Github Link: (https://github.com/soukhyakj/MongoDB/tree/main***)***

INTRODUCTION:

MongoDB is a popular, open-source, NoSQL database that allows you to store and manage large amounts of data in a flexible and scalable way. It is designed to handle the demands of modern web and mobile applications, and is widely used in industries such as finance, healthcare, and e-commerce.

Experiment 1

a. Illustration of Where Clause, AND, OR operations in MongoDB:

1.Where clause:

The most common use of the **WHERE** clause is to filter data based on a condition.

Input:

// find all students with GPA lesser than 3

db.stu.find({gpa:{$lt:3}});

```
db.stu.find({gpa:{$lt:3}});
```

Output:

```
[
  {
    _id: ObjectId('66af17231d6eb7f7f70165cb'),
    name: 'Student 157',
    age: 20,
    courses: "['Physics', 'English']",
    gpa: 2.27,
    home_city: 'City 4',
    blood_group: 'O-',
    is_hotel_resident: true
  },
  {
    _id: ObjectId('66af17231d6eb7f7f70165cc'),
    name: 'Student 316',
    age: 20,
    courses: "['Physics', 'Computer Science', 'Mathematics', 'History']",
    gpa: 2.32,
    blood_group: 'B+',
    is_hotel_resident: true
  },
  {
    _id: ObjectId('66af17231d6eb7f7f70165d1'),
    name: 'Student 563',
    age: 18,
    courses: "['Mathematics', 'English']",
    gpa: 2.25,
    blood_group: 'AB+',
    is_hotel_resident: false
  },
  {
    _id: ObjectId('66af17231d6eb7f7f70165d2'),
    name: 'Student 440',
    age: 21,
    courses: "['History', 'Physics', 'Computer Science']",
```

2.AND clause:

$and clause is used to combine multiple conditions in a query and filter documents based on multiple criteria.

Input:

// find all students who live in “City 5” AND have blood group “A+”

```
db> db.students.find({
... $and:[
... { home_city:"City 5"},
... { blood_group:"A+"}
... ]
... });
```

Output:


```
[
  {
    _id: ObjectId('66a215ba046ad7f15a24ca84'),
    name: 'Student 142',
    age: 24,
    courses: "['History', 'English', 'Physics', 'Computer Science']",
    gpa: 3.41,
    home_city: 'City 5',
    blood_group: 'A+',
    is_hotel_resident: false
  },
  {
    _id: ObjectId('66a215ba046ad7f15a24cba4'),
    name: 'Student 947',
    age: 20,
    courses: "['Physics', 'History', 'English', 'Computer Science']",
    gpa: 2.86,
    home_city: 'City 5',
    blood_group: 'A+',
    is_hotel_resident: true
  },
  {
    _id: ObjectId('66a215ba046ad7f15a24cc16'),
    name: 'Student 567',
    age: 22,
    courses: "['Computer Science', 'History', 'English', 'Mathematics']",
    gpa: 2.01,
    home_city: 'City 5',
    blood_group: 'A+',
    is_hotel_resident: true
  }
]
```


3. OR clause:

The **$or** clause is used to filter documents that match at least one of the conditions specified in the clause.

Input:

//find all students who are hotel resident OR have GPA less than 3.5.

```
db.students.find({  $or:[  {is_hotel_resident: true},  {gpa :{$lt: 3.5}}  ]});
```

Output:

b. Execute the Commands of MongoDB and operations in MongoDB: Insert, Query, Update, Delete and Projection.

1.Insert document

Input:

```
db> const studentData={
... "name":"Alice Smith",
... "age":22,
... "courses":["Mathematics","Computer Science","English"],
... "gpa":3.8,
... "home_city":"New York",
... "blood_group":"A+",
... "is_hotel_resident":false
... };

db> db.students.insertOne(studentData);
```

Output:

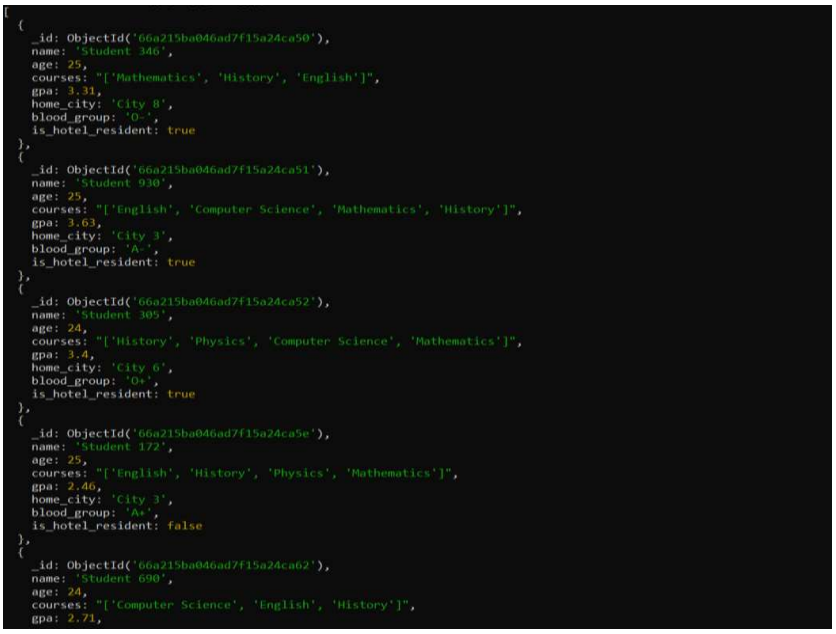
```
{
  acknowledged: true,
  insertedId: ObjectId('6661da38b0d232162dcdcdf6')
}
db>
```

2. Query document

Input:

```
db> db.students.find({age:{$gt:23}});
```

Output:



3. Update document

Input:

```
db>  db.students.updateOne({name:"Alice Smith"},{$set:{gpa:3.8}});
```

Output:

```
{
  acknowledged: true,
  insertedId: null,
  matchedCount: 1,
  modifiedCount: 0,
  upsertedCount: 0
}
```

4.Delete document

Input:

```
db> db.students.deleteOne({name:"John Doe"});
{ acknowledged: true, deletedCount: 0 }
```

Output:

```
{ acknowledged: true, deletedCount: 0 }
```

5.Projection

Input:

```
db> db.students.find({},{name:1,gpa:1});
```

Output:


```
[
  {
    _id: ObjectId('66a215ba046ad7f15a24ca4d'),
    name: 'Student 948',
    gpa: 3.44
  },
  {
    _id: ObjectId('66a215ba046ad7f15a24ca4e'),
    name: 'Student 157',
    gpa: 2.27
  },
  {
    _id: ObjectId('66a215ba046ad7f15a24ca4f'),
    name: 'Student 316',
    gpa: 2.32
  },
  {
    _id: ObjectId('66a215ba046ad7f15a24ca50'),
    name: 'Student 346',
    gpa: 3.31
  },
  {
    _id: ObjectId('66a215ba046ad7f15a24ca51'),
    name: 'Student 930',
    gpa: 3.63
  },
  {
    _id: ObjectId('66a215ba046ad7f15a24ca52'),
    name: 'Student 305',
    gpa: 3.4
  },
  {
    _id: ObjectId('66a215ba046ad7f15a24ca53'),
    name: 'Student 268',
    gpa: 3.98
  },
  {
    _id: ObjectId('66a215ba046ad7f15a24ca54'),
    name: 'Student 563',
    gpa: 2.25
  },
  {
    _id: ObjectId('66a215ba046ad7f15a24ca55'),
    name: 'Student 440',
    gpa: 2.06
  },
```

