Unit-2

Introduction to MongoDB

Dr. Selva kumar S
Assistant Professor
Dept. of CSE, BMSCE

Learning Objectives and Learning Outcomes

Learning Objectives	Learning Outcomes
Introduction to MongoDB 1. To study the features of MongoDB. 2. To learn how to perform CRUD operations. 3. To study aggregation. 4. To study the MapReduce Framework. 5. To import from and export to CSV format.	a) To comprehend the reasons behind the popularity of NoSQL database. b) To be able to perform CRUD operations. c) To comprehend MapReduce framework. d) To understand the aggregation. e) To be able to successfully import from and export to CSV.

Agenda

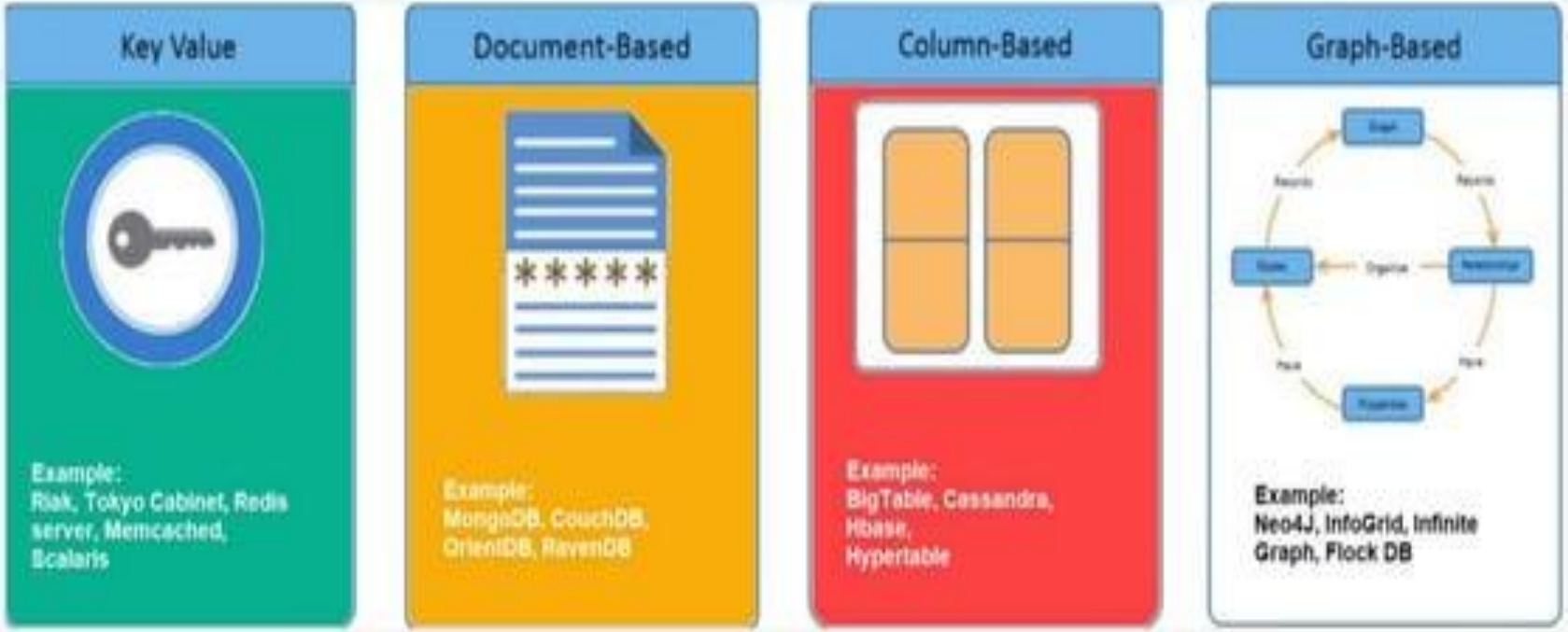
- NoSQL: Types of NoSQL databases
- CAP Theorem
- Terms used in RDBMS and MongoDB
- MongoDB Query Language: CRUD
- Finding documents based on search criteria
- Dealing with Null values
- Count, Limit, Sort Skip, Aggregate Functions

What is NoSQL Database?

- The term "NoSQL database" refer to any "Non-relational" or "Not only SQL" databases provides a mechanism for storage and retrieval of data in a format other than tabular relations model used in relational databases.
- NoSQL database doesn't use tables for storing data.
- It is generally used to store big data and real-time web applications.
- Flexible schema i.e., no predefined or rigid schema.
- It avoids joins and is easy to scale.

Types of NoSQL Database

NoSQL Databases are mainly categorized into four types: Key-value pair, Column-oriented, Graph-based & Document-oriented.



