

# Big Data Analytics (3161607)

Prepared By Karuna Patel

IT Department

Sarvajanik College of Engineering and Technology

Surat

# MongoDB

# OUTLINES

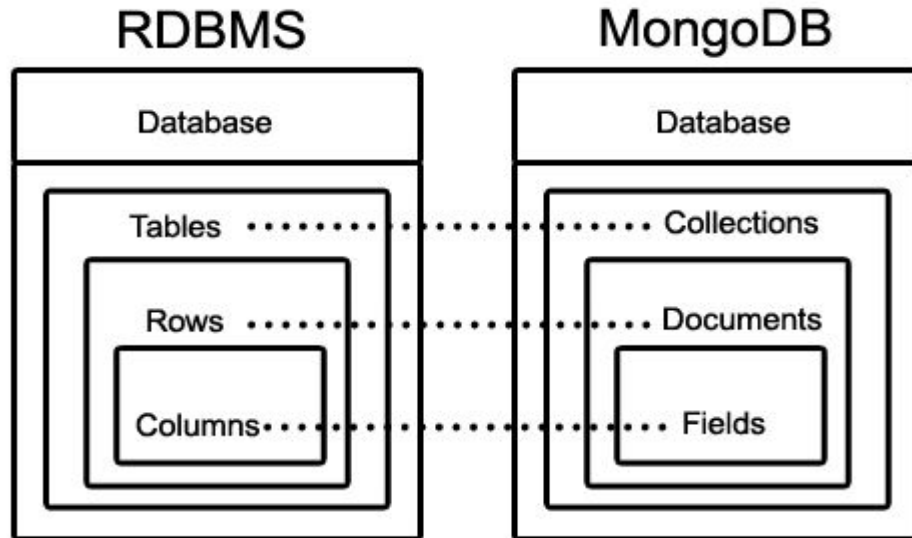
- What is MongoDB?
- Features of MongoDB
- Data types of MongoDB
- CRUD operations in MongoDB
- Aggregate function in MongoDB

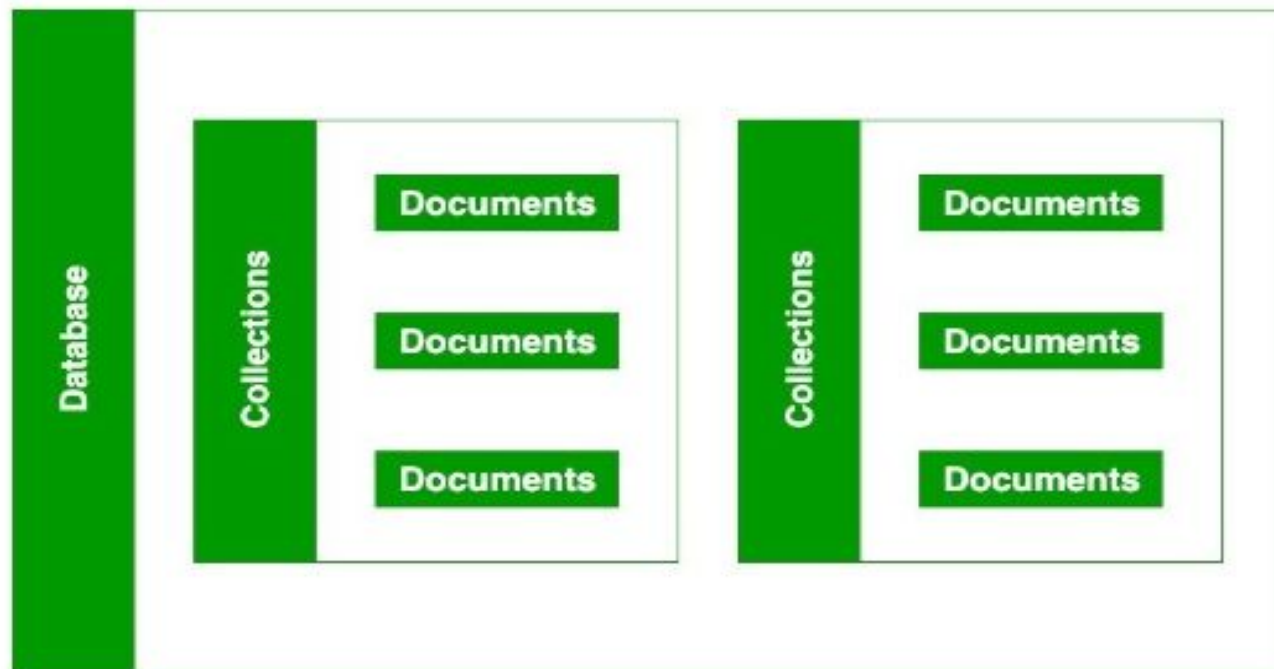
# What is MongoDB?

