CLASS 2:

ADD, UPDATE AND DELETE

ADD OPERATION:

In MongoDB, the "add operation" typically refers to adding a new document or inserting a new record into a collection. If one of the arguments is a date, $add treats the other arguments as milliseconds to add to the date.

Syntax:

{ $add: [<expression1>,

<expression2>, ...

] }

The arguments can be any valid expression as long as they resolve to either all numbers or to numbers and a date.

```
db.stud.find({
$and:[
{home_city: "City 2"},
{blood_group: "B+" }
]
});


_id: ObjectId('665a89d776fc88153fffc0b4'),
name: 'Student 504',
age: 21,
courses: "['Physics', 'Computer Science', 'English', 'Mathematics']",
gpa: 2.42,
home_city: 'City 2',
blood_group: 'B+',
is_hotel_resident: true
,

_id: ObjectId('665a89d776fc88153fffc0eb'),
name: 'Student 367',
age: 19,
courses: "['English', 'Physics', 'History', 'Mathematics']",
gpa: 2.81,
home_city: 'City 2',
blood_group: 'B+',
is_hotel_resident: false
```

UPDATE OPERATION:

The update operation in MongoDB is used to modify documents in a collection. The operation can update specific fields, add new fields, or even remove fields in the matched documents.

Syntax:

```
db.collection.updateMany(
 <filter>,
<update>,
 <options>
 )
```

You can use the following method:

1. UpdateOne (filter, update, options): Updates a single document that matches the filter.

```
test> db.stu.updateOne({name:"Alice Smith"},{$set:{gpa:3.8}});
{
  acknowledged: true,
  insertedId: null,
  matchedCount: 1,
  modifiedCount: 0,
  upsertedCount: 0
}
test>
```

2. UpdateMany (filter, update, options): Updates all documents that match the filter.

```
test> db.stu.updateMany({gpa:{$lt:3.0}},{$inc:{gpa:0.5}});
{
  acknowledged: true,
  insertedId: null,
  matchedCount: 261,
  modifiedCount: 261,
  upsertedCount: 0
}
```

DELETE OPERATION:

The delete operation in MongoDB is used to remove documents from a collection. This can be done using methods such as 'deleteOne' or 'deleteMany', depending on whether you want to delete a single document or multiple documents that match a given filter.

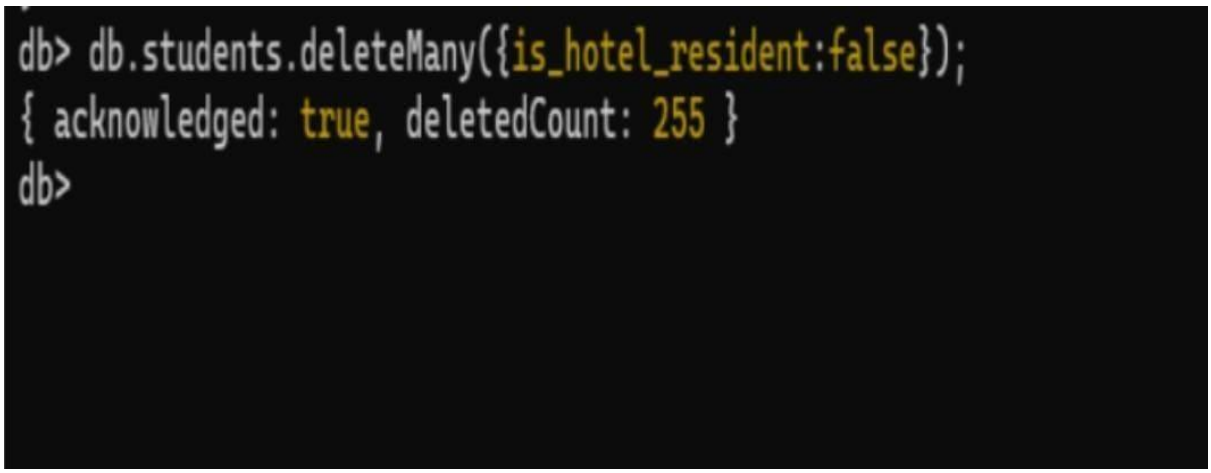
Syntax:

```
db.collection.deleteMany(
   <filter>,
   {
      writeConcern: <document>,
      collation: <document>
   }
)
```

To delete documents from a mongodb collection,you can use following collections:

1. DELETE ONE():

2. DELETE MANY():



DOCUMENTS, COLLECTIONS

Documents:

A document is a basic unit of data. It's a record that is stored in a collection, analogous to a row in a table in relational databases. Documents in MongoDB are represented in BSON (Binary JSON) format, which allows for the embedding of data and arrays within a single record.

