Part B: NoSQL Programming

NoSQL originally referring to non SQL or non relational is a database that provides a mechanism for storage and retrieval of data. This data is modeled in means other than the tabular relations used in relational databases. Such databases came into existence in the late 1960s, but did not obtain the NoSQL moniker until a surge of popularity in the early twenty-first century.

NoSQL databases are used in real-time web applications and big data and their use are increasing over time. NoSQL systems are also sometimes called Not only SQL to emphasize the fact that they may support SQL-like query languages. A NoSQL database includes simplicity of design, simpler horizontal scaling to clusters of machines and finer control over availability. The data structures used by NoSQL databases are different from those used by default in relational databases which makes some operations faster in NoSQL. The suitability of a given NoSQL database depends on the problem it should solve. Data structures used by NoSQL databases are sometimes also viewed as more flexible than relational database tables. Many NoSQL stores compromise consistency in favor of availability, speed and partition tolerance. Barriers to the greater adoption of NoSQL stores include the use of low-level query languages, lack of standardized interfaces, and huge previous investments in existing relational databases. Most NoSQL stores lack true ACID(Atomicity, Consistency, Isolation, Durability) transactions but a few databases, such as MarkLogic, Aerospike, FairCom c-treeACE, Google Spanner (though technically a NewSQL database), Symas LMDB, and OrientDB have made them central to their designs. Most NoSQL databases offer a concept of eventual consistency in which database changes are propagated to all nodes so queries for data might not return updated data immediately or might result in reading data that is not accurate which is a problem known as stale reads. Also some NoSQL systems may exhibit lost writes and other forms of data loss. Some NoSQL systems provide concepts such as write-ahead logging to avoid data loss. For distributed transaction processing across multiple databases, data consistency is an even bigger challenge. This is difficult for both NoSQL and relational databases. Even current relational databases do not allow referential integrity constraints to span databases. There are few systems that maintain both X/Open XA standards and ACID transactions for distributed transaction processing.

MongoDB

MongoDB is a general-purpose document database designed for modern application development and for the cloud. Its scale-out architecture allows you to meet the increasing demand for your system by adding more nodes to share the load. Any relational database has a typical schema design that shows number of tables and the relationship between these tables. While in MongoDB, there is no concept of relationship.

Advantages of MongoDB over RDBMS

- **Schema less** − MongoDB is a document database in which one collection holds different documents. Number of fields, content and size of the document can differ from one document to another.
- Structure of a single object is clear.
- No complex joins.

- Deep query-ability. MongoDB supports dynamic queries on documents using a document-based query language that's nearly as powerful as SQL.
- Tuning.
- **Ease of scale-out** − MongoDB is easy to scale.
- Conversion/mapping of application objects to database objects not needed.
- Uses internal memory for storing the (windowed) working set, enabling faster access of data.

Why Use MongoDB?

- **Document Oriented Storage** − Data is stored in the form of JSON style documents.
- Index on any attribute
- Replication and high availability
- Auto-Sharding
- Rich queries
- Fast in-place updates
- Professional support by MongoDB

Where to Use MongoDB?

- Big Data
- Content Management and Delivery
- Mobile and Social Infrastructure
- User Data Management
- Data Hub

Creating Database:

The use Command: MongoDB use DATABASE_NAME is used to create database. The command will create a new database if it doesn't exist, otherwise it will return the existing database.

Syntax

Basic syntax of **use DATABASE** statement is as follows −

use DATABASE_NAME

```
> use mcadb;
switched to db mcadb
```

To check existing databases list, use the command **show dbs**.
> show dbs

Note : To display database, you need to insert at least one document into it.

In MongoDB default database is test. If you didn't create any database, then collections will be stored in test database.

Documents: The Records in a Document Database

MongoDB stores data as JSON documents.

The document data model maps naturally to objects in application code, making it simple for developers to learn and use.
The fields in a JSON document can vary from document to document. Compare that to a traditional relational database table, where adding a field means adding a column to the database table itself and therefore to every record in the database. Documents can be nested to express hierarchical relationships and to store structures such as arrays.
The document model provides flexibility to work with complex, fast-changing, messy data from numerous sources. It enables developers to quickly deliver new application functionality.
For faster access internally and to support more data types, MongoDB converts documents into a format called Binary JSON or BSON. But from a developer perspective, MongoDB is a JSON database.

Collections: Grouping Documents (tables in relational database)

In MongoDB, a collection is a group of documents.

If you are familiar with relational databases, you can think of a collection as a table. But collections in MongoDB are far more flexible. Collections do not enforce a schema, and documents in the same collection can have different fields.

Each collection is associated with one MongoDB database. To show which collections are in a particular database, use the command listCollections.

Create Collection

The createCollection() Method

MongoDB **db.createCollection(name, options)** is used to create collection.

Syntax

Basic syntax of **createCollection()** command is as follows −

db.createCollection(name, options)

In the command, **name** is name of collection to be created. **Options** is a document and is used to specify configuration of collection.

Practicals-B1

Create the below tables, insert suitable tuples and perform the following operations using MongoDB

Employee (SSN, Name, Job, Salary)
Project (ProjectNo, ProjectName, Duration)
Assigned_To (SSN, ProjectNo)

a) List the employees who are working with a particular designation
b) List the employees who are drawing the salary greater than 35000
c) List the employees who are working as Analyst and drawing the salary greater than 35000
d) List the employees who are working for a particular project.
e) Update the employee salary with a new value for particular employee.

