "nMatched" : 1, "nUpserted" : 0, "nModified" : 1 })
>
```

Outcome: The "food" collection after the execution of the above update statement is as follows.

```
C:\windows\system32\cmd.exe - mongo
> db.food.find().pretty();
{ "_id" : 1, "fruits" : [ "banana", "An apple", "cherry" ] }
{ "_id" : 3, "fruits" : [ "pineapple", "strawberry", "grapes" ] }
{ "_id" : 4, "fruits" : [ "apple", "grapes" ] }
{ "_id" : 5, "fruits" : [ "orange", "grapes" ] }
{
        "_id" : 2,
        "fruits" : [
                "orange",
                "butterfruit",
                "mango"
        ],
        "price" : [
                {
                        "orange" : 60,
                        "butterfruit" : 200,
                        "mango" : 120
                }
        ]
}
>
```

Objective: To update the document with "_id:3" by popping two elements from the list of elements present in the array "fruits". The elements popped are "pineapple" and "grapes".

The document with "_id:3" before the update is

```
C:\windows\system32\cmd.exe - mongo
> db.food.find({_id:3});
{ "_id" : 3, "fruits" : [ "pineapple", "strawberry", "grapes" ] }
>
```

Act:

db.food.update({_id:3},{$pullAll:{fruits: ['pineapple','grapes']}});

```
C:\windows\system32\cmd.exe - mongo
> db.food.update({_id:3},{$pullAll:{fruits: [ 'pineapple','grapes' ]}});
WriteResult({ "nMatched" : 1, "nUpserted" : 0, "nModified" : 1 })
>
```

Outcome: The document with "_id:3" after the update is as follows:

```
C:\windows\system32\cmd.exe - mongo
> db.food.find({_id:4});
{ "_id" : 4, "fruits" : [ "apple", "grapes" ] }
>
```

Objective: To update the documents having "banana" as an element in the array "fruits" and pop out the element "banana" from those documents.

The "food" collection before the update is as follows:

```
C:\windows\system32\cmd.exe - mongo
> db.food.find().pretty();
{ "_id" : 1, "fruits" : [ "banana", "An apple", "cherry" ] }
{ "_id" : 3, "fruits" : [ "strawberry" ] }
{ "_id" : 4, "fruits" : [ "apple", "grapes" ] }
{ "_id" : 5, "fruits" : [ "orange", "grapes" ] }
{
        "_id" : 2,
        "fruits" : [
                "orange",
                "butterfruit",
                "mango"
        ],
        "price" : [
                {
                        "orange" : 60,
                        "butterfruit" : 200,
                        "mango" : 120
                }
        ]
}
>
```

Act:

db.food.update({fruits:'banana'}, {$pull:{fruits:'banana'}})

```
C:\windows\system32\cmd.exe - mongo
> db.food.update({fruits:'banana'}, {$pull:{fruits:'banana'}});
WriteResult({ "nMatched" : 1, "nUpserted" : 0, "nModified" : 1 })
>
```

Outcome: The "food" collection after the update is as follows:

```
C:\windows\system32\cmd.exe - mongo
> db.food.find().pretty();
{ "_id" : 1, "fruits" : [ "An apple", "cherry" ] }
{ "_id" : 3, "fruits" : [ "strawberry" ] }
{ "_id" : 4, "fruits" : [ "apple", "grapes" ] }
{ "_id" : 5, "fruits" : [ "orange", "grapes" ] }
{
        "_id" : 2,
        "fruits" : [
                "orange",
                "butterfruit",
                "mango"
        ],
        "price" : [
                {
                        "orange" : 60,
                        "butterfruit" : 200,
                        "mango" : 120
                }
        ]
}
>
```

Objective: To pull out an array element based on index position.

There is no direct way of pulling the array elements by looking up their index numbers. However a workaround is available. The document with "_id:4" in the food collection prior to the update is as follows:

```
C:\windows\system32\cmd.exe - mongo
> db.food.find({_id:4}).pretty();
{ "_id" : 4, "fruits" : [ "apple", "grapes" ] }
>
```

Act: The update statement is

db.food.update({_id:4}, {$unset : {"fruits.1" : null }}) ;
db.food.update({_id:4}, {$pull : {"fruits" : null}});

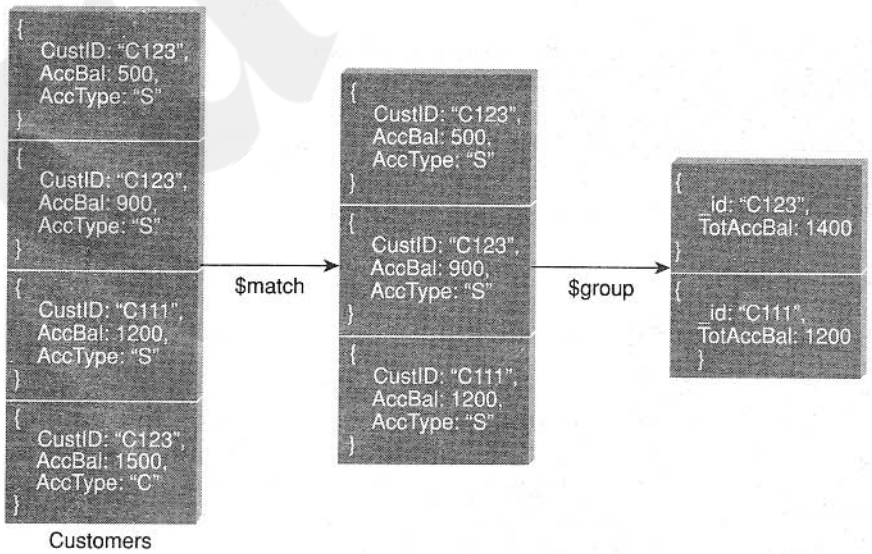
```
C:\windows\system32\cmd.exe - mongo
> db.food.update({_id:4}, {$unset : {"fruits.1" : null }}) ;
WriteResult({ "nMatched" : 1, "nUpserted" : 0, "nModified" : 1 })
> db.food.update({_id:4}, {$pull : {"fruits" : null}});
WriteResult({ "nMatched" : 1, "nUpserted" : 0, "nModified" : 1 })
>
```

Outcome: After update, the document with _id:4 in the food collection is

```
C:\windows\system32\cmd.exe - mongo
> db.food.find({_id:4}).pretty();
{ "_id" : 4, "fruits" : [ "apple" ] }
>
```

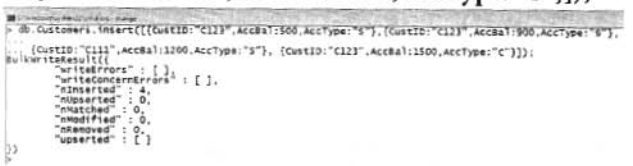
6.5.9 Aggregate Function

Objective: Consider the collection "Customers" as given below. It has four documents. We would like to filter out those documents where the "AccType" has a value other than "S". After the filter, we should be left with three documents where the "Acctype": "S". It is then required to group the documents on the basis of CustID and sum up the "AccBal" for each unique "CustID". This is similar to the output received with group by clause in RDBMS. Once the groups have been formed [as per the example below, there will be only two groups: (a) "CustID" : "C123" and (b) "CustID" : "C111"], filter and display that group where the "TotAccBal" column has a value greater than 1200.



Let us start off by creating the collection "Customers" with the above displayed four documents:

```
db.Customers.insert([{CustID:"C123",AccBal:500,AccType:"S"},
{CustID:"C123", AccBal: 900, AccType:"S"},
{CustID:"C111", AccBal: 1200, AccType:"S"},
{CustID:"C123", AccBal: 1500, AccType:"C"}]);
```

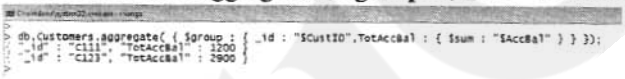


To confirm the presence of four documents in the "Customers" collection, use the below syntax:

```
db.Customers.find().pretty();
```

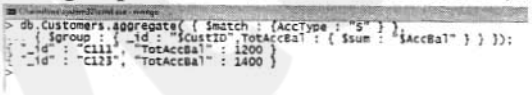
To group on "CustID" and compute the sum of "AccBal", use the below syntax:

```
db.Customers.aggregate( { $group : { _id : "$CustID",TotAccBal : { $sum : "$AccBal" } } });
```



In order to first filter on "AccType:S" and then group it on "CustID" and then compute the sum of "AccBal", use the below syntax:

```
db.Customers.aggregate( { $match : {AccType : "S" } },
{ $group : { _id : "$CustID",TotAccBal : { $sum : "$AccBal" } } });
```



In order to first filter on "AccType:S" and then group it on "CustID" and then to compute the sum of "AccBal" and then filter those documents wherein the "TotAccBal" is greater than 1200, use the below syntax:

```
db.Customers.aggregate( { $match : {AccType : "S" } },
{ $group : { _id : "$CustID",TotAccBal : { $sum : "$AccBal" } } },
{ $match : {TotAccBal : { $gt : 1200 } }});
```

To group on "CustID" and compute the average of the "AccBal" for each group:

```
db.Customers.aggregate( { $group : { _id : "$CustID",TotAccBal : { $avg : "$AccBal" } } });
```

```
> db.Customers.aggregate( { $group : { _id : "$CustID",TotAccBal : { $avg : "$AccBal" } } });
{ "_id" : "C111", "TotAccBal" : 1200 }
{ "_id" : "C123", "TotAccBal" : 966.6666666666666 }
>
```

To group on "CustID" and determine the maximum "AccBal" for each group:

```
db.Customers.aggregate( { $group : { _id : "$CustID",TotAccBal : { $max : "$AccBal" } } });
```

```
>
> db.Customers.aggregate( { $group : { _id : "$CustID",TotAccBal : { $max : "$AccBal" } } });
{ "_id" : "C111", "TotAccBal" : 1200 }
{ "_id" : "C123", "TotAccBal" : 1500 }
>
>
```

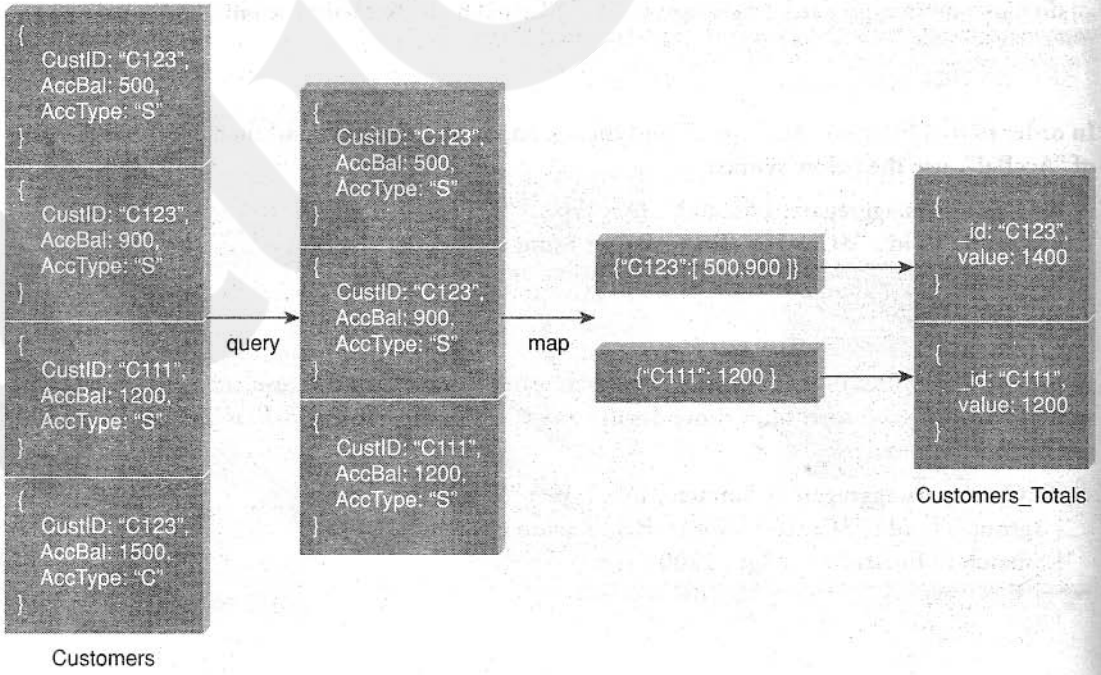
To group on "CustID" and determine the minimum "AccBal" for each group:

```
db.Customers.aggregate( { $group : { _id : "$CustID",TotAccBal : { $min : "$AccBal" } } });
```

```
> db.Customers.aggregate( { $group : { _id : "$CustID",TotAccBal : { $min : "$AccBal" } } });
{ "_id" : "C111", "TotAccBal" : 1200 }
{ "_id" : "C123", "TotAccBal" : 500 }
>
```

6.5.10 MapReduce Function

Objective: Consider the collection "Customers" below. There are four documents. Run a query to filter out those documents where the key "AccType" has a value other than "S". Then for each unique CustID, prepare a list of AccBal values. For example, for CustID: "C123", the AccBals are 500,900. This task will be assigned to the mapper function. The output from the mapper function serves as the input to the reducer function. The reducer function then aggregates the AccBal for each CustID. For example, for CustID: "C123", the value is 1400, etc.



Given below is the syntax that we will use to accomplish the objective.