- MongoDB is an open-source document-oriented database that is designed to store a large scale of data and also allows you to work with that data very efficiently.
- It is categorized under the NoSQL (Not only SQL) database because the storage and retrieval of data in the MongoDB are not in the form of tables.
- Mongo DB is a document-oriented database. It is an open source product, developed and supported by a company named 10gen.

- The MongoDB database is developed and managed by MongoDB.Inc under SSPL(Server Side Public License) and initially released in February 2009.
- It also provides official driver support for all the popular languages like C, C++, C#, and .Net, Go, Java, Node.js, Perl, PHP, Python, Motor, Ruby, Scala, Swift, Mongoid.

- MongoDB is a scalable, open source, high performance, document-oriented database.





- The MongoDB database contains collections just like the MySQL database contains tables.
- You are allowed to create multiple databases and multiple collections.
- Now inside of the collection we have documents. These documents contain the data we want to store in the MongoDB



database and a single collection can contain multiple documents and you are schema-less means it is not necessary that one document is similar to another.

- The documents are created using the fields. Fields are key-value pairs in the documents, it is just like columns in the relation database.

- The value of the fields can be of any BSON data types like double, string, boolean, etc.

# Features of MongoDB

## 1.Schema-less Database:

- It is the great feature provided by the MongoDB.
- A Schema-less database means one collection can hold different types of documents in it.

- Or in other words, in the MongoDB database, a single collection can hold multiple documents and these documents may consist of the different numbers of fields, content and size.
- It is not necessary that the one document is similar to another document like in the relational databases. Due to this cool feature, MongoDB provides great flexibility to databases.

## 2.Document Oriented:

- In MongoDB, all the data stored in the documents instead of tables like in RDBMS.
- In these documents, the data is stored in fields(key-value pair) instead of rows and columns which make the data much more flexible in comparison to RDBMS. And each document contains its unique object id.