SQL Vs NoSQL

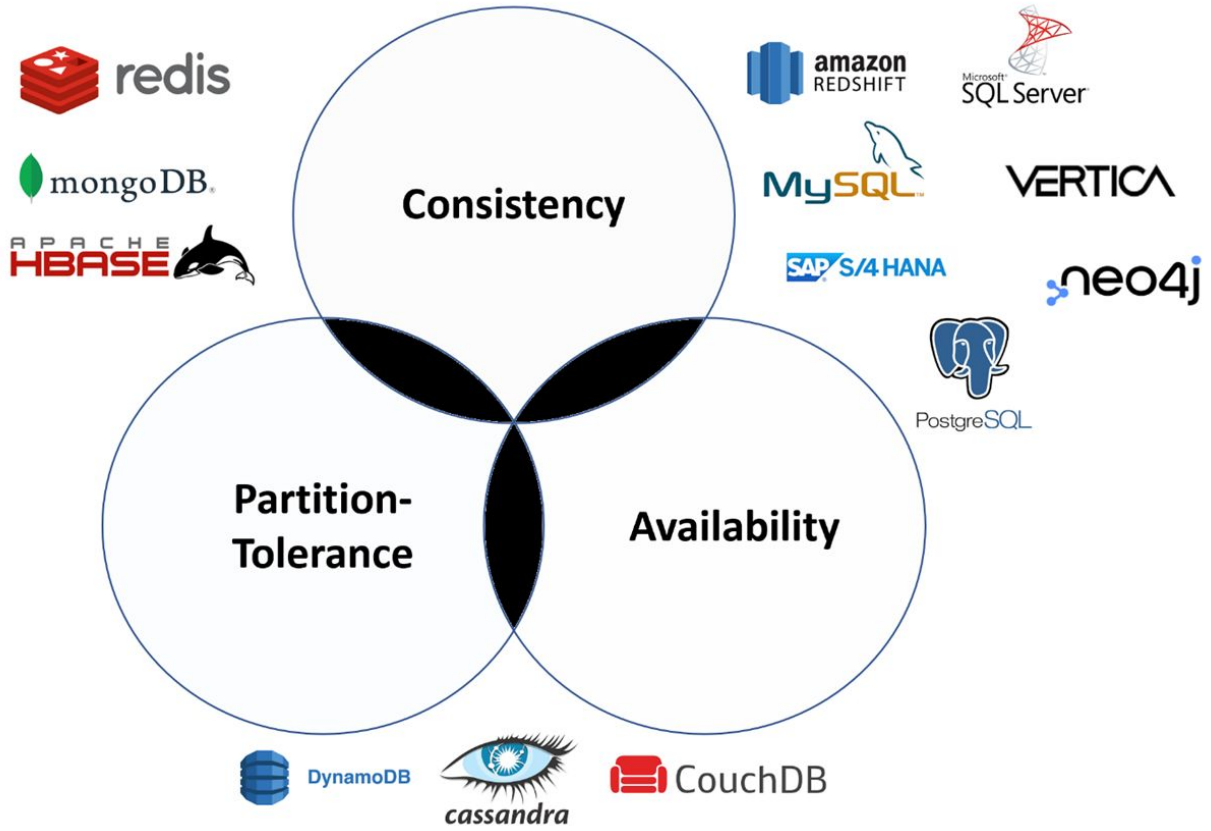
SQL	NoSQL
SQL databases are primarily called RDBMS or Relational Databases.	NoSQL databases are primarily called as Non-relational or distributed database.
SQL databases are table-based databases.	NoSQL databases can be document based, key-value pairs, graph databases.
Vertical Scalability	Horizontal Scalability
Fixed or Predefined schema.	Flexible schema.
Not suitable for hierarchical data storage.	Suitable for hierarchical data storage.
Oracle, MySQL, Microsoft SQL Server, and PostgreSQL.	Document: MongoDB and CouchDB, Key-value: Redis and DynamoDB, Column-based: Cassandra and HBase, Graph: Neo4j and Amazon Neptune.

When to use NoSQL?

1. When a huge amount of data needs to be stored and retrieved.
2. The relationship between the data you store is not that important
3. The data changes over time and is not structured.
4. Constraints and Joins support is not required at database level.
5. The data is growing continuously, and you need to scale the database regularly to handle the data.

CAP Theorem

The CAP theorem says that a distributed system can deliver only two of three desired characteristics:
consistency, ***availability*** and ***partition tolerance***



What is MongoDB?

- MongoDB is:
  - Cross-Platform.
  - Open Source.
  - Non-relational.
  - Distributed.
  - NoSQL.
  - Document-oriented data store.

Why MongoDB?


Ad-hoc queries
Schema-less database
Document-oriented
Indexing
Replication
Aggregation
GridFS
Sharding
High performance

Features of MongoDB

- **Indexing**: Efficient search & data processing in very less time.
- **Scalability**: MongoDB scales horizontally using sharding (partitioning data across various servers). Also, new machines can be added to a running database.
- **Replication & High Availability:** Multiple copies of data is replicated on different servers which protects the database from hardware failures. If one server goes down, the data can be retrieved easily from other active servers which also had the data stored on them.
- **Load balancing:** It has an automatic load balancing configuration because of data placed in shards.
- **Aggregation:** Aggregation operations process data records and return the computed results. It is like the GROUPBY clause in SQL. A few aggregation expressions are sum, avg, min, max, etc.
- **Languages supported:** The list of supported languages includes Node.js, C, C++, C#, Go, Java, Perl, PHP, Python, Ruby, Rust, Scala, and Swift.

Using JSON

- JSON, or JavaScript Object Notation, is a human-readable data interchange format.

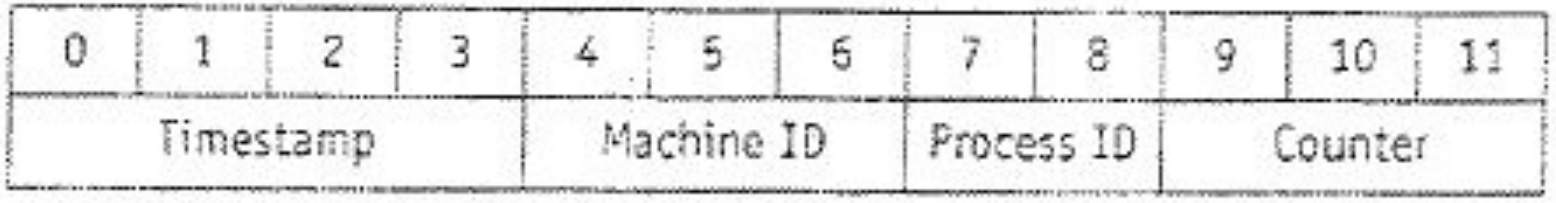
- JSON *objects* are associative containers, wherein a string key is mapped to a value (which can be a number, string, boolean, array, an empty value — null, or even another object).

- **BSON**, or Binary JSON, is the data format that MongoDB uses to organize and store d

```
{ _id:
Name :  "Sujeet",
Age :  22,
Address :  { {street: "H1 Block",
             City: "Sultanpuri",
             State: "Delhi"
             Zip: "110086"
             Country: "India"} }
}
```