To create a collection with a name “Employee”

```
> db.createCollection("Employee");
{ "ok" : 1 }
```

To create a collection with a name “Project”

```
> db.createCollection("Project");
{ "ok" : 1 }
```

To create a collection with a name “Assigned_to”

```
> db.createCollection("Assigned_to");
{ "ok" : 1 }
```

To insert data values in to a collection “Employee”

```
> db.Employee.insert({SSN:"E101",Ename:"ArunKumar",Job:"Engineer",Salary:"25000"});
> db.Employee.insert({SSN:"E102",Ename:"Akash",Job:"Engineer",Salary:"25000"});
> db.Employee.insert({SSN:"E103",Ename:"Aasha",Job:"Designer",Salary:"35000"});
> db.Employee.insert({SSN:"E104",Ename:"Bhaskar",Job:"Programmer",Salary:"40000"});
> db.Employee.insert({SSN:"E105",Ename:"Bharani",Job:"Programmer",Salary:"40000"});
> db.Employee.insert({SSN:"E106",Ename:"Dinesh",Job:"Analyst",Salary:"45000"});
> db.Employee.insert({SSN:"E107",Ename:"Dhanush",Job:"Analyst",Salary:"45000"});
> db.Employee.insert({SSN:"E108",Ename:"Druva",Job:"Tester",Salary:"30000"});
> db.Employee.insert({SSN:"E109",Ename:"Dwarak",Job:"Tester",Salary:"30000"});
> db.Employee.insert({SSN:"E110",Ename:"Girish",Job:"Engineer",Salary:"27000"});
```

To display the employee details from a collection “Employee”

```
> db.employee.find();          OR        > db.employee.find().pretty();
```

To insert data values in to a collection “Project”

```
> db.Project.insert({ProjectNo:"P1",ProjectName:"Core Banking",Duration:"24 Months"});
> db.Project.insert({ProjectNo:"P2",ProjectName:"Loan Lending",Duration:"6 Months"});
> db.Project.insert({ProjectNo:"P3",ProjectName:"Leave Management",Duration:"8 Months"});
> db.Project.insert({ProjectNo:"P4",ProjectName:"Biometric System",Duration:"7 Months"});
> db.Project.insert({ProjectNo:"P4",ProjectName:"Clinical Care Services",Duration:"12 Months"});
```

To display the project details from a collection “Project”

```
> db.Project.find();          OR          > db.Project.find().pretty();
```

To insert data values in to a collection “Assigned_to”

```
> db.Assigned_to.insert({SSN:"E101",ProjectNo:"P1"});
> db.Assigned_to.insert({SSN:"E102",ProjectNo:"P1"});
> db.Assigned_to.insert({SSN:"E103",ProjectNo:"P1"});
> db.Assigned_to.insert({SSN:"E104",ProjectNo:"P2"});
> db.Assigned_to.insert({SSN:"E105",ProjectNo:"P2"});
> db.Assigned_to.insert({SSN:"E106",ProjectNo:"P3"});
> db.Assigned_to.insert({SSN:"E107",ProjectNo:"P3"});
> db.Assigned_to.insert({SSN:"E108",ProjectNo:"P3"});
> db.Assigned_to.insert({SSN:"E109",ProjectNo:"P4"});
> db.Assigned_to.insert({SSN:"E110",ProjectNo:"P4"});
```

To display the employee details from a collection “Assigned_to”

```
> db.Assigned_to.find();   OR      > db.Assigned_to.find().pretty();
```

a) List the employees who are working with a particular designation

```
> db.Employee.find({Job:"Engineer"},{SSN:1,Ename:1,Job:1,Salary:1,_id:0});

{ "SSN" : "E101", "Ename" : "ArunKumar", "Job" : "Engineer", "Salary" : "25000" }
{ "SSN" : "E102", "Ename" : "Akash", "Job" : "Engineer", "Salary" : "25000" }
{ "SSN" : "E110", "Ename" : "Girish", "Job" : "Engineer", "Salary" : "27000" }
```

b) List the employees who are drawing the salary greater than 35000

```
> db.Employee.find({Salary:{$gt:"35000"}},{SSN:1,Ename:1,Job:1,Salary:1,_id:0});

{ "SSN" : "E104", "Ename" : "Bhaskar", "Job" : "Programmer", "Salary" : "40000" }
{ "SSN" : "E105", "Ename" : "Bharani", "Job" : "Programmer", "Salary" : "40000" }
{ "SSN" : "E106", "Ename" : "Dinesh", "Job" : "Analyst", "Salary" : "45000" }
{ "SSN" : "E107", "Ename" : "Dhanush", "Job" : "Analyst", "Salary" : "45000" }
```

c) List the employees who are working as Analyst and drawing the salary greater than 35000

```
> db.Employee.find({$and:[{Job:"Analyst"},{Salary:{$gt:"35000"}}]},{SSN:1,Ename:1,Job:1,Salary:1,_id:0});

{ "SSN" : "E106", "Ename" : "Dinesh", "Job" : "Analyst", "Salary" : "45000" }
{ "SSN" : "E107", "Ename" : "Dhanush", "Job" : "Analyst", "Salary" : "45000" }
```

d) List the employees who are working for a particular project.

Step-1 : Firstly, find the ProjectNo for the ProjectName “Core Banking” from the collection “Project”

```
> db.Project.find({ProjectName:"Core Banking"},{ProjectNo:1,_id:0});
{ "ProjectNo" : "P1" }
```

Step-2 : Then, find the employees (SSN) who are working for a project “Core Banking” from the collection “Assigned_to”

```
> db.Assigned_to.find({ProjectNo:"P1"},{SSN:1,_id:0});
{ "SSN" : "E101" }
{ "SSN" : "E102" }
{ "SSN" : "E103" }
```

Step-3 : Then, find the employee name for the SSN from the collection “employee”

```
> db.Employee.find({SSN:"E101"},{Ename:1,_id:0});
```

```
{ "Ename" : "ArunKumar" }

> db.Employee.find({SSN:"E102"},{Ename:1,_id:0});
{ "Ename" : "Akash" }

> db.Employee.find({SSN:"E103"},{Ename:1,_id:0});
{ "Ename" : "Aasha" }
```

Or by using OR operator

```
> db.Employee.find({$or:[{SSN:"E101"},{SSN:"E102"},{SSN:"E103"}]},{Ename:1,_id:0});
{ "Ename" : "ArunKumar" }
{ "Ename" : "Akash" }
{ "Ename" : "Aasha" }
```

e) Update the employee salary with a new value for particular employee.