Syntax:

```
{  field1: value1 field2: value2 ....
fieldN: valueN
}
```

Collections:

A collection is a grouping of MongoDB documents. Collections are akin to tables in relational databases. A single database is allowed to store multiple collections.

Syntax:

```
db.createCollection(name, options)
```

CLASS 3:

WHERE, CRUD

Where:

Where operation allows you to execute JavaScript code on the server to query documents. This operation provides a way to perform more complex queries that can't be achieved using standard query operators. The $where clause can include any JavaScript expression or function that returns a boolean value.

Syntax:

{ $where: <string JavaScript Code> }

```
// Find all students with GPA greater than 3.5
db.students.find({ gpa: { $gt: 3.5 } });

// Find all students from "City 3"
db.students.find({ home_city: "City 3" });
```

CRUD:

(Create, Read, Update, and Delete) are the basic set of operations that allow users to interact with the MongoDB server. As we know, to use MongoDB we need to interact with the MongoDB server to perform certain operations like entering new data into the application, updating data into the application, deleting data from the application, and reading the application data.

C → CREATE / INSERT

R → READ

U →UPDATE

D →DELETE

Read:

The create or insert operations are used to insert or add new documents in the collection. If a collection does not exist, then it will create a new collection in the database.

Create / Insert:

The create or insert operations are used to insert or add new documents in the collection. If a collection does not exist, then it will create a new collection in the database.

To insert an operation ,you can use following methods:

1. Insert One(): We are inserting details of a single student in the form of document in the student collection using,
   db.collection.insertOne() ……..method

```
db> const studentData = {
... "name":"Sam",
... "age":22,
... "courses":["Computer Science" , "Mathematics"]
,
... "gpa":3.4,
... "home_city":"City 3",
... "blood_group":"B+",
... "is_hotel_resident":false
... }

db> db.students.insertOne(studentData)
{
  acknowledged: true,
  insertedId: ObjectId('6658a0c70cce0c5ec1cdcdf6')
}
db>
```

2. Insert Many(): We are inserting details of the multiple students in the form of documents in the student collection using,
   db.collection.insertMany()……method.

CLASS 4:

PROJETION, LIMITS AND SELECTORS

Projection:

MongoDB Projection is a special feature allowing you to select only the necessary data rather than selecting the whole set of data from the document. For Example, If a Document contains 10 fields and only 5 fields are to be shown the same can be achieved using the Projections.

This enable us to:

Project concise yet transparent data

Filter data without impacting the overall database performance

Benefits:

1. Reduced Data Transfer:

By projecting only the necessary fields, you minimize the amount of data transferred over the network, which can significantly improve performance, especially when dealing with large documents or datasets.

2. Improved Query Performance:

Fetching only the required fields reduces the amount of data that MongoDB needs to process and retrieve from the disk, resulting in faster query execution.

3. Lower Memory Usage:

When you project only the needed fields, the result set is smaller, which can help reduce memory consumption on the client-side application.

4. Increased Read Efficiency:

By narrowing down the data fetched from the database, projections can help reduce the read load on the database, making it more efficient and scalable under high load conditions.

5. Simpler Data Handling:

Returning only the relevant fields makes the application code simpler and easier to manage, as it does not need to deal with unnecessary data.

6. Security and Privacy:

Projections can help enforce security and privacy by ensuring that

sensitive fields are not exposed to unauthorized users or systems.

Get Selected Attributes:

To select specific fields in a MongoDB query, you use the find() method with a projection document. The projection document specifies which fields to include (using 1) or exclude (using 0).