Experiment 2

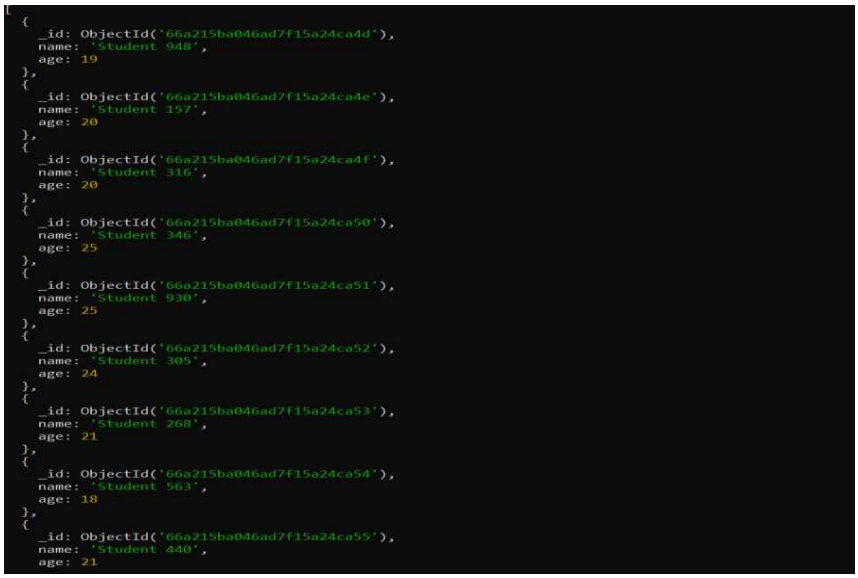
a. Develop a MongoDB query to select certain fields and ignore some fields of the documents:

1.Get only name & age for all students

Input:

```
db> db.students.find({},{name:1,age:1});
```

Output:



2.Select certain fields and ignore some fields

Input:

```
db> db.students.find({},{name:1,gpa:1,_id:0});
```

Output:

3.Ignoring attributes

input:

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

Output:

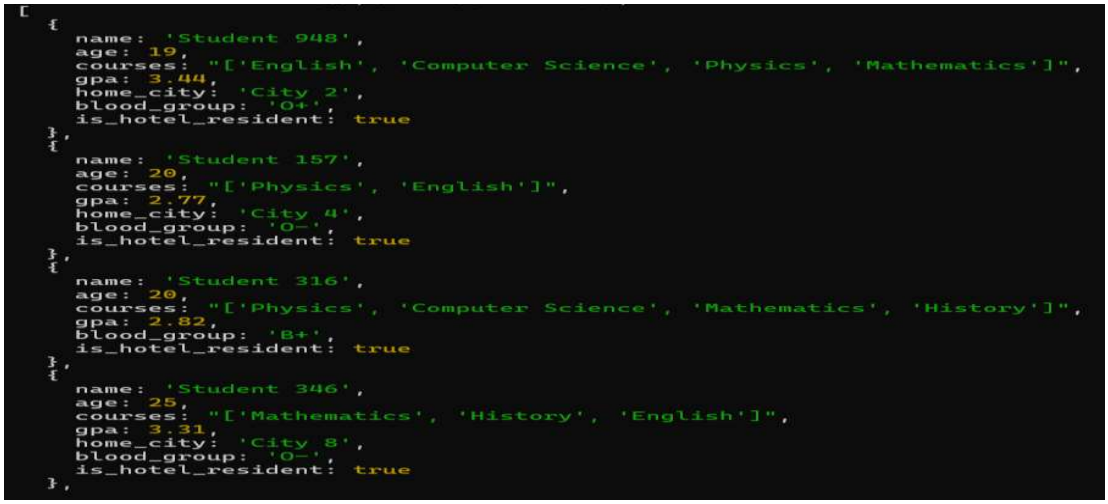
b. Develop a MongoDB query to display the first 5 documents from the results obtained in a. [use of limit and find]

Input:

// Display the first 5 documents

```
db> db.students.find({},{_id:0}).limit(5);
```

Output:



Experiment 3

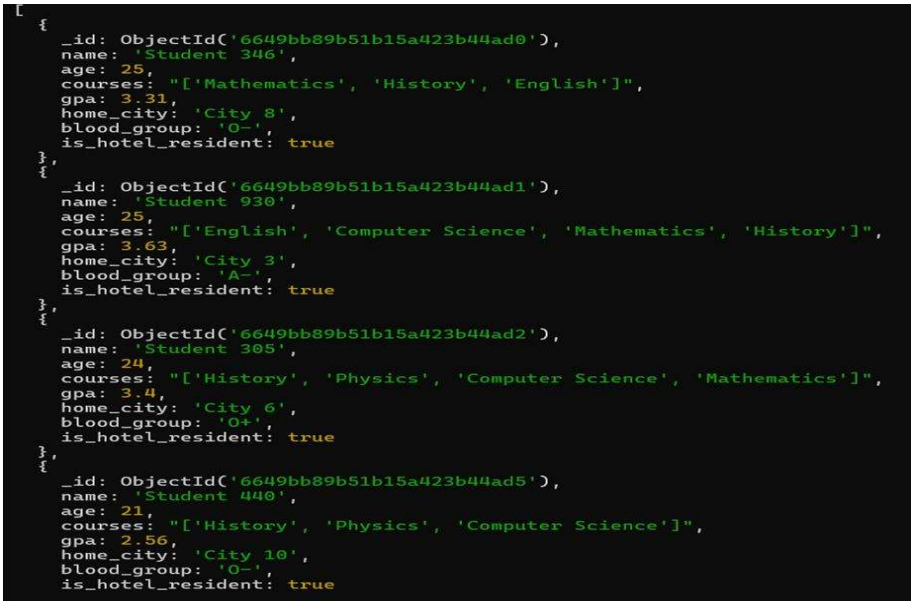
a. Execute query selectors (comparison selectors, logical selectors) and list out the results on any collection