```
db.Customers.mapReduce (
map          →      function() { emit ( this.CustID, this.AccBal ); },
reduce       →      function(key, values) { return Array.sum (values ) },
                    {
query        →          query: { AccType: "S"},
output       →          out: "Customer_Totals"
                    }
                    )
```

Map Function

```
var map = function(){
emit ( this.CustID, this.AccBal );}
```

```
> var map = function(){
... emit ( this.CustID, this.AccBal );}
>
```

Reduce Function

```
var reduce = function(key, values){ return Array.sum(values) ; }
```

```
> var reduce = function(key, values){ return Array.sum(values) ; }
>
```

To execute the query

```
db.Customers.mapReduce(map, reduce,{out: "Customer_Totals", query:{AccType:"S"}});
```

```
> db.Customers.mapReduce(map, reduce,{out: "Customer_Totals", query:{AccType:"S"}});
{
        "result" : "Customer_Totals",
        "timeMillis" : 7,
        "counts" : {
                "input" : 3,
                "emit" : 3,
                "reduce" : 1,
                "output" : 2
        },
        "ok" : 1,
}
>
```

The output as archived in "Customer_Totals" collection:

```
> db.Customer_Totals.find().pretty();
{ "_id" : "C111", "value" : 1200 }
{ "_id" : "C123", "value" : 1400 }
>
```

6.5.11 Java Script Programming

Objective: To compute the factorial of a given positive number. The user is required to create a function by the name "factorial" and insert it into the "system.js" collection.

Before we proceed, a quick check on what is contained in the "system.js" collection:

```
> db.system.js.find();
>
```

As per the screenshot above, currently there are no functions in the system.js collection.

Act:

```
db.system.js.insert({_id:"factorial",
```

```
value:function(n)
{
    if (n==1)
    return 1;
    else
    return n * factorial(n-1);
}
    }
);
```

```
> db.system.js.insert({_id:"factorial",
... value:function(n)
... {
...  if (n==1)
... return 1;
...  else
...  return n * factorial(n-1);
... }
... }
...  );
WriteResult({ "nInserted" : 1 })
>
```

Confirm the presence of the "factorial" function in the system.js collection.

```
> db.system.js.find();
{ "_id" : "factorial", "value" : function (n)
{
 if (n==1)
return 1;
 else
 return n * factorial(n-1);
 } }
>
```

To execute the function "factorial", use the eval() method.

db.eval("factorial(3)");

```
> db.eval("factorial(3)");
6
>
```

db.eval("factorial(5)");

```
> db.eval("factorial(5)");
120
>
```

db.eval("factorial(1)");

```
> db.eval("factorial(1)");
1
>
```

6.5.12 Cursors in MongoDB

Objective: To create a collection by the name "alphabets" and insert documents in it containing two fields, "_id" and "alphabet". The values stored in the "alphabet" field should be "a", "b", "c", "d", etc. with one value stored per document. There should be 26 documents in all. We need to use cursor to iterate through the "alphabets" collection.

Note: "Alphabets" is the name of the collection and "alphabet" is the name of the field.

Act: To create the collection "alphabets" with its 26 documents.

```
db.alphabets.insert({_id:1,alphabet:"a"});
db.alphabets.insert({_id:2,alphabet:"b"});
```

```
db.alphabets.insert({_id:3,alphabet:"c"});
db.alphabets.insert({_id:4,alphabet:"d"});
db.alphabets.insert({_id:5,alphabet:"e"});
db.alphabets.insert({_id:6,alphabet:"f"});
db.alphabets.insert({_id:7,alphabet:"g"});
db.alphabets.insert({_id:8,alphabet:"h"});
db.alphabets.insert({_id:9,alphabet:"i"});
db.alphabets.insert({_id:10,alphabet:"j"});
db.alphabets.insert({_id:11,alphabet:"k"});
db.alphabets.insert({_id:12,alphabet:"l"});
db.alphabets.insert({_id:13,alphabet:"m"});
db.alphabets.insert({_id:14,alphabet:"n"});
db.alphabets.insert({_id:15,alphabet:"o"});
db.alphabets.insert({_id:16,alphabet:"p"});
db.alphabets.insert({_id:17,alphabet:"q"});
db.alphabets.insert({_id:18,alphabet:"r"});
db.alphabets.insert({_id:19,alphabet:"s"});
db.alphabets.insert({_id:20,alphabet:"t"});
db.alphabets.insert({_id:21,alphabet:"u"});
db.alphabets.insert({_id:22,alphabet:"v"});
db.alphabets.insert({_id:23,alphabet:"w"});
db.alphabets.insert({_id:24,alphabet:"x"});
db.alphabets.insert({_id:25,alphabet:"y"});
db.alphabets.insert({_id:26,alphabet:"z"});
```

