National University
Of Computer and Emerging Sciences

LAB # 11

Name – Muhammad Taha

Roll NO – 23P-0559

SECTION – BCS(4A)

Subject – Database Systems - LAB

1. Create a database named SchoolDB.

use SchoolDB

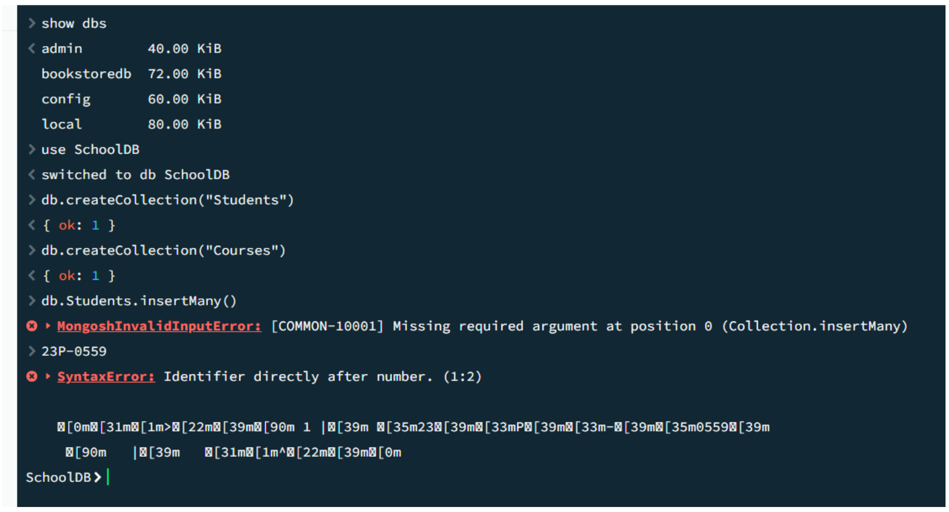
2. Create two collections:

o Students

o Courses

db.createCollection("Students")

db.createCollection("Courses")



```
> show dbs
< admin        40.00 KiB
  bookstoredb  72.00 KiB
  config       60.00 KiB
  local        80.00 KiB
> use SchoolDB
< switched to db SchoolDB
> db.createCollection("Students")
< { ok: 1 }
> db.createCollection("Courses")
< { ok: 1 }
> db.Students.insertMany()
✖ ▸ MongoshInvalidInputError: [COMMON-10001] Missing required argument at position 0 (Collection.insertMany)
> 23P-0559
✖ ▸ SyntaxError: Identifier directly after number. (1:2)

   ⊠[0m⊠[31m⊠[1m>⊠[22m⊠[39m⊠[90m 1 |⊠[39m ⊠[35m23⊠[39m⊠[33mP⊠[39m⊠[33m-⊠[39m⊠[35m0559⊠[39m
    ⊠[90m   |⊠[39m   ⊠[31m⊠[1m^⊠[22m⊠[39m⊠[0m
SchoolDB>
```

3. Insert the following documents into the Students collection:

Insert in the collection of the Students:

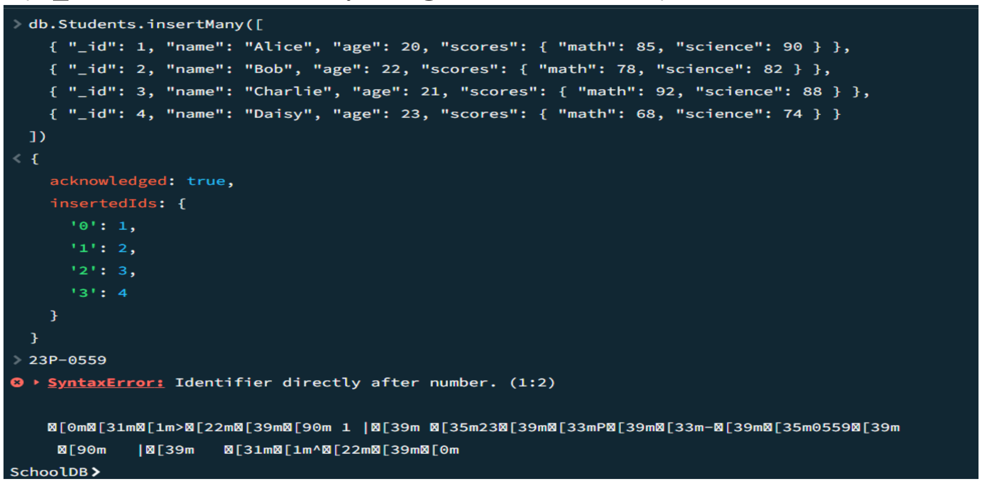
db.Students.insertMany([

{ "_id": 1, "name": "Alice", "age": 20, "scores": { "math": 85, "science": 90 } },