Step-1 : Firstly, find the current salary of an employee with a SSN “E101”

```
> db.Employee.find({SSN:"E101"},{Salary:1,_id:0});
{ "Salary" : "25000" }
```

Step-2 : Then, Update the salary with a new value of “33000” to an employee with a SSN “E101”

```
> db.Employee.update({SSN:"E101"},{$set:{Salary:"33000"}});
WriteResult({ "nMatched" : 1, "nUpserted" : 0, "nModified" : 1 })
```

Step-3 : Then, Then display the updated salary of an employee with a SSN “E101”

```
> db.Employee.find({SSN:"E101"},{Salary:1,_id:0});
{ "Salary" : "33000" }
```

Practicals-B2

Create the below tables, insert suitable tuples and perform the following operations using MongoDB

Part (PartNo, PartName, Price, Colour),
Supplier (SupplierNo, SuplierName, Address)
Part_Supplier(PartNo,SupplierNo)

a) List the supplier name who are supplying particular parts
b) List the SteelGrey colored Part names whose price greater than ₹ 1000
c) List the part names which are supplied by suppliers from a particular address.
d) Update the price of the White colored parts with a new price.
e) Remove the suppliers who are supplying parts from a particular address.

To create a collection with a name “Part”

```
> db.createCollection("Part");
{ "ok" : 1 }
```

To create a collection with a name “Supplier”

```
> db.createCollection("Supplier");
{ "ok" : 1 }
```

To create a collection with a name “Part_Supplier”

```
> db.createCollection("Part_Supplier");
{ "ok" : 1 }
```

To insert data values in to a collection “Part”

```
> db.Part.insert({PartNo:"P1",PartName:"Mirror",Price:1250,Colour:"SteelGrey"});
> db.Part.insert({PartNo:"P2",PartName:"Bearings", Price:950,Colour:"SteelGrey"});
> db.Part.insert({PartNo:"P3",PartName:"GearBox", Price:1250,Colour:"Black"});
> db.Part.insert({PartNo:"P4",PartName:"SteeringWheel", Price:1350,Colour:"Black"});
> db.Part.insert({PartNo:"P5",PartName:"Speedometer", Price:3500,Colour:"White"});
> db.Part.insert({PartNo:"P6",PartName:"WiperBlades", Price:475,Colour:"Black"});
> db.Part.insert({PartNo:"P7",PartName:"WindowWisers", Price:850,Colour:"GreySilver"});
> db.Part.insert({PartNo:"P8",PartName:"Music Player", Price:6500,Colour:"GreySilver"});
> db.Part.insert({PartNo:"P9",PartName:"Mobile Charger", Price:550,Colour:"Black"});
> db.Part.insert({PartNo:"P10",PartName:"Cup Holder", Price:350,Colour:"Black"});
```

To display the Part details from a collection “Part”

```
> db.Part.find();    OR      > db.Part.find().pretty();
```

To insert data values in to a collection “Supplier”

```
> db.Supplier.insert({SupplierNo:"S1",SupplierName:"SS Enterprises",Address:"Bangalore"});
> db.Supplier.insert({SupplierNo:"S2",SupplierName:"KR India Ltd",Address:"Chennai"});
> db.Supplier.insert({SupplierNo:"S3",SupplierName:"SRS Automotives Ltd",Address:"Pune"});
> db.Supplier.insert({SupplierNo:"S4",SupplierName:"HM Machines Ltd",Address:"Noida"});
> db.Supplier.insert({SupplierNo:"S5",SupplierName:"MG Electronics Ltd",Address:"Bangalore"});
```

To display the supplier details from a collection “Supplier”

```
> db.Supplier.find();    OR          > db.Supplier.find().pretty();
```

To insert data values in to a collection “Part_Supplier”

```
> db.Part_Supplier.insert({PartNo:"P1",SupplierNo:"S1"});
> db.Part_Supplier.insert({PartNo:"P2",SupplierNo:"S2"});
> db.Part_Supplier.insert({PartNo:"P3",SupplierNo:"S3"});
> db.Part_Supplier.insert({PartNo:"P4",SupplierNo:"S4"});
> db.Part_Supplier.insert({PartNo:"P5",SupplierNo:"S5"});
> db.Part_Supplier.insert({PartNo:"P6",SupplierNo:"S1"});
> db.Part_Supplier.insert({PartNo:"P7",SupplierNo:"S2"});
> db.Part_Supplier.insert({PartNo:"P8",SupplierNo:"S3"});
> db.Part_Supplier.insert({PartNo:"P9",SupplierNo:"S1"});
> db.Part_Supplier.insert({PartNo:"P10",SupplierNo:"S4"});
```

To display the supplier details from a collection “Supplier”

```
> db.Part_Supplier.find();          OR          > db.Part_Supplier.find().pretty();
```

a) List the supplier name who are supplying particular parts

Step-1 : Firstly, find the PartNo for a particular PartName “Mirror” from a collection “Part”

```
> db.Part.find({PartName:"Mirror"},{PartNo:1,_id:0});
{ "PartNo" : "P1" }
```

Step-2 : Then, find the SupplierNo for a given PartNo “P1” from a collection “Part_Supplier”

```
> db.Part_Supplier.find({PartNo:"P1"},{SupplierNo:1,_id:0});
{ "SupplierNo" : "S1" }
```

Step-3 : Then, find the SupplierName for a given SupplierNo “S1” from a collection “Supplier”

```
> db.Supplier.find({SupplierNo:"S1"},{SupplierName:1,_id:0});
{ "SupplierName" : "SS Enterprises" }
```

b) List the SteelGrey colored Part names whose price greater than ₹ 1000

Step: Use AND operator to include two conditions (i.e. color is steelgrey and price is greater than 1000) from the collection Part

```
> db.Part.find({$and:[{Colour:"SteelGrey"},{Price:{$gt:"1000"}}]},{PartName:1,_id:0});
{ "PartName" : "Mirror" }
{ "PartName" : "Bearings" }
```

c) List the part names which are supplied by suppliers from a particular address.