### SQL vs NoSQL

Table

Collection

Row

Document

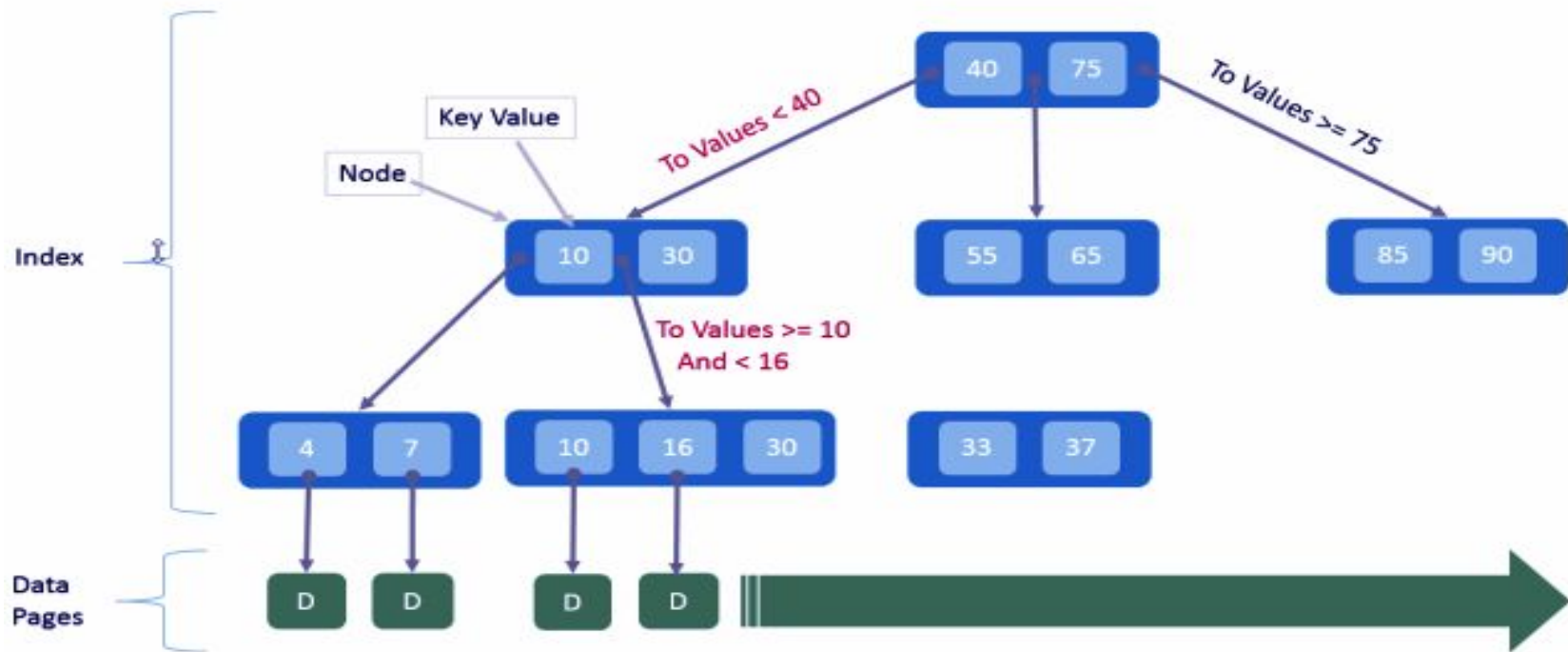
Column

Key

*MongoDB Features – Document Oriented*

## Indexing:

- In MongoDB database every field in the documents is indexed with primary and secondary indices this makes easier and takes less time to get or search data from the pool of the data. If the data is not indexed then database search each document with the specified query which takes lots of time and not so efficient.