Syntax:

db. students. find ({}, { _id: 0 })

```
b> db.students.find({},{_id:0});

 {
   name: 'Student 328',
   age: 21,
   courses: "['Physics', 'Computer Science', 'English']",
   gpa: 3.42,
   home_city: 'City 2',
   blood_group: 'AB-',
   is_hotel_resident: true
 },
 {
   name: 'Student 468',
   age: 21,
   courses: "['Computer Science', 'Physics', 'Mathematics', 'History']",
   gpa: 3.97,
   blood_group: 'A-',
   is_hotel_resident: true
 },
 {
   name: 'Student 504',
   age: 21,
   courses: "['Physics', 'Computer Science', 'English', 'Mathematics']",
   gpa: 2.92,
   home_city: 'City 2',
   blood_group: 'B+',
   is_hotel_resident: true
 },
 {
   name: 'Student 915',
   age: 22,
   courses: "['Computer Science', 'History', 'Physics', 'English']",
   gpa: 3.37,
   blood_group: 'AB+',
   is_hotel_resident: true
 },
 {
   name: 'Student 367',
   age: 25,
   courses: "['History', 'Physics', 'Computer Science']",
   gpa: 3.11,
```

Ignore attributes:

Ignoring an attribute (or field) in query results is done using projection. By setting the value of a field to 0 in the projection document, you can exclude that field from the returned documents.

Syntax:

```
Db. students. find({},{_id:0});
```

```
test> db.stu.find({},{name:1,age:1}).count();
500
test>
```

Limit():

Limit() method limits the number of records or documents that you want. It basically defines the max limit of records/documents that you want.

Syntax:

```
db.collectionName.find(<query>).limit(<number>)
```

Coding:

Getting first five documents:

```
Type "it" for more
db> db.students.find({} , {_id:0}).limit(5)
[
  {
    name: 'Student 948',
    age: 19,
    courses: "['English', 'Computer Science', 'Physics', 'Mathematics']",
    gpa: 3.44,
    home_city: 'City 2',
    blood_group: 'O+',
    is_hotel_resident: true
  },
  {
    name: 'Student 157',
    age: 20,
    courses: "['Physics', 'English']",
    gpa: 2.27,
    home_city: 'City 4',
    blood_group: 'O-',
    is_hotel_resident: true
  },
  {
```

Selector:

- AND operation
- OR operation
- Comparison between gt and lt:

Grater than:

The $gt operator in MongoDB is used to specify a query condition where the field value must be greater than a specified value. It stands for "greater than."

Lesser than:

the less than operation is performed using the $lt operator. This operator allows you to query documents where a specific field's value is less than a given value.

Greater Than ($gt):

- **Functionality:** The $gt operator selects documents where the value of a specified field is greater than (but not equal to) a given value.

Example:

```
db. collection. find({ field: { $gt: value } })
```

selects documents where the field value is greater than value.

- **Use Cases:** Filtering documents with values exceeding a certain threshold, selecting the latest entries based on a timestamp, or retrieving documents with numerical values above a specific limit.

Less Than ($lt):

- **Functionality**: The $lt operator selects documents where the value of a specified field is less than (but not equal to) a given value.

Example:

```
db. collection. find({ field: { $lt: value } })
```

selects documents where the field value is less than value.

- **Use Cases**: Filtering documents with values below a certain threshold, selecting older entries based on a timestamp, or retrieving documents with numerical values below a specific limit.

Example:

```
db. students. find({age:{$gt:20} });
```

```
db> db.stud.find({age:{$gt:20}});
[
  {
    _id: ObjectId('665a89d776fc88153fffc09f'),
    name: 'Student 346',
    age: 25,
    courses: "['Mathematics', 'History', 'English']",
    gpa: 3.31,
    home_city: 'City 8',
    blood_group: 'O-',
    is_hotel_resident: true
  },
  {
    _id: ObjectId('665a89d776fc88153fffc0a0'),
    name: 'Student 930',
    age: 25,
    courses: "['English', 'Computer Science', 'Mathematics', 'History']",
    gpa: 3.63,
    home_city: 'City 3',
    blood_group: 'A-',
    is_hotel_resident: true
  },
  {
    _id: ObjectId('665a89d776fc88153fffc0a1'),
    name: 'Student 305',
    age: 24,
    courses: "['History', 'Physics', 'Computer Science', 'Mathematics']",
    gpa: 3.4,
    home_city: 'City 6',
    blood_group: 'O+',
    is_hotel_resident: true
```