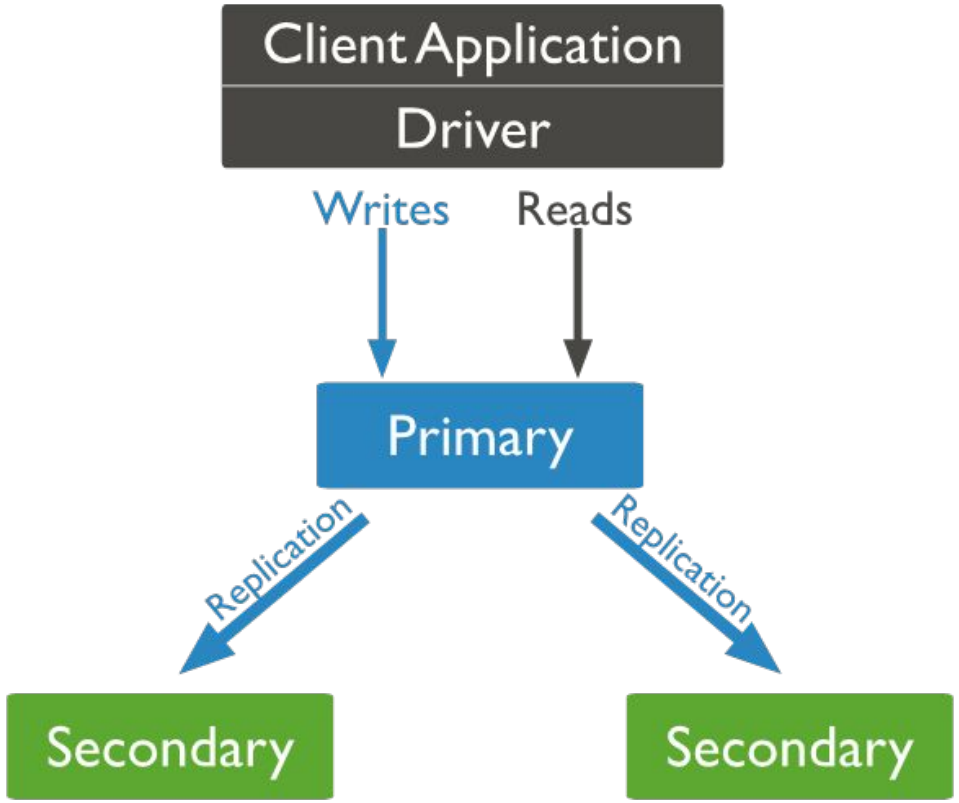
Creating or Generating a Unique Key

- Each JSON document should have a unique identifier.
- It is similar to the primary key of relational databases.
- This facilitates search for documents based on the unique identifier.
- ObjectIds use 12 bytes of storage, which gives them a string representation that is 24 hexadecimal digits: 2 digits for each byte.



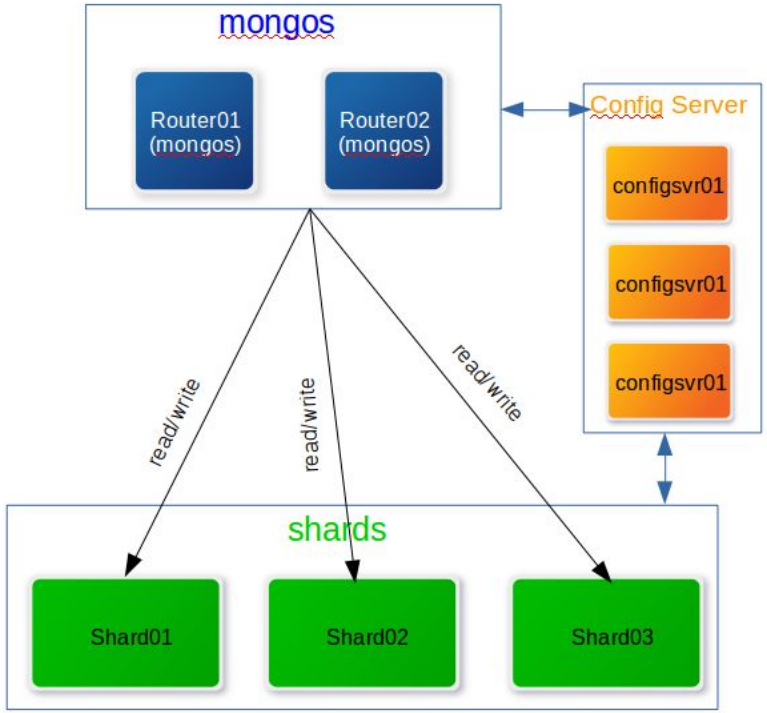
Replication

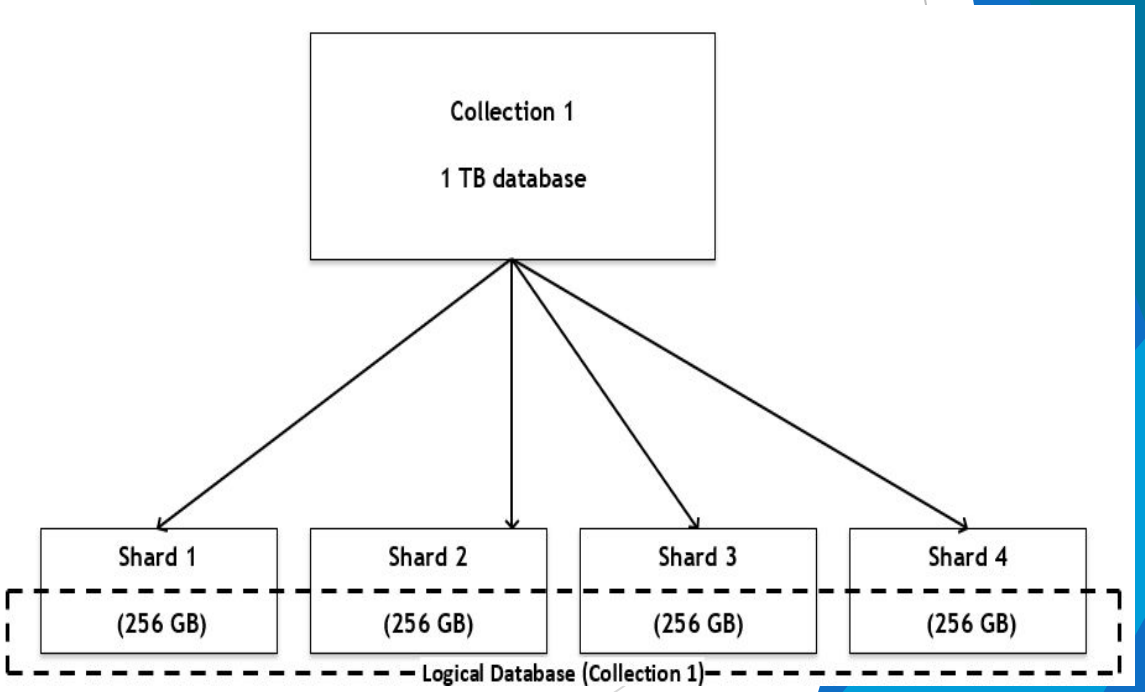
- MongoDB provides data redundancy and high availability.
- It helps to recover from hardware failure and service interruption.



Sharding

- Sharding is a method for distributing or partitioning data across multiple machines.
- Horizontal scaling, also known as scale-out, refers to adding machines to share the data set and load.