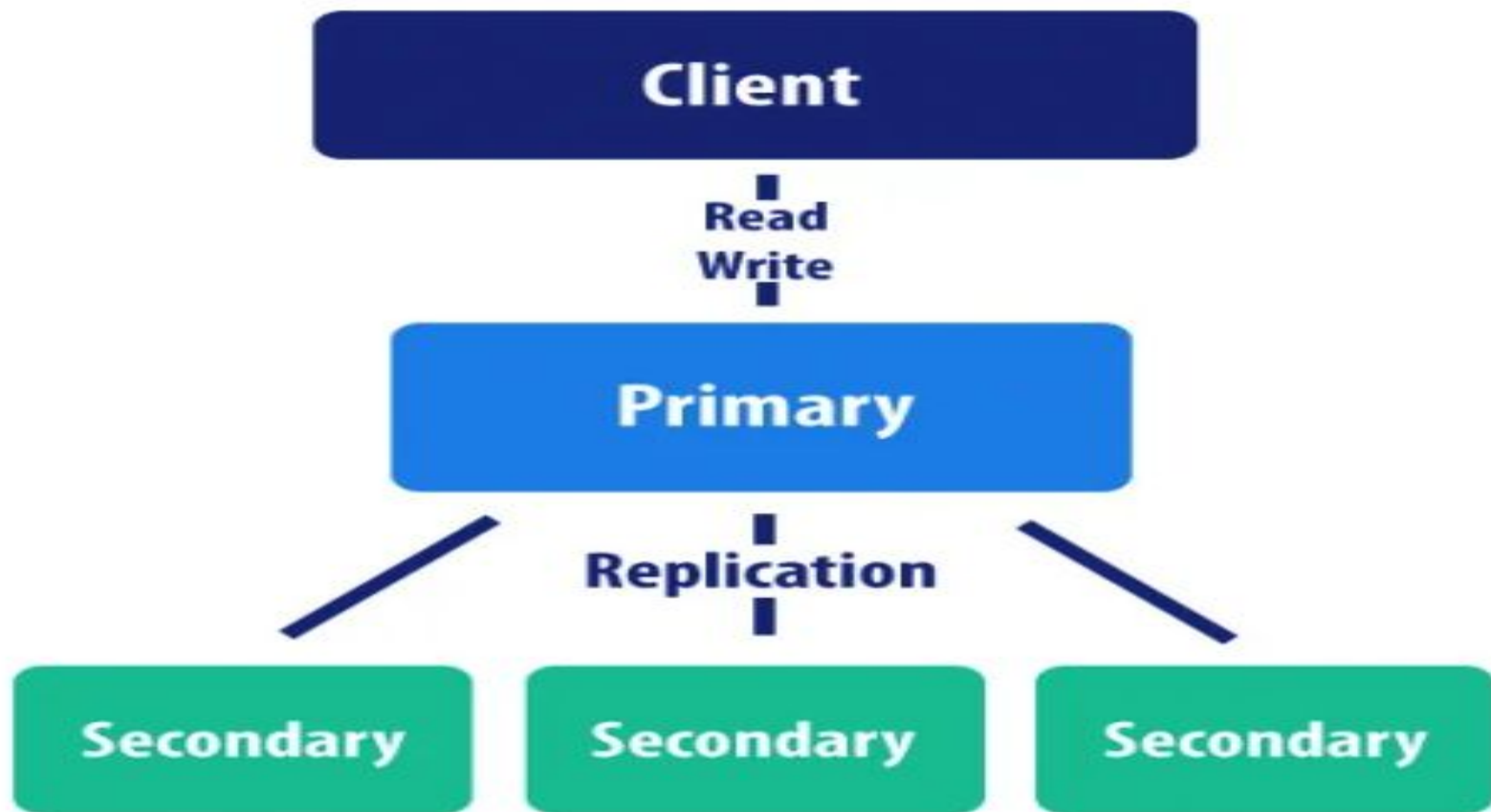


*MongoDB Features – Indexing*



## Replication:

MongoDB provides high availability and redundancy with the help of replication, it creates multiple copies of the data and sends these copies to a different server so that if one server fails, then the data is retrieved from another server.



## Aggregation:

It allows to perform operations on the grouped data and get a single result or computed result. It is similar to the SQL GROUPBY clause. It provides three different aggregations i.e, aggregation pipeline, map-reduce function, and single-purpose aggregation methods



# mongoDB

## Aggregation Framework



*MongoDB Features – Aggregation*

## High Performance:

The performance of MongoDB is very high and data persistence as compared to another database due to its features like scalability, indexing, replication, etc.

# Data types of MongoDB

MongoDB supports many datatypes.

- **Integer** : This type is used to store a numerical value. Integer can be 32 bit or 64 bit depending upon your server.
- **Boolean** : This type is used to store a boolean (true/ false) value.
- **String** : This is the most commonly used datatype to store the data.

- **Double** : This type is used to store floating point values.
- **Min/ Max keys** : This type is used to compare a value against the lowest and highest BSON elements.
- **Object** : This datatype is used for embedded documents.
- **Null** : This type is used to store a Null value.
- **Date** : This datatype is used to store the current date or time in UNIX time format. You can specify your own date time by

creating object of Date and passing day, month, year into it.

- **Regular expression:** This datatype is used to store regular expression.



# CRUD operations in MongoDB

CRUD operations *create*, *read*, *update*, and *delete* documents.

## 1.Create Operations:

Create or insert operations add new documents to a collection. If the collection does not currently exist, insert operations will create the collection.

MongoDB provides the following methods to insert documents into a collection:

Create Database:

Syntax: use database\_name

```
use studentinfo
```

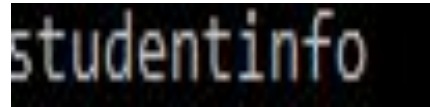
Output:

```
switched to db studentinfo
```

To see current database

Syntax:db

Output:

A screenshot of a terminal window with a black background and white text. The text 'studentinfo' is displayed, likely representing the output of a database command.

To view all database

Syntax: show dbs

Output:

```
admin      40.00 KiB
config     60.00 KiB
cscorner   80.00 KiB
local      72.00 KiB
st1        72.00 KiB
studentdata 72.00 KiB
test       8.00 KiB
```

To create collection:

**Syntax:** db.createCollection(collection\_name')

**Output:**

```
studentinfo> db.createCollection('student_record')  
{ ok: 1 }
```

To view all collections

**Syntax:** Show collections

```
show collectins
```

Output:

```
studentinfo> show collections
student_record
```

2.insert: insert one()

**Syntax:** db.collectionname.insertOne({id:1,'name':'Ranbir','subject':'Bigdata','city':'Mumbai','age':22})

```
db.student_record.insertOne({id:1,'name':'Ranbir','subject':'Bigdata','city':'Mumbai','age':22})
```

Output:

```
{  
  acknowledged: true,  
  insertedId: ObjectId("63f5b832bb3c56ccada887ae")  
}
```

## Output:

```
studentinfo> db.student_record.insertOne({id:4,'name':'Kiara','subject':'Algoritham','city':'Agra','age':25,'marks':79})
{
  acknowledged: true,
  insertedId: ObjectId("63f5be54bb3c56ccada887b1")
}
```



To insert Many rows: `db.collection.insertMany()`

Syntax:

```
db.student_record.insertMany([ {id:2,'name':'Kartik','subject':'Java',  
'city':'Surat','age':23}, {id:3,'name':'Ananya','subject':'Python',  
'city':Delhi,'age':22} ])
```

Output:

## Output:

```
studentinfo> db.student_record.insertMany([{id:2,'name':'Kartik','subject':'Java','city':'Surat','age':23},{id:3,'name':  
'Ananya','subject':'Pytho','city':'Delhi','age':22}])  
{  
  acknowledged: true,  
  insertedIds: {  
    '0': ObjectId("63f5ba9ebb3c56ccada887af"),  
    '1': ObjectId("63f5ba9ebb3c56ccada887b0")  
  }  
}
```