```
C:\windows\system32\cmd.exe - mongo
> db.alphabets.insert({_id:1,alphabet:"a"});
WriteResult({ "nInserted" : 1 })
> db.alphabets.insert({_id:2,alphabet:"b"});
WriteResult({ "nInserted" : 1 })
> db.alphabets.insert({_id:3,alphabet:"c"});
WriteResult({ "nInserted" : 1 })
> db.alphabets.insert({_id:4,alphabet:"d"});
WriteResult({ "nInserted" : 1 })
> db.alphabets.insert({_id:5,alphabet:"e"});
WriteResult({ "nInserted" : 1 })
> db.alphabets.insert({_id:6,alphabet:"f"});
WriteResult({ "nInserted" : 1 })
> db.alphabets.insert({_id:7,alphabet:"g"});
WriteResult({ "nInserted" : 1 })
> db.alphabets.insert({_id:8,alphabet:"h"});
WriteResult({ "nInserted" : 1 })
> db.alphabets.insert({_id:9,alphabet:"i"});
WriteResult({ "nInserted" : 1 })
> db.alphabets.insert({_id:10,alphabet:"j"});
WriteResult({ "nInserted" : 1 })
> db.alphabets.insert({_id:11,alphabet:"k"});
WriteResult({ "nInserted" : 1 })
> db.alphabets.insert({_id:12,alphabet:"l"});
WriteResult({ "nInserted" : 1 })
> db.alphabets.insert({_id:13,alphabet:"m"});
WriteResult({ "nInserted" : 1 })
> db.alphabets.insert({_id:14,alphabet:"n"});
WriteResult({ "nInserted" : 1 })
> db.alphabets.insert({_id:15,alphabet:"o"});
WriteResult({ "nInserted" : 1 })
> db.alphabets.insert({_id:16,alphabet:"p"});
WriteResult({ "nInserted" : 1 })
> db.alphabets.insert({_id:17,alphabet:"q"});
WriteResult({ "nInserted" : 1 })
> db.alphabets.insert({_id:18,alphabet:"r"});
WriteResult({ "nInserted" : 1 })
```

```
> db.alphabets.insert({_id:19,alphabet:"s"});
WriteResult({ "nInserted" : 1 })
> db.alphabets.insert({_id:21,alphabet:"u"});
WriteResult({ "nInserted" : 1 })
> db.alphabets.insert({_id:22,alphabet:"v"});
WriteResult({ "nInserted" : 1 })
> db.alphabets.insert({_id:23,alphabet:"w"});
WriteResult({ "nInserted" : 1 })
> db.alphabets.insert({_id:24,alphabet:"x"});
WriteResult({ "nInserted" : 1 })
> db.alphabets.insert({_id:25,alphabet:"y"});
WriteResult({ "nInserted" : 1 })
> db.alphabets.insert({_id:26,alphabet:"z"});
WriteResult({ "nInserted" : 1 })
>
```

Confirm the presence of 26 documents in the "alphabets" collection.

```
C:\windows\system32\cmd.exe - mongo
> db.alphabets.find();
{ "_id" : 1, "alphabet" : "a" }
{ "_id" : 2, "alphabet" : "b" }
{ "_id" : 3, "alphabet" : "c" }
{ "_id" : 4, "alphabet" : "d" }
{ "_id" : 5, "alphabet" : "e" }
{ "_id" : 6, "alphabet" : "f" }
{ "_id" : 7, "alphabet" : "g" }
{ "_id" : 8, "alphabet" : "h" }
{ "_id" : 9, "alphabet" : "i" }
{ "_id" : 10, "alphabet" : "j" }
{ "_id" : 11, "alphabet" : "k" }
{ "_id" : 12, "alphabet" : "l" }
{ "_id" : 13, "alphabet" : "m" }
{ "_id" : 14, "alphabet" : "n" }
{ "_id" : 15, "alphabet" : "o" }
{ "_id" : 16, "alphabet" : "p" }
{ "_id" : 17, "alphabet" : "q" }
{ "_id" : 18, "alphabet" : "r" }
{ "_id" : 19, "alphabet" : "s" }
{ "_id" : 20, "alphabet" : "t" }
Type "it" for more
>
```

A quick word on how the db.collection.find() method works. This is the primary method for read operation. In other words, it allows one to fetch the documents from the collection. To be able to access the documents, one needs to iterate the cursor.

However, in the mongo shell, if the returned cursor is not assigned to a variable using the var keyword, then cursor is automatically iterated up to 20 times to print the first 20 documents in the result.