Step-1 : Firstly, find the SupplierNo for a particular address “Chennai” from a collection “Supplier”

```
> db.Supplier.find({Address:"Chennai"},{SupplierNo:1,_id:0});
{ "SupplierNo" : "S2" }
```

Step-2 : Then, find the PartNo supplied by SupplierNo from a collection “Part_Supplier”

```
> db.Part_Supplier.find({SupplierNo:"S2"},{PartNo:1,_id:0});
{ "PartNo" : "P2" }
{ "PartNo" : "P7" }
```

Step-3 : Then, find the PartName for a particular PartNo (P2, P1) from a collection “Part”

```
> db.Part.find({PartNo:"P2"},{PartName:1,_id:0});
{ "PartName" : "Bearings" }

> db.Part.find({PartNo:"P7"},{PartName:1,_id:0});
{ "PartName" : "WindowWisers" }
```

OR
By using OR operator:

```
> db.Part.find({$or:[{PartNo:"P2"},{PartNo:"P7"}]},{PartName:1,_id:0});
{ "PartName" : "Bearings" }
{ "PartName" : "WindowWisers" }
```

d) Update the price of the White colored parts with a new price.

Step-1 : Firstly, find the PartNo and its Price for a Colour “White” from a collection “Part”

```
> db.Part.find({Colour:"White"},{PartNo:1,Colour:1,Price:1,_id:0});
{ "PartNo" : "P5", "Price" : 3500, "Colour" : "White" }
```

Step-2 : Then, update the Price of a part with a new price “5000” in a collection “Part”

```
> db.Part.update({Colour:"White"},{$set:{Price:"5000"}});
WriteResult({ "nMatched" : 1, "nUpserted" : 0, "nModified" : 1 })
```

Step-3 : Then, Check the collection “Part” with the change in the price for a PartNo “P5”

```
> db.Part.find({Colour:"White"},{PartNo:1,Price:1,_id:0});
{ "PartNo" : "P5", "Price" : "5000" }
```

e) Remove the suppliers along with the supplying parts from a particular address.

Step-1 : Firstly, find the supplier details from a collection “Supplier” for a particular address “Chennai”

```
> db.Supplier.find({Address:"Chennai"},{SupplierName:1,SupplierNo:1,_id:0});
{ "SupplierNo" : "S2", "SupplierName" : "KR India Ltd" }
```

Step-2 : Then, find the Parts they are supplying from particular address (“Chennai”).

```
> db.Part_Supplier.find({SupplierNo:"S2"},{PartNo:1,_id:0});
{ "PartNo" : "P2" }
{ "PartNo" : "P7" }
```

Step-3 : Then, remove the documents for the SupplierNo: “S2” from the collection “Part_Supplier”

```
> db.Part_Supplier.remove({SupplierNo:"S2"});
WriteResult({ "nRemoved" : 2 })
```

Step-4 : Lastly, remove the documents for the SupplierNo: “S2” from the collection “Supplier”

```
> db.Supplier.remove({SupplierNo:"S2"});
WriteResult({ "nRemoved" : 1 })
```

Step-5 : Confirm the removal of documents by checking the present status of the documents from the collections “Part_Supplier” and “Supplier”.

```
 > db.Part_Supplier.find();
…
> db.Supplier.find();
```

Practicals-B3

Create the below tables, insert suitable tuples and perform the following operations using MongoDB

Boat (BoatId, BoatName, Colour) ,
Reserves (BoatId, SailorName, SailorId, Day)

a) Find the boat name which was used on a particular day.
b) Obtain the number of boats obtained by a particular sailor :#sname
c) Retrieve boats of color :"GreyWhite" , "Red" and "SteelGrey"
d) Update the sailor name with a new name who was using particular boat name
e) Remove the boat reservations details for a particular boat.

To create a collection with a name "Boat"

```
> db.createCollection("Boat");
{ "ok" : 1 }
```

To create a collection with a name "Reserves"

```
> db.createCollection("Reserves");
{ "ok" : 1 }
```

To insert data values in to a collection "Boat"

```
> db.Boat.insert({BoatId:"B1",BoatName:"SRS Boat",Colour:"GreyWhite"});
> db.Boat.insert({BoatId:"B2",BoatName:"KSS Boat",Colour:"SteelGrey"});
> db.Boat.insert({BoatId:"B3",BoatName:"SKZ Boat",Colour:"Red"});
> db.Boat.insert({BoatId:"B4",BoatName:"MNS Boat",Colour:"DarkGreen"});
> db.Boat.insert({BoatId:"B5",BoatName:"SSS Boat",Colour:"SkyBlue"});
```

To insert data values in to a collection "Reserves"

```
> db.Reserves.insert({BoatId:"B1",SailorName:"Girish",SailorId:"S1",Day:"Sunday"});
> db.Reserves.insert({BoatId:"B2",SailorName:"Rajashekar",SailorId:"S2",Day:"Monday"});
> db.Reserves.insert({BoatId:"B3",SailorName:"Umesh",SailorId:"S3",Day:"Tuesday"});
> db.Reserves.insert({BoatId:"B4",SailorName:"Girish",SailorId:"S1",Day:"Wednesday"});
> db.Reserves.insert({BoatId:"B5",SailorName:"Rajashekar",SailorId:"S2",Day:"Thusday"});
> db.Reserves.insert({BoatId:"B1",SailorName:"Rajan",SailorId:"S3",Day:"Friday"});
> db.Reserves.insert({BoatId:"B2",SailorName:"Girish",SailorId:"S1",Day:"Saturday"});
```

To display the Boat details from a collection "Boat"

```
> db.Boat.find();        OR        > db.Boat.find().pretty();
```

To display the Reserve details from a collection "Reserves"

```
> db.Reserves.find();      OR      > db.Reserves.find().pretty();
```

a) Find the boat name which was used on a particular day.