**2.Read operations** retrieve documents from a collection; i.e. query a collection for documents. MongoDB provides the following methods to read documents from a collection: To show all rows and records in collections

**Syntax:** `db.collection_name.find()`

**Example:** `db. student_record.find()`

```
studentinfo> db.student_record.find()  
[  
  {  
    _id: ObjectId("63f5b832bb3c56ccada887ae"),  
    id: 1,  
    name: 'Ranbir',  
    subject: 'Bigdata',  
    city: 'Mumbai',  
    age: 22  
  },  
  {  
    _id: ObjectId("63f5ba9ebb3c56ccada887af"),  
    id: 2,  
    name: 'Kartik',  
    subject: 'Java',  
    city: 'Surat',  
    age: 23  
  },  
  {  
    _id: ObjectId("63f5ba9ebb3c56ccada887b0"),  
    id: 3,  
    name: 'Ananya',  
    subject: 'Pytho',  
    city: 'Delhi',  
    age: 22  
  },  
  {  
    _id: ObjectId("63f5be54bb3c56ccada887b1"),  
    id: 4,  
    name: 'Kiara',  
    subject: 'Algoritham',  
    city: 'Agra',  
    age: 25,  
    marks: 79  
  }  
]
```

To view data in pretty form

**Syntax:** `db.collection_name.find().pretty()`

**Output:**

```
studentinfo> db.student_record.find().pretty()  
[  
  {  
    _id: ObjectId("63f5b832bb3c56ccada887ae"),  
    id: 1,  
    name: 'Ranbir',  
    subject: 'Bigdata',  
    city: 'Mumbai',  
    age: 22  
  },  
  {  
    _id: ObjectId("63f5ba9ebb3c56ccada887af"),  
    id: 2,  
    name: 'Kartik',  
    subject: 'Java',  
    city: 'Surat',  
    age: 23  
  },  
  {  
    _id: ObjectId("63f5ba9ebb3c56ccada887b0"),  
    id: 3,  
    name: 'Ananya',  
    subject: 'Pytho',  
    city: 'Delhi',  
    age: 22  
  },  
  {  
    _id: ObjectId("63f5be54bb3c56ccada887b1"),  
    id: 4,  
    name: 'Kiara',  
    subject: 'Algoritham',  
    city: 'Agra',  
    age: 25,  
    marks: 79  
  }  
]
```

To find specific data:

**Syntax:** `db.collection_name.findOne({'name':'Ranbir'})`

**Output:**

```
studentinfo> db.student_record.find({'name':'Ananya'})
[
  {
    _id: ObjectId("63f5ba9ebb3c56ccada887b0"),
    id: 3,
    name: 'Ananya',
    subject: 'Pytho',
    city: 'Delhi',
    age: 22
  }
]
```

### 3.Update Operations

Update operations modify existing documents in a collection. MongoDB provides the following methods to update documents of a collection:

**Syntax:** `db.collection.updateOne(filter, update)`



## Output:

```
studentinfo> db.student_record.updateOne({'name':'Kartik'},{$set:{'subject':'Perl'}})
{
  acknowledged: true,
  insertedId: null,
  matchedCount: 1,
  modifiedCount: 1,
  upsertedCount: 0
}
```

## Output:

```
{  
  _id: ObjectId("63f5ba9ebb3c56ccada887af"),  
  id: 2,  
  name: 'Kartik',  
  subject: 'Perl',  
  city: 'Surat',  
  age: 23  
},
```

```
db.collection.updateMany(filter, update)
```

## 4.Delete Operations:

Delete operations remove documents from a collection. MongoDB provides the following methods to delete documents of a collection:

**Syntax:** `db.collection.deleteOne()`

Output:

```
studentinfo> db.student_record.deleteOne({'marks':79})
{ acknowledged: true, deletedCount: 1 }
```

```
studentinfo> db.student_record.find()
[
  {
    _id: ObjectId("63f5b832bb3c56ccada887ae"),
    id: 1,
    name: 'Ranbir',
    subject: 'Bigdata',
    city: 'Mumbai',
    age: 22
  },
  {
    _id: ObjectId("63f5ba9ebb3c56ccada887af"),
    id: 2,
    name: 'Kartik',
    subject: 'Perl',
    city: 'Surat',
    age: 23
  },
  {
    _id: ObjectId("63f5ba9ebb3c56ccada887b0"),
    id: 3,
    name: 'Ananya',
    subject: 'Pytho',
    city: 'Delhi',
    age: 22
  }
]
```

```
db.collection.deleteMany()
```