1.Comparison selectors

Input:

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

Output:



2. Logical selectors

Input:

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

Output:

```
[
  {
    _id: ObjectId('6649bb89b51b15a423b44ae5'),
    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
  },
  {
    _id: ObjectId('6649bb89b51b15a423b44c93'),
    name: 'Student 872',
    age: 24,
    courses: "['English', 'Mathematics', 'History']",
    gpa: 3.36,
    home_city: 'City 2',
    blood_group: 'B+',
    is_hotel_resident: true
  }
]
```

b. Execute query selectors (Geospatial selectors, Bitwise selectors) and list out the results on any collection

1. Geospatial selectors

Input:

```
db> db.locations.find({
... location:{
... $geoWithin:{
... $centerSphere:[[-74.005,40.712],0.00621376]
... }
... }
... });
```

Output:

```
[
  {
    _id: 1,
    name: 'Coffee Shop A',
    location: { type: 'Point', coordinates: [ -73.985, 40.748 ] }
  },
  {
    _id: 2,
    name: 'Restaurant B',
    location: { type: 'Point', coordinates: [ -74.009, 40.712 ] }
  },
  {
    _id: 5,
    name: 'Park E',
    location: { type: 'Point', coordinates: [ -74.006, 40.705 ] }
  }
]
```

2. Bitwise selectors

Input:

```
db> const LOBBY_PERMISSION=1;

db> const CAMPUS_PERMISSION=2;

db> db.students_permission.find({
... permissions:{$bitsAllSet:[LOBBY_PERMISSION,CAMPUS_PERMISSION]}
... });
```

Output:

```
[
  {
    _id: ObjectId('66635182d29d811170a4e560'),
    name: 'George',
    age: 21,
    permissions: 6
  },
  {
    _id: ObjectId('66635182d29d811170a4e561'),
    name: 'Henry',
    age: 27,
    permissions: 7
  },
  {
    _id: ObjectId('66635182d29d811170a4e562'),
    name: 'Isla',
    age: 18,
    permissions: 6
  }
]
```

Experiment 4

Create and demonstrate how projection operators ($, $elematch and $slice) would be used in the MondoDB.

1.$ projection operator

Input:

```
db> db.candidates.find({},{name:1,age:1,gpa:1});
```

Output:

2.$elemMatch projection operator

Input:

```
db> db.candidates.find({courses:{$elemMatch:{$eq:"Computer Science"}}},{name:1,"courses.$":1});
```

Output:

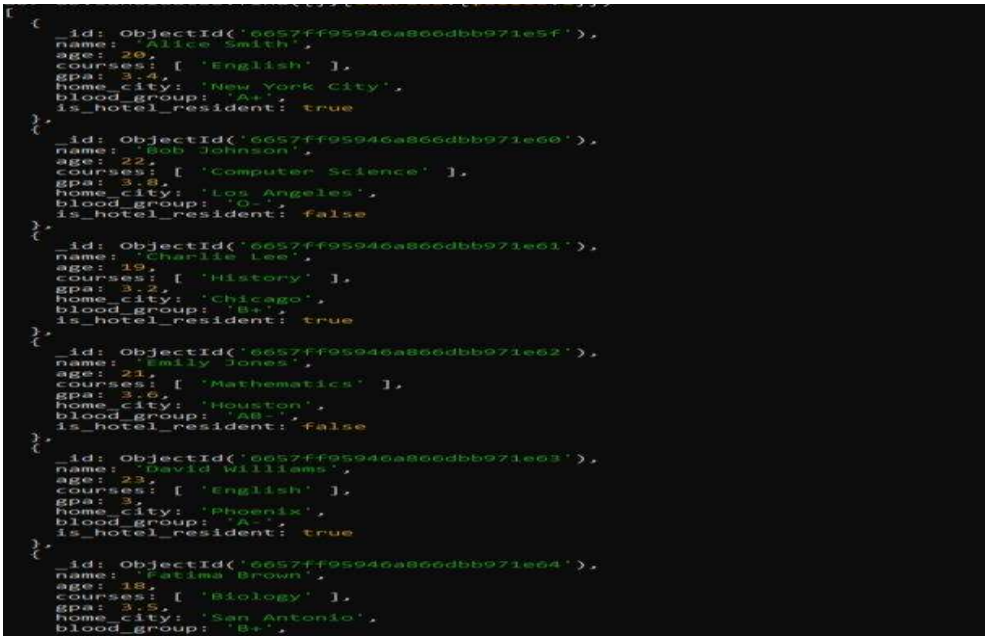
```
[
  { _id: ObjectId('66a3861aa0bde03994b0ee76'), name: 'Bob Johnson' },
  { _id: ObjectId('66a3861aa0bde03994b0ee7b'), name: 'Gabriel Miller' },
  { _id: ObjectId('66a3861aa0bde03994b0ee7f'), name: 'Kevin Lewis' }
]
```

3.$slice

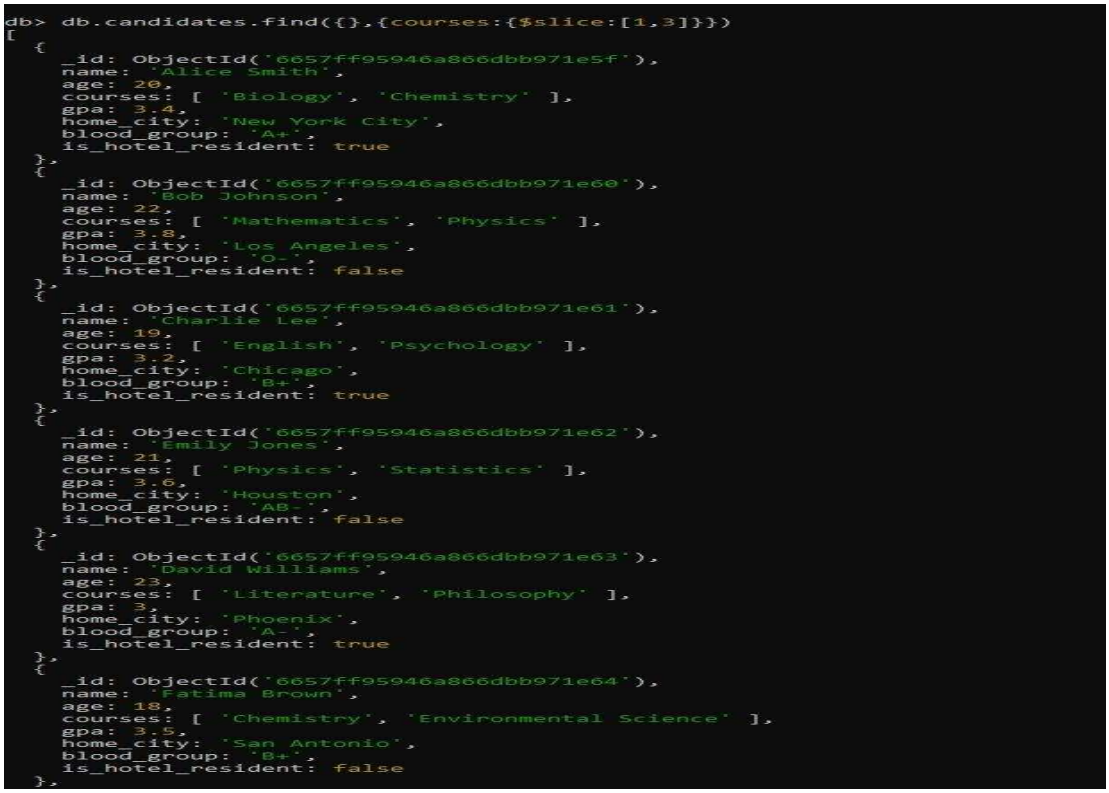
Input:

```
db> db.candidates.find({},{courses:{$slice:1}})
[
```

Output:



Output:



Experiment 5

Execute Aggregation operations ($avg, $min,$max, $push, $addToSet etc.).