```
C:\windows\system32\cmd.exe - mongo
> var myCursor = db.alphabets.find();
> myCursor;
{ "_id" : 1, "alphabet" : "a" }
{ "_id" : 2, "alphabet" : "b" }
{ "_id" : 3, "alphabet" : "c" }
{ "_id" : 4, "alphabet" : "d" }
{ "_id" : 5, "alphabet" : "e" }
{ "_id" : 6, "alphabet" : "f" }
{ "_id" : 7, "alphabet" : "g" }
{ "_id" : 8, "alphabet" : "h" }
{ "_id" : 9, "alphabet" : "i" }
{ "_id" : 10, "alphabet" : "j" }
{ "_id" : 11, "alphabet" : "k" }
{ "_id" : 12, "alphabet" : "l" }
{ "_id" : 13, "alphabet" : "m" }
{ "_id" : 14, "alphabet" : "n" }
{ "_id" : 15, "alphabet" : "o" }
{ "_id" : 16, "alphabet" : "p" }
{ "_id" : 17, "alphabet" : "q" }
{ "_id" : 18, "alphabet" : "r" }
{ "_id" : 19, "alphabet" : "s" }
{ "_id" : 20, "alphabet" : "t" }
Type "it" for more
>
```

Let us now look at designing manual cursors to iterate through the documents in the "alphabets" collection. We will use two methods with manual cursors: hasNext() and next(). We now quickly explain the two methods.

Method 1: hasNext() method. Return value: Boolean. The hasNext() method returns true if the cursor returned by the db.Collection.find() query can iterate further to return more documents.

Method 2: next() method. The next() method returns the next document in the cursor as returned by the db.collection.find() method.

```
C:\windows\system32\cmd.exe - mongo
> var myCur=db.alphabets.find({});
> while(myCur.hasNext()){
... var myRec=myCur.next();
... print("The alphabet is : " + myRec.alphabet);
... }
The alphabet is : a
The alphabet is : b
The alphabet is : c
The alphabet is : d
The alphabet is : e
The alphabet is : f
The alphabet is : g
The alphabet is : h
The alphabet is : i
The alphabet is : j
The alphabet is : k
The alphabet is : l
The alphabet is : m
The alphabet is : n
The alphabet is : o
The alphabet is : p
The alphabet is : q
The alphabet is : r
The alphabet is : s
The alphabet is : t
The alphabet is : u
The alphabet is : v
The alphabet is : w
The alphabet is : x
The alphabet is : y
The alphabet is : z
>
```

The same result can be obtained by iterating through the cursor using a forEach loop.

```
C:\windows\system32\cmd.exe - mongo
> var cur=db.alphabets.find({});
> var myRec;
> cur.forEach( function(myRec) {
... print("The alphabet is : " + myRec.alphabet);
...
... });
The alphabet is : a
The alphabet is : b
The alphabet is : c
The alphabet is : d
The alphabet is : e
The alphabet is : f
The alphabet is : g
The alphabet is : h
The alphabet is : i
The alphabet is : j
The alphabet is : k
The alphabet is : l
The alphabet is : m
The alphabet is : n
The alphabet is : o
The alphabet is : p
The alphabet is : q
The alphabet is : r
The alphabet is : s
The alphabet is : t
The alphabet is : u
The alphabet is : v
The alphabet is : w
The alphabet is : x
The alphabet is : y
The alphabet is : z
>
```

6.5.13 Indexes

Assume the collection with the following documents:

```
> db.books.find().pretty();
{
        "_id" : 6,
        "Category" : "Machine Learning",
        "Bookname" : "Machine Learning for Hackers",
        "Author" : "Drew Conway",
        "qty" : 25,
        "price" : 400,
        "rol" : 30,
        "pages" : 350
}
{
        "_id" : 7,
        "Category" : "Web Mining",
        "Bookname" : "Mining the Social Web",
        "Author" : "Matthew A.Russell",
        "qty" : 55,
        "price" : 500,
        "rol" : 30,
        "pages" : 250
}
{
        "_id" : 8,
        "Category" : "Python",
        "Bookname" : "Python for Data Analysis",
        "Author" : "Wes McKinney",
        "qty" : 8,
        "price" : 150,
        "rol" : 20,
        "pages" : 150
}
{
        "_id" : 9,
        "Category" : "Visualization",
        "Bookname" : "Visualizing Data",
        "Author" : "Ben Fry",
        "qty" : 12,
        "price" : 325,
        "rol" : 6,
        "pages" : 450
}
{
        "_id" : 10,
        "Category" : "Web Mining",
        "Bookname" : "Algorithms for the intelligent web",
        "Author" : "Haralambos Marmanis",
        "qty" : 5,
        "price" : 850,
        "rol" : 10,
        "pages" : 120
}
>
```

Create an index on the key "Category" in the "books" collection.