To drop Collection:

Syntax: `db.collection_name.drop()`

```
student_record> db.student_record.drop()  
false
```

To drop DataBase:

Syntax: `db.dropDatabase()`

```
student_record> db.dropDatabase()  
{ ok: 1, dropped: 'student_record' }
```

## Aggregate function in MongoDB

- Aggregation is a way of processing a large number of documents in a collection by means of passing them through different stages.
- The stages make up what is known as a pipeline.
- The stages in a pipeline can filter, sort, group, reshape and modify documents that pass through the pipeline.

For the aggregation in MongoDB, you should use `aggregate()` method.

### Syntax:

```
db.collection_name.aggregate(aggregate_operation)
```

#### 1.\$sum:

Sums up the defined value from all documents in the collection.



**Syntax:** `db.collection_name.aggregate([{$group:{$sum:1}}])`

**Output:**

```
studentinfo> db.student.aggregate([{$group:{_id:"$subject",TotalSubject:{$sum:1}}]})
[
  { _id: 'Algoritham', TotalSubject: 1 },
  { _id: 'Maths', TotalSubject: 2 },
  { _id: 'Python', TotalSubject: 1 }
]
```

**2.\$avg:** Calculates the average of all given values from all documents in the collection.

### Syntax:

```
db.collection_name.aggregate([{$group:{_id:""},{$avg:1}}])
```

### Output:

```
studentinfo> db.student.aggregate([{$group:{_id:"$marks",averagemarks:{$avg:1}}]])
[
  { _id: 83, averagemarks: 1 },
  { _id: 94, averagemarks: 1 },
  { _id: 88, averagemarks: 1 },
  { _id: 91, averagemarks: 1 }
]
```

**3.\$min:** Gets the minimum of the corresponding values from all documents in the collection.

**Syntax:**

```
db.collection_name.aggergate([{$group:{_id:""},{$min:1}}])
```

**Output:**

```
studentinfo> db.student.aggregate([{$group:{_id:"$marks",minmarks:{$min:1}}]})
[
  { _id: 83, minmarks: 1 },
  { _id: 94, minmarks: 1 },
  { _id: 88, minmarks: 1 },
  { _id: 91, minmarks: 1 }
]
```

**4.\$max:** Gets the maximum of the corresponding values from all documents in the collection.

**Syntax:** `db.collection_name.aggrergate([{$group:{_id:""},{$min:1}}})`

**Output:**

```
studentinfo> db.student.aggregate([{$group:{_id:"$age",maxage:{$max:1}}}])
[
  { _id: 23, maxage: 1 },
  { _id: 22, maxage: 1 },
  { _id: 24, maxage: 1 },
  { _id: 21, maxage: 1 }
]
```

**5.\$match:** This is a filtering operation and thus this can reduce the amount of documents that are given as input to the next stage.

**Syntax:** `db.collection_name.aggregate([{$match:{age:{$lt:22}}}]])`

**Output:**

```
studentinfo> db.student.aggregate([{$match:{age:{$lt:22}}}]])
[
  {
    _id: ObjectId("63f6ef9d191f4ccedbfd7478"),
    id: 4,
    name: 'Ananya',
    subject: 'Maths',
    city: 'Delhi',
    age: 21,
    marks: 91,
    status: 'Pass'
  }
]
```

\$match:

```
studentinfo> db.student_record.aggregate([{$match:{age:22}}])
[
  {
    _id: ObjectId("63f5b832bb3c56ccada887ae"),
    id: 1,
    name: 'Ranbir',
    subject: 'Bigdata',
    city: 'Mumbai',
    age: 22
  },
  {
    _id: ObjectId("63f5ba9ebb3c56ccada887b0"),
    id: 3,
    name: 'Ananya',
    subject: 'Pytho',
    city: 'Delhi',
    age: 22
  }
]
```

## \$match:

```
studentinfo> db.student_record.aggregate([{$match:{'subject':'Bigdata'}}])
[
  {
    _id: ObjectId("63f5b832bb3c56ccada887ae"),
    id: 1,
    name: 'Ranbir',
    subject: 'Bigdata',
    city: 'Mumbai',
    age: 22
  }
]
```

6. **\$group**: It is used to group documents based on some value.