1. $avg

Input:

```
db> db.students.aggregate([
... {$group:
... {_id:null,avgAge:{$avg:"$age"}}
... }])
```

Output:

```
[ { _id: null, avgAge: 21.534 } ]
```

2.$min

Input:

```
db> db.students.aggregate([ { $group: { _id: null, minAge: { $min: "$age" } } }])
```

Output:

```
[ { _id: null, minAge: 18 } ]
```

3.$max

Input:

```
db> db.students.aggregate([ { $group: { _id: null, maxAge: { $max: "$age" } } }])
```

Output:

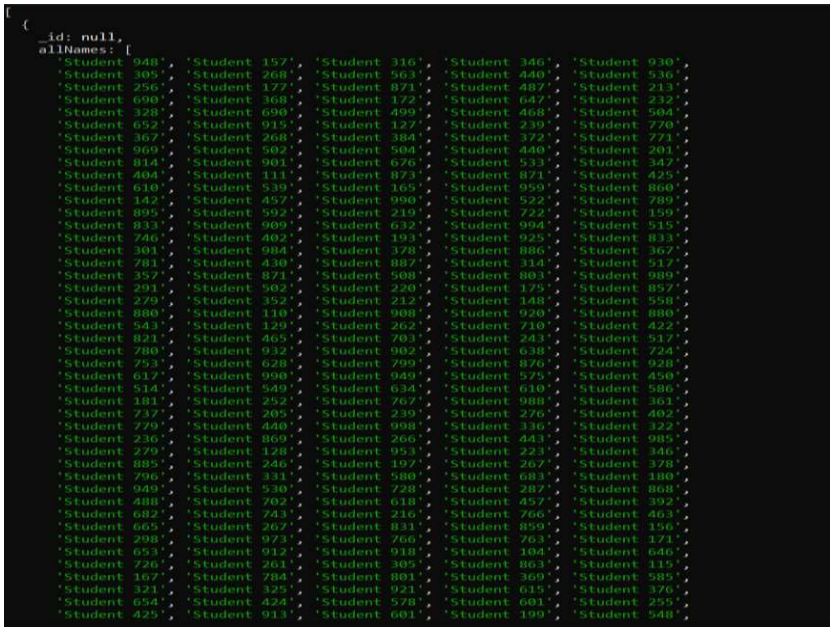
```
[ { _id: null, maxAge: 25 } ]
```

4.$push

Input1:

```
db> db.students.aggregate([ { $group: { _id: null, allNames: { $push: "$name" } } }])
```

Output1:



Input2:

```
db> db.sstudents.aggregate([
... {$group:{
... _id:null,collectAGE:{$push:"$age"},
... collectGPA:{$push:"$gpa"}
... }}
... ]);
```

Output2:

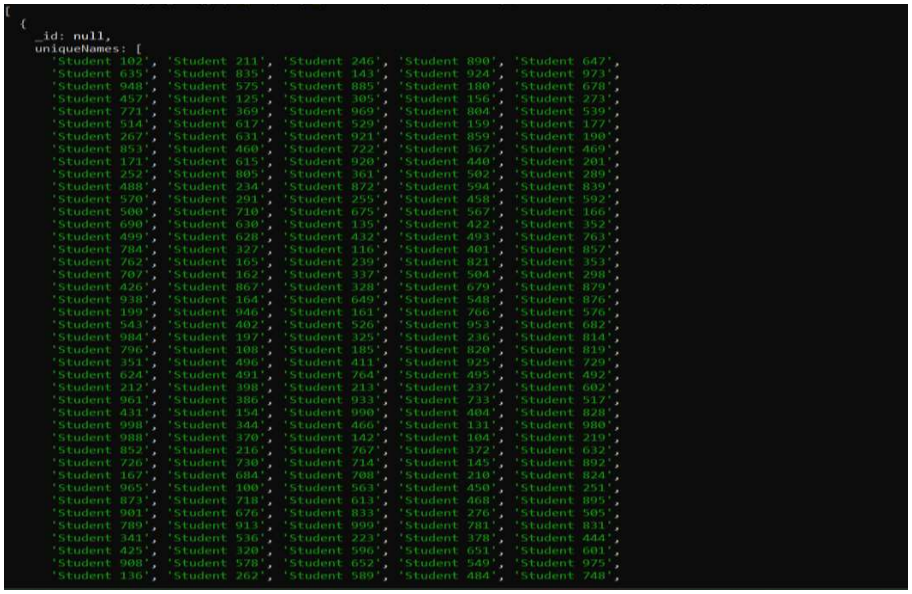
```
[
  {
    _id: null,
    collectAGE: [
      20, 22, 19, 21, 23,
      18, 24, 20, 22, 19,
      21, 23
    ],
    collectGPA: [
      3.4, 3.8, 3.2, 3.6,   3,
      3.5, 3.9, 3.3, 3.7, 3.1,
        4, 3.5
    ]
  }
]
```