```
> db.books.ensureIndex({"Category":1});
{
        "createdCollectionAutomatically" : false,
        "numIndexesBefore" : 1,
        "numIndexesAfter" : 2,
        "ok" : 1
}
>
```

Check on the status, that is, number and name of the indexes:

```
> db.books.stats();
{
        "ns" : "test.books",
        "count" : 5,
        "size" : 1200,
        "avgObjSize" : 240,
        "storageSize" : 8192,
        "numExtents" : 1,
        "nindexes" : 2,
        "lastExtentSize" : 8192,
        "paddingFactor" : 1,
        "systemFlags" : 1,
```

```
        "userFlags" : 1,
        "totalIndexSize" : 16352,
        "indexSizes" : {
                "_id_" : 8176,
                "Category_1" : 8176
        },
        "ok" : 1
}
```

Get the list of all indexes on the "books" collection:

```
> db.books.getIndexes();
[
        {
                "v" : 1,
                "key" : {
                        "_id" : 1
                },
                "name" : "_id_",
                "ns" : "test.books"
        },
        {
                "v" : 1,
                "key" : {
                        "Category" : 1
                },
                "name" : "Category_1",
                "ns" : "test.books"
        }
]
```

To use the index on "Category" in the "books" collection, use the hint method:

```
> db.books.find({"Category":"Web Mining"}).pretty().hint({"Category":1});
{
        "_id" : 7,
        "Category" : "Web Mining",
        "Bookname" : "Mining the Social Web",
        "Author" : "Matthew A.Russell",
        "qty" : 55,
        "price" : 500,
        "rol" : 30,
        "pages" : 250
}
{
        "_id" : 10,
        "Category" : "Web Mining",
        "Bookname" : "Algorithms for the intelligent web",
        "Author" : "Haralambos Marmanis",
        "qty" : 5,
        "price" : 850,
        "rol" : 10,
        "pages" : 120
}
```

Check the explain plan to get a deeper understanding on the use of index.

```
> db.books.find({"Category":"Web Mining"}).pretty().hint({"Category":1}).explain();
{
        "cursor" : "BtreeCursor Category_1",
        "isMultiKey" : false,
        "n" : 2,
        "nscannedObjects" : 2,
        "nscanned" : 2,
        "nscannedObjectsAllPlans" : 2,
        "nscannedAllPlans" : 2,
        "scanAndOrder" : false,
        "indexOnly" : false,
        "nYields" : 0,
        "nChunkSkips" : 0,
```

```
        "millis" : 0,
        "indexBounds" : {
                "Category" : [
                        [
                                "Web Mining",
                                "Web Mining"
                        ]
                ]
        },
        "server" : "PUNITP123103L:27017",
        "filterSet" : false
}
```

Let us look at the case of covered index. Observe that the "indexOnly" property will be set to true for covered index.

```
> db.books.find({"Category":"Web Mining"},{"Category":1,_id:0}).pretty().hint({"Category":1}).explain();
{
        "cursor" : "BtreeCursor Category_1",
        "isMultiKey" : false,
        "n" : 2,
        "nscannedObjects" : 0,
        "nscanned" : 2,
        "nscannedObjectsAllPlans" : 0,
        "nscannedAllPlans" : 2,
        "scanAndOrder" : false,
        "indexOnly" : true,
        "nYields" : 0,
        "nChunkSkips" : 0,
        "millis" : 0,
        "indexBounds" : {
                "Category" : [
                        [
                                "Web Mining",
                                "Web Mining"
                        ]
                ]
        },
        "server" : "PUNITP123103L:27017",
        "filterSet" : false
}
```

In order to have the index cover the query, ensure that only those columns are projected on which the index is built. In the above example, the index is built on the "Category" column, and "Category" is the only column that is projected. Even the identifier (_id) is suppressed.

6.5.14 MongoImport

This command used at the command prompt imports CSV (Comma Separated Values) or TSV (Tab Separated Values) files or JSON (Java Script Object Notation) documents into MongoDB.

Objective: Given a CSV file "sample.txt" in the D: drive, import the file into the MongoDB collection, "SampleJSON". The collection is in the database "test".

The "sample.txt" file is as follows:

```
_id,FName,LName
1,Samuel,Jones
2,Virat,Kumar
3,Raul,"A Simpson"
4,,"Andrew Simon"
```

Act:

At the command prompt, execute the following command:

Mongoimport --db test --collection SampleJSON --type csv --headerline --file d:\sample.txt

On successful execution of the command, the message at the prompt will be as follows:

```
connected to: 127.0.0.1
2015-02-20T21:09:27.301+0530 imported 4 objects
```

Output: To confirm the output, log into MongoDB shell and navigate to the "SampleJSON" collection in the "test" database.

The following are the JSON documents in the collection:

```
> db
test
> show collections
Customers
SampleJSON
books
fs.chunks
fs.files
persons
system.indexes
usercounters
users
> db.SampleJSON.find().pretty();
{ "_id" : 1, "FName" : "Samuel", "LName" : "Jones" }
{ "_id" : 2, "FName" : "Virat", "LName" : "Kumar" }
{ "_id" : 3, "FName" : "Raul", "LName" : "A Simpson" }
{ "_id" : 4, "FName" : "", "LName" : "Andrew Simon" }
>
```

6.5.15 MongoExport

This command used at the command prompt exports MongoDB JSON documents into CSV (Comma Separated Values) or TSV (Tab Separated Values) files or JSON (Java Script Object Notation) documents.

Objective: This command used at the command prompt exports MongoDB JSON documents from "Customers" collection in the "test" database into a CSV file "Output.txt" in the D: drive.

Given below is a snapshot of the JSON documents in the "Customers" collection of the "test" database.

```
> db
test
> show collections
Customers
SampleJSON
books
fs.chunks
fs.files
persons
system.indexes
usercounters
users
> db.Customers.find().pretty();
{
        "_id" : ObjectId("54df6d4f46a31d28183b9a5b"),
        "CustID" : "C123",
        "AccBal" : 500,
        "AccType" : "S"
}
{
        "_id" : ObjectId("54df6d4f46a31d28183b9a5c"),
        "CustID" : "C123",
        "AccBal" : 900,
        "AccType" : "S"
}
```

```
{
        "_id" : ObjectId("54df6d4f46a31d28183b9a5d"),
        "CustID" : "C111",
        "AccBal" : 1200,
        "AccType" : "S"
}
{
        "_id" : ObjectId("54df6d4f46a31d28183b9a5e"),
        "CustID" : "C123",
        "AccBal" : 1500,
        "AccType" : "C"
}
>
```

Act: At the command prompt, execute the following command:

Mongoexport --db test --collection Customers --csv --fieldFile d:\fields.txt --out d:\output.txt

Before executing this command, ensure that you have created a "fields.txt" with a format defined as follows. The "fields.txt" file:

CustID
AccBal
AccType

For the MongoExport command to execute successfully, ensure that the fields are spelt as is in the MongoDB collection. The case also has to be maintained. It is mandatory to ensure that only one field name is placed per line.

On successful execution of the command, the message at the prompt will be as follows:

```
connected to: 127.0.0.1
exported 4 records
```

Output: To confirm the output, navigate to the D: drive and check the file "Output.txt".

"Output.txt"
CustID,AccBal,AccType
"C123",500.0,"S"
"C123",900.0,"S"
"C111",1200.0,"S"
"C123",1500.0,"C"

6.5.16 Automatic Generation of Unique Numbers for the "_id" Field

Step 1: Run the insert() method on a new collection "usercounters". This is to start off with an initial value of 0 for the "seq" field.

```
db.usercounters.insert(
{
_id: "empid",
seq:0
}
)
```

Step 2: Create a user-defined function "getnextseq". This method will invoke "findAndModify()" method on the "usercounters" collection. This is to increment the value of seq field by 1 and update the same in "usercounters" collection.

```
function getnextseq(name) {
var ret=db.usercounters.findAndModify(
{
query: {_id:name},
update: {$inc:{seq:1}},
new:true
}
);
return ret.seq;
}
```

Step 3: Run the insert() method on the collection where you need to have the "_id" field and get the uniquely generated number. Notice the call to getnextseq() method as value to _id. The return value from the getnextseq() method becomes the value of _id.

```
db.users.insert(
{
_id:getnextseq("empid"),
Name: "sarah jane"
}
)
```