**Syntax**: `db.collection_name.aggregate([{$group:{_id:{$sum:1}}}]])`

**Output**:

```
studentinfo> db.student_record.aggregate([{$group:{_id:"$name",TotalName:{$sum:1}}}]])
[
  { _id: 'Ranbir', TotalName: 1 },
  { _id: 'Ananya', TotalName: 1 },
  { _id: 'Kartik', TotalName: 1 }
]
```



## Example:2

```
studentinfo> db.student_record.aggregate([{$group:{_id:"$name",MinimumAge:{$min:1}}}] )
[
  { _id: 'Kartik', MinimumAge: 1 },
  { _id: 'Ananya', MinimumAge: 1 },
  { _id: 'Ranbir', MinimumAge: 1 }
]
```

**7.\$skip:** It is used to skip n number of documents and passes the remaining documents

**Syntax:** `db.student.find({}).pretty().skip(1)`

**Output:**

```
studentinfo> db.student.find({city:'Surat'}).pretty().skip(1)
[
  {
    _id: ObjectId("63f6eee5191f4ccedbffd7477"),
    id: 3,
    name: 'Kiara',
    subject: 'Python',
    city: 'Surat',
    age: 24,
    marks: 83
  }
]
```

## Example:2

```
studentinfo> db.student.find({subject:'Maths'}).pretty().skip(1);
[
  {
    _id: ObjectId("63f6ef9d191f4ccedbffd7478"),
    id: 4,
    name: 'Ananya',
    subject: 'Maths',
    city: 'Delhi',
    age: 21,
    marks: 91,
    status: 'Pass'
  }
]
```

**8.\$limit:** It is used to pass first n number of documents thus limiting them.

**Syntax:** db.collection\_name.find({}).limit(1)

**Output:**

```
studentinfo> db.student.find({age:23}).limit(1)
[
  {
    _id: ObjectId("63f6eee5191f4ccedbfd7476"),
    id: 2,
    name: 'Kartik',
    subject: 'Algoritham',
    city: 'Surat',
    age: 23,
    marks: 94
  }
]
```

**\$sort:** sort data in ascending and descending order.

**Syntax:** `db.collection_name.aggggregate( {$sort: {} })`

**Output:**

```
studentinfo> db.student.aggregate({$sort:{'city':1}})
[
  {
    _id: ObjectId("63f6ef9d191f4ccedbfd7478"),
    id: 4,
    name: 'Ananya',
    subject: 'Maths',
    city: 'Delhi',
    age: 21,
    marks: 91,
    status: 'Pass'
  },
  {
    _id: ObjectId("63f6eee5191f4ccedbfd7475"),
    id: 1,
    name: 'Ranbir',
    subject: 'Maths',
    city: 'Mumbai',
    age: 22,
    marks: 88
  },
  {
    _id: ObjectId("63f6eee5191f4ccedbfd7476"),
    id: 2,
    name: 'Kartik',
    subject: 'Algoritham',
    city: 'Surat',
    age: 23,
    marks: 94
  },
  {
    _id: ObjectId("63f6eee5191f4ccedbfd7477"),
    id: 3,
    name: 'Kiara',
    subject: 'Python',
    city: 'Surat',
    age: 24,
    marks: 83
  }
]
```

## Descending order

```
studentinfo> db.student.aggregate({$sort:{'city':-1}})
[
  {
    _id: ObjectId("63f6eee5191f4ccedbfd7476"),
    id: 2,
    name: 'Kartik',
    subject: 'Algoritham',
    city: 'Surat',
    age: 23,
    marks: 94
  },
  {
    _id: ObjectId("63f6eee5191f4ccedbfd7477"),
    id: 3,
    name: 'Kiara',
    subject: 'Python',
    city: 'Surat',
    age: 24,
    marks: 83
  },
  {
    _id: ObjectId("63f6eee5191f4ccedbfd7475"),
    id: 1,
    name: 'Ranbir',
    subject: 'Maths',
    city: 'Mumbai',
    age: 22,
    marks: 88
  },
  {
    _id: ObjectId("63f6ef9d191f4ccedbfd7478"),
    id: 4,
    name: 'Ananya',
    subject: 'Maths',
    city: 'Delhi',
    age: 21,
    marks: 91,
    status: 'Pass'
  }
]
studentinfo>
```

## Syntax:

```
studentinfo> db.student_record.count()
DeprecationWarning: Collection.count() is deprecated. Use countDocuments or estimatedDocumentCount.
3
```

```
studentinfo> db.student_record.aggregate([{$count:"name"}])
[ { name: 3 } ]
```



To find data with only name filed:

```
[ { '_id': ObjectId('63f6eee5191f4ccdbfd7477'), name: 'Kiara' } ]
```