5.$addToSet

Input1:

```
db> db.students.aggregate([ { $group: { _id: null, uniqueNames: { $addToSet: "$name" } } }])
[
```

Output1:



Input2:

```
db> db.candidates.aggregate([ {$unwind:"$courses"},
... {$group:{_id:null,uniqueCourses:{$addToSet:"$courses"}}}
... ]);
```

Output2:

Experiment 6

Execute Aggregation Pipeline and its operations (pipeline must contain $match, $group, $sort, $project, $skip etc.

1.**$match:**

Input:

```
db> db.sstudent.aggregate([
... {$match:{age:{$gt:23}}}
... ])
```

Output:

```
[
  {
    _id: 1,
    name: 'Alice',
    age: 25,
    major: 'Computer Science',
    scores: [ 85, 92, 78 ]
  },
  {
    _id: 3,
    name: 'Charlie',
    age: 28,
    major: 'English',
    scores: [ 75, 82, 89 ]
  }
]
```

2.$sort:

Input:

```
db> db.sstudent.aggregate([ {$sort:{age:-1}}] )
```

Output:

```
[
  {
    _id: 3,
    name: 'Charlie',
    age: 28,
    major: 'English',
    scores: [ 75, 82, 89 ]
  },
  {
    _id: 1,
    name: 'Alice',
    age: 25,
    major: 'Computer Science',
    scores: [ 85, 92, 78 ]
  },
  {
    _id: 5,
    name: 'Eve',
    age: 23,
    major: 'Biology',
    scores: [ 80, 77, 93 ]
  },
  {
    _id: 2,
    name: 'Bob',
    age: 22,
    major: 'Mathematics',
    scores: [ 90, 88, 95 ]
  },
  {
    _id: 4,
    name: 'David',
    age: 20,
    major: 'Computer Science',
    scores: [ 98, 95, 87 ]
  }
]
```

3.$project:

Input:

```
db> db.sstudent.aggregate([ {$project:{_id:0,name:1,age:1}}] )
```

Output:

```
[
  { name: 'Alice', age: 25 },
  { name: 'Bob', age: 22 },
  { name: 'Charlie', age: 28 },
  { name: 'David', age: 20 },
  { name: 'Eve', age: 23 }
]
```

4.$group:

Input:

```
db> db.sstudent.aggregate([
... {$group:{_id:"$major",averageAge:{$avg:"$age"},
... totalStudents:{$sum:1}}}
... ])
```

Output:

```
[
  { _id: 'Mathematics', averageAge: 22, totalStudents: 1 },
  { _id: 'English', averageAge: 28, totalStudents: 1 },
  { _id: 'Biology', averageAge: 23, totalStudents: 1 },
  { _id: 'Computer Science', averageAge: 22.5, totalStudents: 2 }
]
```

5.$skip:

Input:

```
db> db.sstudent.aggregate([{ $project: { _id: 0, name: 1, averageScore: { $avg: "$scores" } } }, { $match: { averageScore: { $gt: 65 } } }, { $skip: 1 }] )
```

Output:

```
[
  { name: 'Bob', averageScore: 91 },
  { name: 'Charlie', averageScore: 82 },
  { name: 'David', averageScore: 93.33333333333333 },
  { name: 'Eve', averageScore: 83.33333333333333 }
]
```