Step-1 : Firstly, find the BoatId for the particular day from the collection "Reserves"

```
> db.Reserves.find({Day:"Friday"},{BoatId:1,_id:0});
{ "BoatId" : "B1" }
```

Step-2 : Then, find BoatName for the given BoatNo=”B#” from the collection “Boat”

```
> db.Boat.find({BoatId:"B1"},{BoatName:1,_id:0});
{ "BoatName" : "SRS Boat" }
```

b) Obtain the number of boats obtained by a particular sailor :#sname

Step-1 : Firstly, find the number of boats in the form of BoatId for the SailorName “Girish” from the collection “Reserves”

```
> db.Reserves.find({SailorName:"Girish"},{BoatId:1,_id:0});
{ "BoatId" : "B4" }
{ "BoatId" : "B2" }
{ "BoatId" : "B1" }
```

Step-2 : Then, find BoatName for the given BoatNo=”B#” from the collection “Boat”

```
> db.Boat.find({BoatId:"B4"},{BoatName:1,_id:0});
{ "BoatName" : "MNS Boat" }
> db.Boat.find({BoatId:"B2"},{BoatName:1,_id:0});
{ "BoatName" : "KSS Boat" }
> db.Boat.find({BoatId:"B1"},{BoatName:1,_id:0});
{ "BoatName" : "SRS Boat" }
```

OR by using OR operator

Find BoatName for the given BoatNo=”B#” by using OR operator from the collection “Boat”

```
> db.Boat.find({$or:[{BoatId:"B4"},{BoatId:"B2"},{BoatId:"B1"}]},{BoatName:1,_id:0});
{ "BoatName" : "SRS Boat" }
{ "BoatName" : "KSS Boat" }
{ "BoatName" : "MNS Boat" }
```

Or By using Group operator:

List the Sailor name along with number of boats they have assigned from the collection “Reserves” by using Group operator

```
> db.Reserves.aggregate([{$group:{_id: "$SailorName",NoofBoats:{$count:{}}}}]);
{ "_id" : "Rajashekar", "NoofBoats" : 2 }
{ "_id" : "Umesh", "NoofBoats" : 1 }
{ "_id" : "Rajan", "NoofBoats" : 1 }
{ "_id" : "Girish", "NoofBoats" : 3 }
```

c) Retrieve boats of color :”GreyWhite” , “Red” and “SteelGrey”

Find the BoatIds and their names for the given colour by using OR operator from the collection “Boat”

```
>db.Boat.find({$or:[{Colour:"GreyWhite"},{Colour:"Red"},{Colour:"SteelGrey"}]},{BoatId:1,BoatName:1,_id:0});
{ "BoatId" : "B1", "BoatName" : "SRS Boat" }
{ "BoatId" : "B2", "BoatName" : "KSS Boat" }
{ "BoatId" : "B3", "BoatName" : "SKZ Boat" }
```

OR

You can also find the BoatId and their name for the given colour from the collection “Boat”

```
> db.Boat.find({Colour:"GreyWhite"},{BoatId:1,BoatName:1,_id:0});
{ "BoatId" : "B1", "BoatName" : "SRS Boat" }
```

OR

```
> db.Boat.find({Colour:"SteelGrey"},{BoatId:1,BoatName:1,_id:0});
{ "BoatId" : "B2", "BoatName" : "KSS Boat" }
```

OR

```
> db.Boat.find({Colour:"Red"},{BoatId:1,BoatName:1,_id:0});
{ "BoatId" : "B3", "BoatName" : "SKZ Boat" }
```

d) Update the sailor name with a new name who was using particular boat name

Step-1 : Firstly, find the boatId for the BoatName “SSS Boat” from the collection “Reserves”

```
> db.Boat.find({BoatName:"SSS Boat"},{BoatId:1,_id:0});
{ "BoatId" : "B5" }
```

Step-2 : Then, find the SailorId for this Boat from the collection “Reserves”

```
> db.Reserves.find({BoatId:"B5"},{SailorId:1,_id:0});
{ "SailorId" : "S2" }
```

Step-3 : Then, find the existing SailorName from the collection “Reserves”

```
> db.Reserves.find({SailorId:"S2"},{SailorName:1,_id:0});
{ "SailorName" : "Rajashekar" }
{ "SailorName" : "Rajashekar" }
```

Step-4 : Then, update the existing Sailor name “Rajshekar” to new name “Rajshekar N Murthy” in the collection “Reserves”

```
> db.Reserves.update({SailorId:"S2"},{$set:{SailorName:"Rajashekar N Murthy"}});
WriteResult({ "nMatched" : 1, "nUpserted" : 0, "nModified" : 1 })
```

Step-5 : Then, Select the Sailor name from collection “Reserves” to check the update

```
> db.Reserves.find({SailorId:"S2"},{SailorName:1,_id:0});
{ "SailorName" : "Rajashkar N Murthy" }
{ "SailorName" : "Rajashekar" }
```

Note: By default, MongoDB will update only a single document. To update multiple documents, you need to set a parameter 'multi' to true.

To update multiple documents :

```
> db.Reserves.update({SailorId:"S2"},{$set:{SailorName:"Rajashekar N Murthy"}},{multi:true});
WriteResult({ "nMatched" : 2, "nUpserted" : 0, "nModified" : 2 })
```

Select the Sailor name from collection “Reserves” to check for the multiple updates

```
> db.Reserves.find({SailorId:"S2"},{SailorName:1,_id:0});
{ "SailorName" : "Rajashekar N Murthy" }
{ "SailorName" : "Rajashekar N Murthy" }
```

e) Remove the boat reservations details for a particular boat.