To find the data without name filed:

```
[ { '_id': ObjectId('63f6eee5191f4ccdbfd7477'),  
studentinfo> db.student.find({subject:'Python'}, {name:0})  
[  
  {  
    _id: ObjectId("63f6eee5191f4ccdbfd7477"),  
    id: 3,  
    subject: 'Python',  
    city: 'Surat',  
    age: 24,  
    marks: 83  
  }  
]
```

**Count():** To find the total number of the document, unlike find() method it does not find all the document rather it counts them and return a number.

```
studentinfo> db.student.find().count()
4
```

```
studentinfo> db.student.find({subject:'Maths'}).count()
2
```

**distinct():** method that finds distinct values of the specified field

**Syntax:** db.collection\_name.distinct(“”)

**Output:**

```
studentinfo> db.student.distinct('city')  
[ 'Delhi', 'Mumbai', 'Surat' ]  
studentinfo\
```

**\$lt**: finds the less than value according to condition.

```
studentinfo> db.student.find({age:{$lt:22}})
[
  {
    _id: ObjectId("63f6ef9d191f4ccedbfd7478"),
    id: 4,
    name: 'Ananya',
    subject: 'Maths',
    city: 'Delhi',
    age: 21,
    marks: 91,
    status: 'Pass'
  }
]
```

`$lte`: finds the  
less than equal  
value according  
to condition.

```
studentinfo> db.student.find({age:{$lte:23}})
[
  {
    _id: ObjectId("63f6eee5191f4ccedbffd7475"),
    id: 1,
    name: 'Ranbir',
    subject: 'Maths',
    city: 'Mumbai',
    age: 22,
    marks: 88
  },
  {
    _id: ObjectId("63f6eee5191f4ccedbffd7476"),
    id: 2,
    name: 'Kartik',
    subject: 'Algoritham',
    city: 'Surat',
    age: 23,
    marks: 94
  },
  {
    _id: ObjectId("63f6ef9d191f4ccedbffd7478"),
    id: 4,
    name: 'Ananya',
    subject: 'Maths',
    city: 'Delhi',
    age: 21,
    marks: 91,
    status: 'Pass'
  }
]
```

`$gt`: finds the  
greater than value  
According to  
condition.

```
studentinfo> db.student.find({marks:{$gt:90}})
[
  {
    _id: ObjectId("63f6eee5191f4ccedbffd7476"),
    id: 2,
    name: 'Kartik',
    subject: 'Algoritham',
    city: 'Surat',
    age: 23,
    marks: 94
  },
  {
    _id: ObjectId("63f6ef9d191f4ccedbffd7478"),
    id: 4,
    name: 'Ananya',
    subject: 'Maths',
    city: 'Delhi',
    age: 21,
    marks: 91,
    status: 'Pass'
  }
]
```

**\$gte**: finds the greater than equal value according to condition.

```
studentinfo> db.student.find({marks:{$gte:94}})
[
  {
    _id: ObjectId("63f6eee5191f4ccedbfd7476"),
    id: 2,
    name: 'Kartik',
    subject: 'Algoritham',
    city: 'Surat',
    age: 23,
    marks: 94
  }
]
```



## Example:2

```
studentinfo> db.student.find({age:{$gte:24}})
[
  {
    _id: ObjectId("63f6eee5191f4ccedb7477"),
    id: 3,
    name: 'Kiara',
    subject: 'Python',
    city: 'Surat',
    age: 24,
    marks: 83
  }
]
```

```
studentinfo> db.student.find({}, {name:1,subject:1},{age:{$lte:23}})
[
  {
    _id: ObjectId("63f6eee5191f4ccdbfd7475"),
    name: 'Ranbir',
    subject: 'Maths'
  },
  {
    _id: ObjectId("63f6eee5191f4ccdbfd7476"),
    name: 'Kartik',
    subject: 'Algoritham'
  },
  {
    _id: ObjectId("63f6eee5191f4ccdbfd7477"),
    name: 'Kiara',
    subject: 'Python'
  },
  {
    _id: ObjectId("63f6ef9d191f4ccdbfd7478"),
    name: 'Ananya',
    subject: 'Maths'
  }
]
```

# Questions

- 1.What is MongoDB? Discuss important features of MongoDB.
- 2.Explain basic CRUD operations with example in MongoDB
- 3.Explain database, collection, document and fields with respect to MongoDB.  
Also give its equivalent term in RDBMS.
- 4.Explain use of aggregate function in MongoDB with suitable example
- 5.What are the data types of MongoDB? Explain in brief.
- 6.Define features of MongoDB.

THANK YOU