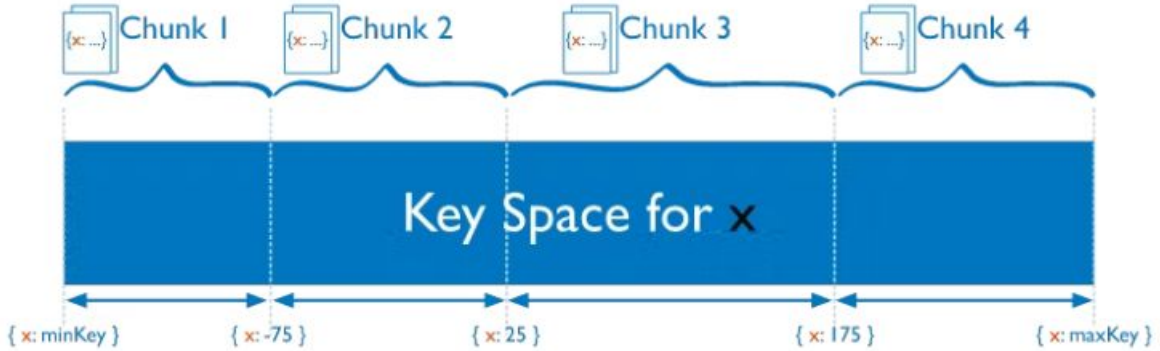
Advantages of Sharding

- Increased read/write throughput.
- Increased storage capacity.
- Data Locality.

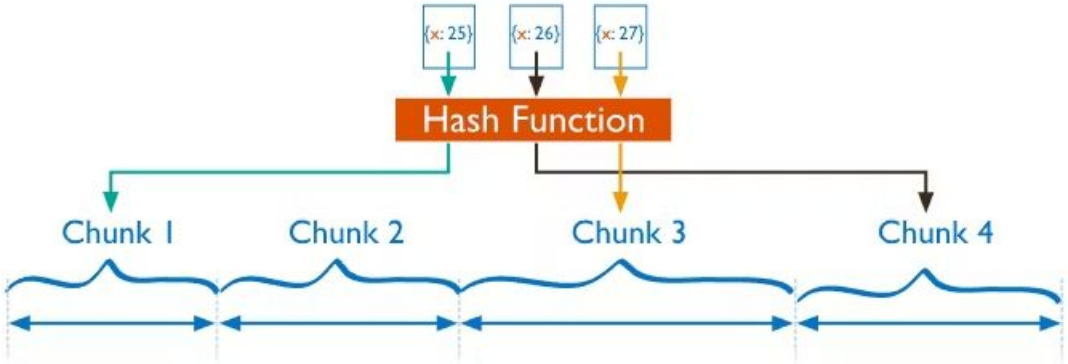
Sharding Strategy

MongoDB supports two sharding strategies for distributing data across sharded clusters:

- Ranged Sharding



- Hashed Sharding



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.

Who's is using MongoDB?

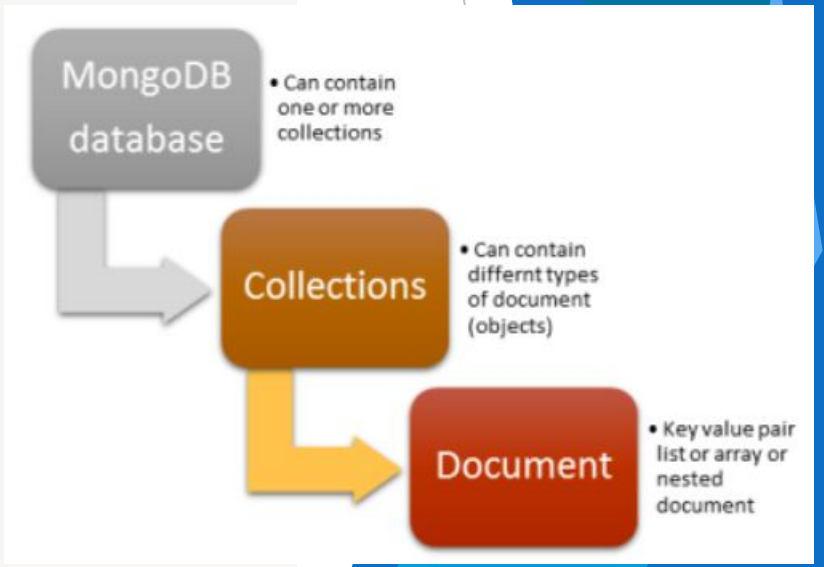
MongoDB has been adopted as backend software by a few major websites and services including Toyota, Cisco, Shutterfly, Adobe, Ericsson, Craigslist, eBay, and Foursquare.