Step-1 : Firstly, find the boat name for the BoatId “B3” from the collection “Boat”

```
> db.Boat.find({BoatId:"B3"},{BoatName:1,_id:0});
{ "BoatName" : "SKZ Boat" }
```

Step-2 : Then, find the Sailor name for the BoatId “B3” from the collection “Reserves”

```
> db.Reserves.find({BoatId:"B3"},{SailorName:1,SailorId:1,_id:0});
{ "SailorName" : "Umesh", "SailorId" : "S3" }
```

Step-3 : Then, remove the Sailor details for the BoatId “B3” from the collection “Reserves”

```
> db.Reserves.remove({BoatId:"B3"});
WriteResult({ "nRemoved" : 1 })
```

Step-4 : Check the removal of the Sailor details for the BoatId “B3” from the collection “Reserves”

```
> db.Reserves.find({},{SailorName:1,BoatId:1,_id:0});
{ "BoatId" : "B2", "SailorName" : "Rajashekar N Murthy" }
{ "BoatId" : "B4", "SailorName" : "Girish" }
{ "BoatId" : "B5", "SailorName" : "Rajashekar N Murthy" }
{ "BoatId" : "B1", "SailorName" : "Rajan" }
{ "BoatId" : "B2", "SailorName" : "Girish" }
{ "BoatId" : "B1", "SailorName" : "Girish" }
```

Practicals-B4

Create the below tables, insert suitable tuples and perform the following operations using MongoDB

Part (PartNo, PartName, Price, Colour),
Warehouse (WarehouseNo, WarehouseName, Address)
Shipment (PartNo, WarehouseNo, Quantity, Date)

a) List the parts shipped from warehouse located at a particular address
b) List the total quantity of parts supplied to each warehouse
c) List the Warehouses along with maximum quantity of parts have been placed.
d) List the Parts and Warehouse names to which shipment takes place in a particular month
e) List the Warehouse name which starts with "S" as the first character.

To create a collection with a name "Part"

This collection has been already created in the practicals-2, use the same table.

To create a collection with a name "Shipment"

```
> db.createCollection("Shipment");
{ "ok" : 1 }
```

To insert data values in to a collection "Warehouse"

```
> db.Warehouse.insert({WarehouseNo:"WN1",WarehouseName:"SSS Warehouse",Address:"Tumkur"});
> db.Warehouse.insert({WarehouseNo:"WN2",WarehouseName:"SKS Warehouse",Address:"Bangalore"});
> db.Warehouse.insert({WarehouseNo:"WN3",WarehouseName:"LNZ Warehouse",Address:"Mysore"});
> db.Warehouse.insert({WarehouseNo:"WN4",WarehouseName:"RNS Warehouse",Address:"Shimoga"});
> db.Warehouse.insert({WarehouseNo:"WN5",WarehouseName:"KNS Warehouse",Address:"Kolar"});
```

To insert data values in to a collection "Shipment"

```
> db.Shipment.insert({PartNo:"P1",WarehouseNo:"WN1",QuantitY:"50",ShipDate:"12-jan-22"});
> db.Shipment.insert({PartNo:"P2",WarehouseNo:"WN2",QuantitY:"80",ShipDate:"21-jan-22"});
> db.Shipment.insert({PartNo:"P3",WarehouseNo:"WN3",QuantitY:"120",ShipDate:"23-Feb-22"});
> db.Shipment.insert({PartNo:"P4",WarehouseNo:"WN4",QuantitY:"75",ShipDate:"13-Feb-22"});
> db.Shipment.insert({PartNo:"P5",WarehouseNo:"WN5",QuantitY:"150",ShipDate:"18-Feb-22"});
> db.Shipment.insert({PartNo:"P6",WarehouseNo:"WN1",QuantitY:"125",ShipDate:"02-Mar-22"});
> db.Shipment.insert({PartNo:"P7",WarehouseNo:"WN1",QuantitY:"175",ShipDate:"05-Mar-22"});
> db.Shipment.insert({PartNo:"P8",WarehouseNo:"WN2",QuantitY:"235",ShipDate:"02-Mar-22"});
> db.Shipment.insert({PartNo:"P9",WarehouseNo:"WN3",QuantitY:"200",ShipDate:"07-Mar-22"});
> db.Shipment.insert({PartNo:"P10",WarehouseNo:"WN4",QuantitY:"160",ShipDate:"10-Mar-22"});
```

a) List the parts shipped from warehouse located at a particular address

Step-1 : Firstly, find the Warehouse details located in a particular address from the collection "Warehouse"

```
> db.Warehouse.find({Address:"Tumkur"},{WarehouseNo:1,WarehouseName:1,_id:0});
{ "WarehouseNo" : "WN1", "WarehouseName" : "SSS Warehouse" }
```

Step-2 : Then, find the parts shipped details to a particular warehouse from the collection "Shipment"

```
> db.Shipment.find({WarehouseNo:"WN1"},{PartNo:1,Quantity:1,ShipDate:1,_id:0});
{ "PartNo" : "P1", "ShipDate" : "12-jan-22" }
{ "PartNo" : "P6", "ShipDate" : "02-Mar-22" }
```

{ "PartNo" : "P7", "ShipDate" : "05-Mar-22" }

Step-3 : Then, find the name of the parts from the collection “Part” by using OR operator

```
> db.Part.find({$or:[{PartNo:"P1"},{PartNo:"P6"},{PartNo:"P7"}]},{PartName:1,_id:0});
{ "PartName" : "Mirror" }
{ "PartName" : "WiperBlades" }
{ "PartName" : "WindowWisers" }
```

b) List the total number of parts supplied to each warehouse

Step-1 : List the WarehouseNo and number of items shipped to each “Warehouse” by using group operator.

```
> db.Shipment.aggregate([{$group:{_id:"$WarehouseNo",Noofitems:{$count:{}}}}]);
{ "_id" : "WN4", "Noofitems" : 2 }
{ "_id" : "WN1", "Noofitems" : 3 }
{ "_id" : "WN2", "Noofitems" : 2 }
{ "_id" : "WN5", "Noofitems" : 1 }
{ "_id" : "WN3", "Noofitems" : 2 }
```

c) List the Warehouses along with maximum quantity of parts have been placed.

Step-1 : List the WarehouseNo and maximum quantity of items shipped to each “Warehouse” by using group operator and max operator.

```
> db.Shipment.aggregate([{$group:{_id: "$WarehouseNo",maxQuantity:{$max:"$Quantity"}}}])
{ "_id" : "WN5", "maxQuantity" : "150" }
{ "_id" : "WN1", "maxQuantity" : "50" }
{ "_id" : "WN2", "maxQuantity" : "80" }
{ "_id" : "WN4", "maxQuantity" : "75" }
{ "_id" : "WN3", "maxQuantity" : "200" }
```

d) List the Parts and Warehouse names to which shipment takes place in a particular month (“January”)

Step-1 : List the Part number and Warehouse names to which shipment takesplace in a particular month by using / / operator from the collection “Shipment”

```
> db.Shipment.find({ShipDate:/ja/},{PartNo:1,WarehouseNo:1,Quantity:1,ShipDate:1,_id:0});
{ "PartNo" : "P1", "WarehouseNo" : "WN1", "ShipDate" : "12-jan-22" }
{ "PartNo" : "P2", "WarehouseNo" : "WN2", "ShipDate" : "21-jan-22" }
```

e) List the Warehouse name which starts with “S” as the first character.

Step-1 : List the Warehouse name which starts with “S” as the first character by using / / operator from the collection “Warehouse”

```
> db.Warehouse.find({WarehouseName:/^S/},{WarehouseName:1,_id:0});
{ "WarehouseName" : "SSS Warehouse" }
{ "WarehouseName" : "SKS Warehouse" }
```

Practicals-B5

Create the below tables, insert suitable tuples and perform the following operations using MongoDB

Book (ISBN, Title, Author, Publisher),
Student (Usn, StudentName, Class)
Borrow (ISBN, Usn, BorrowedDate)

a) List the Book titles in which there is a particular keyword pattern “Computing”.
b) Obtain the Names of students who have borrowed DBMS books.
c) List the Author names who have authored more than 1 Book.
d) List the Publisher names who have published more than 2 Books.
e) List the student names who have borrowed more than 1 Book.

To create a collection with a name “Book”

```
> db.createCollection("Book");
{ "ok" : 1 }
```

To create a collection with a name “Student”

```
> db.createCollection("Student");
{ "ok" : 1 }
```

To create a collection with a name “Borrow”

```
> db.createCollection("Borrow");
{ "ok" : 1 }
```

To insert data values in to a collection “Book”

```
> db.Book.insert({ISBN:"BK1",Title:"Programming Fundamentals",Author:"SK Jain",Publisher:"BPN Publishers"});
> db.Book.insert({ISBN:"BK2",Title:"DBMS",Author:"Agarwal",Publisher:"TATA McGraw"});
> db.Book.insert({ISBN:"BK3",Title:"Analysis and Design of Algorithms",Author:"Arun Shah",Publisher:"Sapna Publishers"});
> db.Book.insert({ISBN:"BK4",Title:"Data Structures",Author:"Arun Shah",Publisher:"Sapna Publishers"});
> db.Book.insert({ISBN:"BK5",Title:"Mathematics",Author:"H P Patil",Publisher:"Sapna Publishers"});
> db.Book.insert({ISBN:"BK6",Title:"Soft Computing",Author:"M A Jayaram",Publisher:"Springer Publishers"});
> db.Book.insert({ISBN:"BK7",Title:"Artificial Intelligence",Author:"Agarwal",Publisher:"Springer Publishers"});
> db.Book.insert({ISBN:"BK8",Title:"Cloud Computing",Author:"Jeevan R S",Publisher:"Springer Publishers"});
> db.Book.insert({ISBN:"BK9",Title:"Operations Research",Author:"Agarwal",Publisher:"Springer Publishers"});
> db.Book.insert({ISBN:"BK10",Title:"Software Testing",Author:"Kirankumar T M",Publisher:"Springer Publishers"});
```

To insert data values in to a collection “Student”

```
> db.Student.insert({Usn:"MCA1",StudentName:"Akash",Class:"I Semester"});
> db.Student.insert({Usn:"MCA2",StudentName:"Anusha",Class:"I Semester"});
> db.Student.insert({Usn:"MCA3",StudentName:"Ashwini",Class:"II Semester"});
> db.Student.insert({Usn:"MCA4",StudentName:"Aishwarya",Class:"II Semester"});
> db.Student.insert({Usn:"MCA5",StudentName:"Bhagya",Class:"II Semester"});
> db.Student.insert({Usn:"MCA6",StudentName:"Bhaskar",Class:"III Semester"});
> db.Student.insert({Usn:"MCA7",StudentName:"Bhuvana",Class:"III Semester"});
> db.Student.insert({Usn:"MCA8",StudentName:"Kumar",Class:"III Semester"});
```

```
> db.Student.insert({Usn:"MCA9",StudentName:"KarunNair",Class:"I Semester"});
> db.Student.insert({Usn:"MCA10",StudentName:"Kavya",Class:"I Semester"});
```