MongoDB Terms

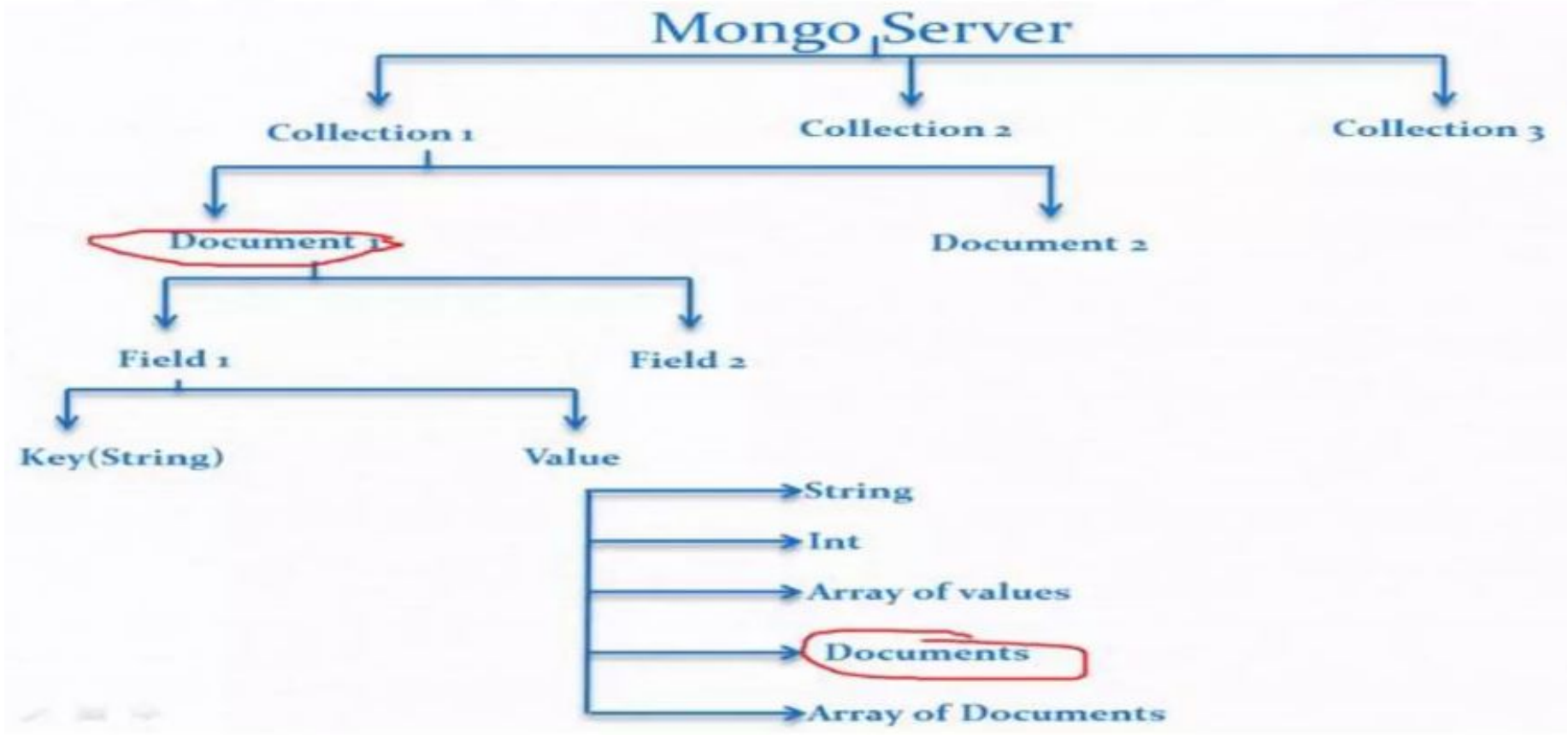
- **Databases:** A database stores one or more collections of documents.
- **Collections:** MongoDB stores documents in collections. Collections are analogous to tables in relational databases.
- **Documents:** MongoDB stores data records as BSON documents & similar to JSON objects. It is analogous to row in relational databases. These are composed of field-and-value pairs and have the following structure:

```
{
   field1: value1,
   field2: value2,
   field3: value3,
   ...
   fieldN: valueN
}
```

- **Fields Names:** These are strings analogous to column in relational databases.



Data Storage Mechanism



Example

```
{
  name: "sue",                          ← field: value
  age: 26,                              ← field: value
  status: "A",                          ← field: value
  groups: [ "news", "sports" ]          ← field: value
}
```

SQL Vs MongoDB Terms

SQL Term	MongoDB Term
Database	Database
Table	Collection
Index	Index
Row	BSON document
Column	BSON field
Primary Key	_id field
Group by	Aggregation
Join	Embedding and Linking

MongoDB Installation

- MongoDB is available in two server editions: *Community* and *Enterprise*.
- Community edition is free & Enterprise edition is a paid version and has some advanced features.
- Installation links:

  Install MongoDB — MongoDB Manual

  Community: Try MongoDB Community Edition | MongoDB

  Enterprise: Try MongoDB Enterprise Advanced | MongoDB

Create Database

The **use Database_name** command makes a new database in the system if it does not exist, if the database exists it uses that database.

For example, use mydb.
Now your database is ready of name mydb.

```
>use mydb
switched to db mydb
```

List all databases

To list down all the databases, use the command below:
show dbs

This command lists down all the databases and their size on the disk.

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

mydb database is not created until you save a document in it.

Create Collection

To create collection, use the command below:
db.createCollection(name, options)

name is name of collection to be created. **Options** is a document and is used to specify configuration of collection such as capped, autoIndexId, size.

```
>db.createCollection('students')
{ "ok" : 1 }
```

Field	Type	Description
capped	Boolean	(Optional) If true, enables a capped collection. Capped collection is a fixed size collection that automatically overwrites its oldest entries when it reaches its maximum size. **If you specify true, you need to specify size parameter also.**
autoIndexId	Boolean	(Optional) If true, automatically create index on _id field.s Default value is false.
size	number	(Optional) Specifies a maximum size in bytes for a capped collection. **If capped is true, then you need to specify this field also.**
max	number	(Optional) Specifies the maximum number of documents allowed in the capped collection.

List all Collections

To see all collections in a database, use the command below:
show collections

```
>show collections
hi
students
```

Drop Database & Collection

- Use the following command to delete a database:

db.dropDatabase()

- Use the following command to delete a collection:

db.collection_name.drop()

```
> db.dropDatabase()
{ "ok" : 1 }
>
```

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

Create Documents

Two methods insert or create documents in collection:

1. db.collection_name.insertOne()
2. db.collection_name.insertMany()

```
>db.students.insertOne({name:"Ved Pandey",age:22,gender:`Male`,course:`BCA`})
{
    "acknowledged" : true,
    "insertedId" : ObjectId("6395da769e1a378e0e4a9f72")
}

>db.students.insertMany([{name:"Neha Singh",age:21,gender:`Female`,course:`B.Com`},{name:"Ankita Tripathi",age:22,gender:`Female`,course:`MCA`,email:`nehagmail.com`}])
{
    "acknowledged" : true,
    "insertedIds" : [
        ObjectId("6395daf9f05532c3d92c7f51"),
        ObjectId("6395daf9f05532c3d92c7f52")
    ]
}
```

Retrieving Documents

- Use the below command to retrieve documents:

db.collection_name.find()

- For a proper formatted output, use the below command: pretty() with find().

db.collection_name.find().pretty()

```
>db.students.find()
{ "_id" : ObjectId("6395d963f07f724a7192d41a"), "name" : "Surya Gupta", "age" : 23, "gender" : "Male", "course" : "B.Tech" }
{ "_id" : ObjectId("6395da769e1a378e0e4a9f72"), "name" : "Ved Pandey", "age" : 22, "gender" : "Male", "course" : "BCA" }
{ "_id" : ObjectId("6395daf9f05532c3d92c7f51"), "name" : "Neha Singh", "age" : 21, "gender" : "Female", "course" : "B.Com" }
{ "_id" : ObjectId("6395daf9f05532c3d92c7f52"), "name" : "Ankita Tripathi", "age" : 22, "gender" : "Female", "course" : "MCA", "email" : "nehagmail.com" }
```