To insert data values in to a collection “Borrow”

```
> db.Borrow.insert({ISBN:"BK1",Usn:"MCA1",BorrowedDate:"12-Jan-21"});
> db.Borrow.insert({ISBN:"BK2",Usn:"MCA1",BorrowedDate:"12-Jan-21"});
> db.Borrow.insert({ISBN:"BK3",Usn:"MCA2",BorrowedDate:"15-Jan-21"});
> db.Borrow.insert({ISBN:"BK4",Usn:"MCA3",BorrowedDate:"17-Jan-21"});
> db.Borrow.insert({ISBN:"BK5",Usn:"MCA4",BorrowedDate:"19-Jan-21"});
> db.Borrow.insert({ISBN:"BK6",Usn:"MCA5",BorrowedDate:"21-Jan-21"});
> db.Borrow.insert({ISBN:"BK7",Usn:"MCA5",BorrowedDate:"21-Jan-21"});
> db.Borrow.insert({ISBN:"BK8",Usn:"MCA5",BorrowedDate:"21-Jan-21"});
> db.Borrow.insert({ISBN:"BK9",Usn:"MCA6",BorrowedDate:"23-Jan-21"});
> db.Borrow.insert({ISBN:"BK10",Usn:"MCA7",BorrowedDate:"25-Jan-21"});
```

a) List the Book titles in which there is a particular keyword pattern “Computing”.

Step-1 : Use pattern matching concept in listing out book titles with a keyword pattern “Computing” from the collection “Book”

```
> db.Book.find({Title:/Compu/},{Title:1,Author:1,_id:0});
{ "Title" : "Soft Computing", "Author" : "M A Jayaram" }
{ "Title" : "Cloud Computing", "Author" : "Jeevan R S" }
```

b) Obtain the Names of students who have borrowed DBMS books.

Step-1 : Firstly, find the ISBN value for the book title “DBMS” from the collection “Book”

```
> db.Book.find({Title:"DBMS"},{ISBN:1,_id:0});
{ "ISBN" : "BK2" }
```

Step-2 : Then, find the USN value for the student from the collection “Borrow”

```
> db.Borrow.find({ISBN:"BK2"},{Usn:1,_id:0});
{ "Usn" : "MCA1" }
```

Step-3 : Then, finally find the student name for the USN from the collection Student.

```
> db.Student.find({Usn:"MCA1"},{StudentName:1,_id:0});
{ "StudentName" : "Akash" }
```

c) List the Author names who have authored more than 2 Books.

Step-1 : Firstly, List the Author names and number of books they authored from the collection “Book”

```
> db.Book.aggregate([{$group:{_id:"$Author",NoofBooks:{$count:{}}}}]);
{ "_id" : "Agarwal", "NoofBooks" : 3 }
{ "_id" : "Arun Shah", "NoofBooks" : 2 }
{ "_id" : "H P Patil", "NoofBooks" : 1 }
{ "_id" : "SK Jain", "NoofBooks" : 1 }
{ "_id" : "M A Jayaram", "NoofBooks" : 1 }
{ "_id" : "Jeevan R S", "NoofBooks" : 1 }
{ "_id" : "Kirankumar T M", "NoofBooks" : 1 }
```

Step-2 : Then, List the Author names who have authored more than 1 book from the collection “Book” by using additional condition $match (Having clause in SQL) on the grouped data items.

```
> db.Book.aggregate([{$group:{_id:"$Author",NoofBooks:{$count:{}}}},{$match:{NoofBooks:{$gte:2}}}]);
{ "_id" : "Arun Shah", "NoofBooks" : 2 }
{ "_id" : "Agarwal", "NoofBooks" : 3 }
```

d) List the Publisher names who have published more than 2 Books.

Step-1 : Firstly, List the Publisher names and number of books they published from the collection “Book”

```
> db.Book.aggregate([{$group:{_id:"$Publisher",NoofBooks:{$count:{}}}}]);
{ "_id" : "BPN Publishers", "NoofBooks" : 1 }
{ "_id" : "TATA McGraw", "NoofBooks" : 1 }
{ "_id" : "Springer Publishers", "NoofBooks" : 5 }
{ "_id" : "Sapna Publishers", "NoofBooks" : 3 }
```

Step-2 : Then, List the Publisher names who have published more than 2 books from the collection “Book” by using additional condition $match (Having clause in SQL) on the grouped data items.

```
> db.Book.aggregate([{$group:{_id:"$Publisher",NoofBooks:{$count:{}}}},{$match:{NoofBooks:{$gte:2}}}]);
{ "_id" : "Sapna Publishers", "NoofBooks" : 3 }
{ "_id" : "Springer Publishers", "NoofBooks" : 5 }
```

e) List the student names who have borrowed more than 1 Book.

Step-1 : Firstly, List the Students USN and number of books they borrowed from the collection “Borrow”

```
> db.Borrow.aggregate([{$group:{_id:"$Usn",NoofBooks:{$count:{}}}}]);
{ "_id" : "MCA3", "NoofBooks" : 1 }
{ "_id" : "MCA4", "NoofBooks" : 1 }
{ "_id" : "MCA5", "NoofBooks" : 3 }
{ "_id" : "MCA6", "NoofBooks" : 1 }
{ "_id" : "MCA7", "NoofBooks" : 1 }
{ "_id" : "MCA1", "NoofBooks" : 2 }
{ "_id" : "MCA2", "NoofBooks" : 1 }
```

Step-2 : Then, List the Students USN who have borrowed more than 2 books from the collection “Borrow” by using additional condition $match (Having clause in SQL) on the grouped data items.

```
> db.Borrow.aggregate([{$group:{_id:"$Usn",NoofBooks:{$count:{}}}},{$match:{NoofBooks:{$gte:2}}}]);
{ "_id" : "MCA5", "NoofBooks" : 3 }
{ "_id" : "MCA1", "NoofBooks" : 2 }
```

Step-3 : Now, List the names of students for the USNs from the collection “Student”

```
> db.Student.find({$or:[{Usn:"MCA1"},{Usn:"MCA5"}]},{Usn:1,StudentName:1,_id:0});
{ "Usn" : "MCA1", "StudentName" : "Akash" }
{ "Usn" : "MCA5", "StudentName" : "Bhagya" }
```