Retrieving Documents with Filters or Criteria

```
>db.students.find({gender:"Male"}).pretty()
{
    "_id" : ObjectId("6395d963f07f724a7192d41a"),
    "name" : "Surya Gupta",
    "age" : 23,
    "gender" : "Male",
    "course" : "B.Tech"
}
{
    "_id" : ObjectId("6395da769e1a378e0e4a9f72"),
    "name" : "Ved Pandey",
    "age" : 22,
    "gender" : "Male",
    "course" : "BCA"
}
```

```
>db.students.find({gender:"Male"},{_id:0,name:1,age:1}).pretty()
{ "name" : "Surya Gupta", "age" : 23 }
{ "name" : "Ved Pandey", "age" : 22 }
```

```
>db.students.find({},{_id:0,name:1,email:true}).pretty()
{ "name" : "Surya Gupta" }
{ "name" : "Ved Pandey" }
{ "name" : "Neha Singh" }
{ "name" : "Ankita Tripathi", "email" : "neha@gmail.com" }
```

Update Documents

- There are 2 ways to update documents:

1. db.collection_name.updateOne()
2. db.collection_name.updateMany()

```
>>> db.students.updateOne({'name':'Ankita Tripathi'},{$set:{'email':'ankita123@gmail.com'}})
{
  acknowledged: true,
  insertedId: null,
  matchedCount: 1,
  modifiedCount: 1,
  upsertedCount: 0
}
```

```
>>> db.students.find({email:"ankita123@gmail.com"}).pretty()
[
  {
    _id: ObjectId("6395ec8a39c0e096224372ee"),
    name: 'Ankita Tripathi',
    age: 24,
    course: 'M.Tech',
    email: 'ankita123@gmail.com'
  }
]
```

Update Documents

```
>>> db.students.updateMany({age:{$eq:24}},{$set:{age:23}})
{
  acknowledged: true,
  insertedId: null,
  matchedCount: 1,
  modifiedCount: 1,
  upsertedCount: 0
}
```

```
>>> db.students.find({email:"ankita123@gmail.com"}).pretty()
[
  {
    _id: ObjectId("6395ec8a39c0e096224372ee"),
    name: 'Ankita Tripathi',
    age: 23,
    course: 'M.Tech',
    email: 'ankita123@gmail.com'
  }
]
```

Delete Documents

- There are 2 ways to delete documents:

1. db.collection_name.deleteOne()
2. db.collection_name.deleteMany()

```
>db.students.deleteOne({age:23})
{ "acknowledged" : true, "deletedCount" : 1 }
```

```
>db.students.deleteMany({age:22})
{ "acknowledged" : true, "deletedCount" : 3 }

>db.students.find().pretty()
{
    "_id" : ObjectId("6395daf9f05532c3d92c7f51"),
    "name" : "Neha Singh",
    "age" : 21,
    "gender" : "Female",
    "course" : "B.Com"
}
{
    "_id" : ObjectId("6395eacc3d031007a5f010b8"),
    "name" : "Neha Singh",
    "age" : 21,
    "gender" : "Female",
    "course" : "B.Com"
}
```

	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})

Collections

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”.

```
db.createCollection(“Person”);
```

Collections

To drop a collection by the name “food”.

```
db.food.drop();
```

Insert Method

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"});

Update Method

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).

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

Adding new field - update method

To add a new field to an existing document in MongoDB, you can use the updateOne() or updateMany() method along with the $set operator.

```
db.collection.updateOne(
  // Filter to select the document(s) you want to update
  { <filter> },
  // Update operation
  {
    // $set operator to add a new field
    $set: {
      newField: "value" // Specify the new field and its value
    }
  }
);

db.users.updateOne(
  { name: "John" }, // Filter documents where name is "John"
  { $set: { age: 30 } } // Add the new field "age" with value 30
);
```

Removing an existing field

To remove an existing field from an existing document in MongoDB, you can use the updateOne() or updateMany() method along with the $unset operator.

```
db.collection.updateOne(
  // Filter to select the document(s) you want to update
  { <filter> },

  // Update operation
  {
    // $unset operator to remove a field
    $unset: {
      fieldName: "" // Specify the name of the field you want to remove
    }
  }
);

db.users.updateOne(
  { name: "John" }, // Filter documents where name is "John"
  { $unset: { age: "" } } // Remove the "age" field
);
```

Save Method

Save() method is used to update an existing document in a collection or insert a new document if it does not already exist.

```
// Update an existing document with _id "123abc"
db.collection.save({
  _id: "123abc",
  name: "Updated Name",
  age: 30,
  status: "active"
});

// Insert a new document if _id "456def" doesn't already exist
db.collection.save({
  _id: "456def",
  name: "New Document",
  age: 25,
  status: "inactive"
});
```

Remove Method

Remove() method is used to delete documents from a collection that match a specified query. It can remove either a single document or multiple documents depending on the criteria provided.

```
db.collection.remove(
   <query>,
   {
     justOne: <boolean>,
     writeConcern: <document>
   }
)
```

```
db.users.remove({ name: "John" });
```

```
db.users.remove({ name: "John" }, { justOne: true });
```

Find Method

To search for documents from the “Students” collection based on certain search criteria.

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

Find Method

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

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

Find Method

To find those documents where the Grade is set to ‘VII’

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

Find Method

To find those documents from the Students collection where the Hobbies is set to either ‘Chess’ or is set to ‘Skating’.

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

Find Method

To find documents from the Students collection where the StudName begins
with “M”.

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

Find Method

To find documents from the Students collection where the StudName has an “e”
in any position.

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

Find Method

To find the number of documents in the Students collection.

```
db.Students.count();
```

Find Method

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

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

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:{Seq:'VII'}}).pretty();
```

RDBMS Equivalent:

```
Select *
        From Students
                Where Grade like 'VII';
```

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

Act:

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

Outcome:

```
> db.Students.find({Grade:{$ne:"VII"}}).pretty();
{
        "_id" : ObjectId("5464849889ac1ab07d489b7f"),
        "StudName" : "Vamsi Bapat",
        "Grade" : "VI"
}
```

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

RDBMS Equivalent:

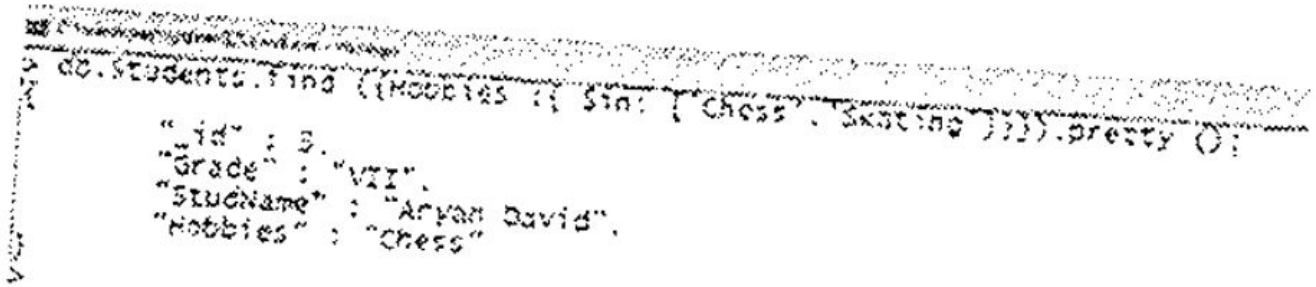
```
Select *
        From Students
                Where Grade <> 'VII';
```

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:



RDBMS Equivalent:

```
Select *
        From Students
                Where Hobbies in ('Chess', 'Skating');
```

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:

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
----- --------------------------- ----- ---------------------------
      Michelle Jacintha           VIII  Internet surfing
      Mabel Mathews               VIII  Baseball
      Hemesh Gibbs                VIII  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';
```

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

Act:

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

RDBMS Equivalent:

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

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

Act:

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

Outcome:

```
> db.Students.find({StudName:/sS/}).pretty()
{
        "_id" : 4,
        "Grade" : "VII",
        "StudName" : "Mersch Gioba",
        "Hobbies" : "Graffiti"
}

        "_id" : 2,
        "StudName" : "Mabel Mathews",
        "Grade" : "VII",
        "Hobbies" : "Baseball"
}
```

RDBMS Equivalent:

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

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();
```

RDBMS Equivalent:

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

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

Act:

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

Outcome:

```
> 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';
```

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%';
```

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

Act:

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

Outcome:

```
> db.Students.find({Grade:{$ne:"VII"}}).pretty();
{
        "_id" : ObjectId("5464849889ac1ab07d489b7f"),
        "StudName" : "Vamsi Bapat",
        "Grade" : "VI"
}
```

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

RDBMS Equivalent:

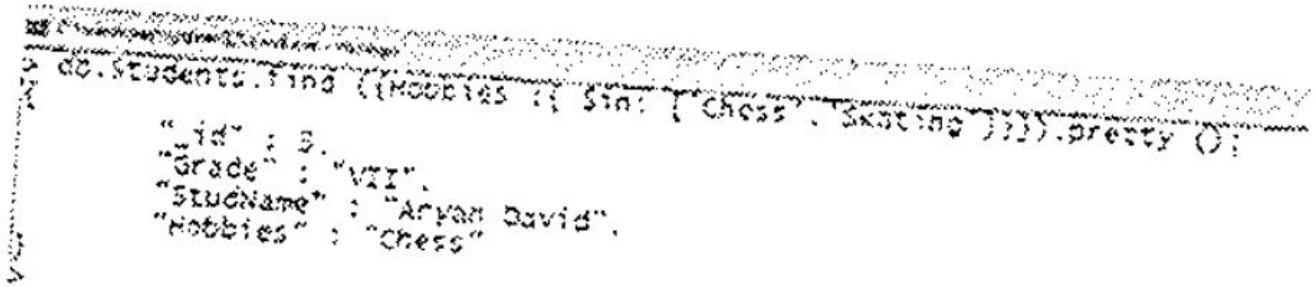
```
Select *
        From Students
                Where Grade <> 'VII';
```

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:



RDBMS Equivalent:

```
Select *
        From Students
                Where Hobbies in ('Chess', 'Skating');
```

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:

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
----- --------------------------- ----- ---------------------------
      Michelle Jacintha           VIII  Internet surfing
      Mabel Mathews               VIII  Baseball
      Hemesh Gibbs                VIII  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';
```

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

Act:

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

RDBMS Equivalent:

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

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

Act:

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

Outcome:

```
> db.Students.find({StudName:/sS/}).pretty()
{
        "_id" : 4,
        "Grade" : "VII",
        "StudName" : "Mersch Gioba",
        "Hobbies" : "Graffiti"
}

        "_id" : 2,
        "StudName" : "Mabel Mathews",
        "Grade" : "VII",
        "Hobbies" : "Baseball"
}
```

RDBMS Equivalent:

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

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();
```

RDBMS Equivalent:

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

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

Act:

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

Outcome:

```
> 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';
```

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

Act:

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

Outcome:

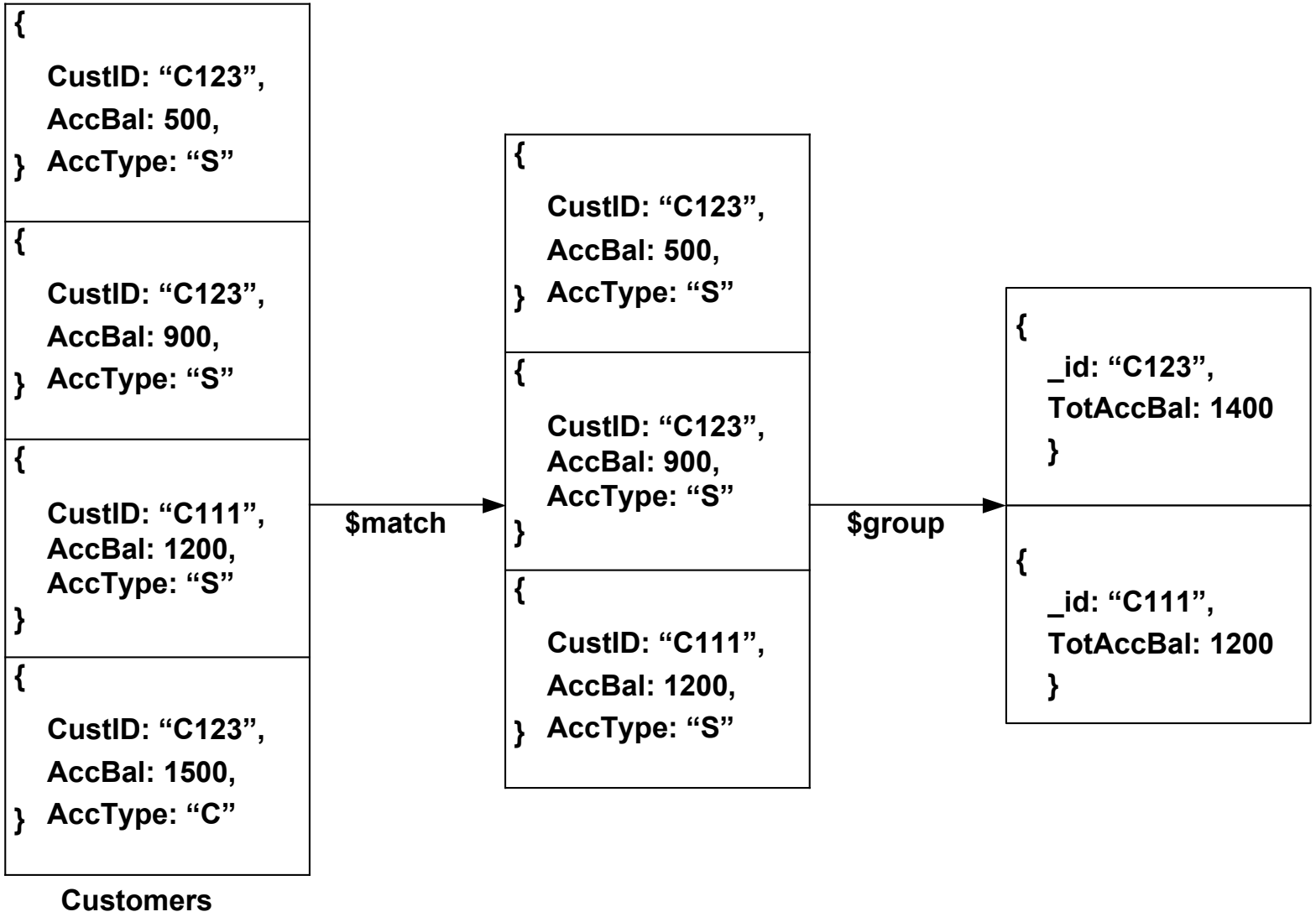
```
> db.Students.find({StudName:{$regex:"^M"}}).pretty();
{
        "_id" : 1,
        "StudName" : "Michelle Jacintha",
        "Grade" : "VII",
        "Hobbies" : "Internet Surfing"
}
{
        "_id" : 2,
        "StudName" : "Mabel Mathews",
        "Grade" : "VII",
        "Hobbies" : "Baseball"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

RDBMS Equivalent:

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

Aggregate Function

Aggregate Function



Aggregate Function

First filter on “AccType:S” and then group it on “CustID” and then 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 } }});
```

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()

Exercise

SELECT name, age FROM employees;

db.employees.find({}, { name: 1, age: 1, _id: 0 });

Exercise

SELECT * FROM employees WHERE age > 30;

db.employees.find({ age: { $gt: 30 } });

Exercise

SELECT * FROM employees ORDER BY age DESC;

db.employees.find().sort({ age: -1 });

Exercise

SELECT * FROM employees LIMIT 10;

db.employees.find().limit(10);

Exercise

SELECT department, COUNT(*) AS total_employees FROM employees GROUP BY department;

```
db.employees.aggregate([ { $group: { _id: "$department", total_employees: { $sum: 1 } } } ]);
```

Import data from a CSV file

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”.

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

Export data to a CSV file

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.

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

Assignment-1

You are tasked with developing a blogging platform where users can create posts, comment on posts, and like posts. Each post can have multiple comments and likes. Design a MongoDB schema to handle this scenario efficiently.

Assignment-2

You are developing a social media analytics platform that tracks user engagement metrics for posts on various social media platforms. Each post contains information such as the platform it was posted on, the number of likes, shares, and comments it received, as well as the timestamp of the post. Design a MongoDB schema to efficiently store and query this data.

How would you retrieve posts made on Facebook with more than 200 likes and sort them in descending order based on the number

Assignment-3

You are developing an e-commerce platform where users can browse and purchase products. Each product has a unique identifier, a name, a category, a price, and available quantity. Additionally, users can add products to their cart and place orders. Design a MongoDB schema to efficiently handle product information, user carts, and orders.

Query to

1. Retrieve All Products.
2. Retrieve Products in a Specific Category (e.g., Electronics).
3. Retrieve Products with Quantity Greater Than 0.
4. Retrieve Products Sorted by Price in Ascending Order.
5. Retrieve Products with Price Less Than or Equal to $100.
6. Retrieve Products Added to a User's Cart (User with ID "789ghi...")
7. Retrieve Orders Placed by a User (User with ID "123abc...")
8. Retrieve Total Price of Orders Placed by a User (User with ID "123abc...")

Assignment-4

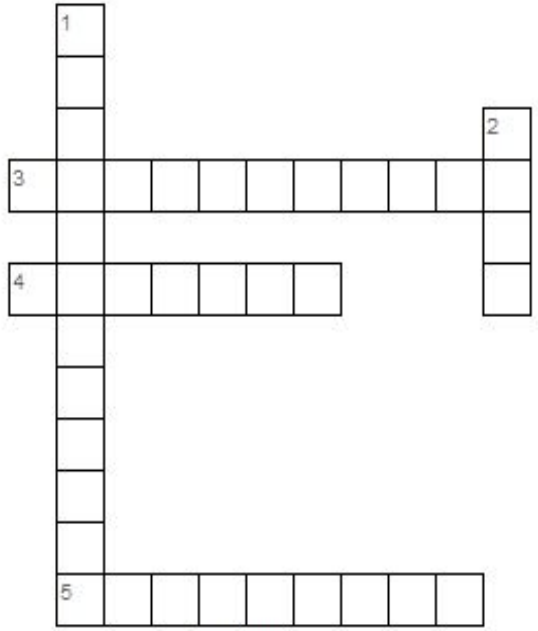
Additional Aggregation queries based on Assignment-3 design:

1 .Calculate Total Number of Products in Each Category.
2. Calculate Total Price of Products in Each Category.
3. Find Average Price of Products.
4. Find Products with Quantity Less Than 10.
5. Sort Products by Price in Descending Order.
6. Calculate Total Price of Orders Placed by Each User.
7. Find Users with the Highest Total Price of Orders.
8. Find Average Total Price of Orders.

Crossword

MongoDB

Basic puzzle on MongoDB



Across

3 MongoDB database stores its data in .....

4 MongoDB uses ---------- schemas.

5 A collection holds one or more -- ----------.

Down

1 MongoDB uses ---------- files.

2 MongoDB uses ---------, a binary object format similar to, but more expensive than JSON.

Further Readings

http://www.mongodb.org/

https://university.mongodb.com/

http://www.tutorialspoint.com/mongodb/

Thank you