{ "_id": 2, "name": "Bob", "age": 22, "scores": { "math": 78, "science": 82 } },

{ "_id": 3, "name": "Charlie", "age": 21, "scores": { "math": 92, "science": 88 } },

{ "_id": 4, "name": "Daisy", "age": 23, "scores": { "math": 68, "science": 74 } }])



```
> db.Students.insertMany([
    { "_id": 1, "name": "Alice", "age": 20, "scores": { "math": 85, "science": 90 } },
    { "_id": 2, "name": "Bob", "age": 22, "scores": { "math": 78, "science": 82 } },
    { "_id": 3, "name": "Charlie", "age": 21, "scores": { "math": 92, "science": 88 } },
    { "_id": 4, "name": "Daisy", "age": 23, "scores": { "math": 68, "science": 74 } }
  ])
< {
    acknowledged: true,
    insertedIds: {
      '0': 1,
      '1': 2,
      '2': 3,
      '3': 4
    }
  }
> 23P-0559
✖ ▸ SyntaxError: Identifier directly after number. (1:2)

    ⊠[0m⊠[31m⊠[1m>⊠[22m⊠[39m⊠[90m 1 |⊠[39m ⊠[35m23⊠[39m⊠[33mP⊠[39m⊠[33m-⊠[39m⊠[35m0559⊠[39m
     ⊠[90m   |⊠[39m   ⊠[31m⊠[1m^⊠[22m⊠[39m⊠[0m
SchoolDB>
```

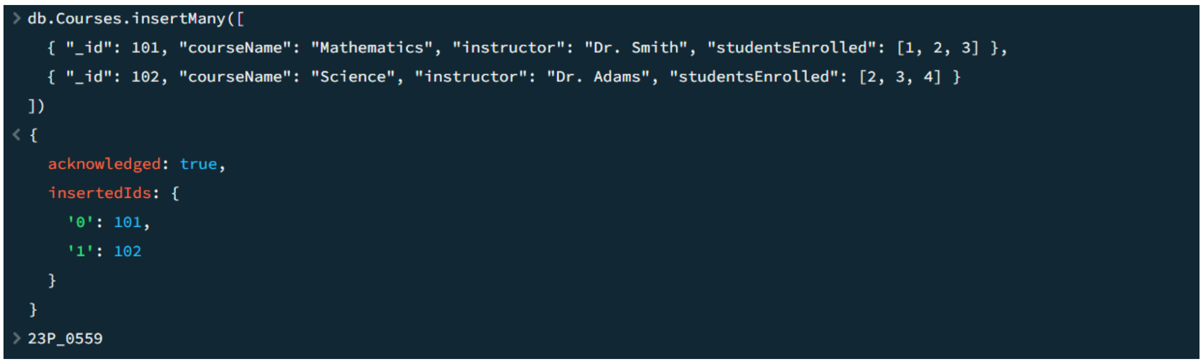
Insert in the collection of the Course:

db.Courses.insertMany([

{ "_id": 101, "courseName": "Mathematics", "instructor": "Dr. Smith", "studentsEnrolled": [1, 2, 3] },

{ "_id": 102, "courseName": "Science", "instructor": "Dr. Adams", "studentsEnrolled": [2, 3, 4] }

])



```
> db.Courses.insertMany([
    { "_id": 101, "courseName": "Mathematics", "instructor": "Dr. Smith", "studentsEnrolled": [1, 2, 3] },
    { "_id": 102, "courseName": "Science", "instructor": "Dr. Adams", "studentsEnrolled": [2, 3, 4] }
  ])
< {
    acknowledged: true,
    insertedIds: {
      '0': 101,
      '1': 102
    }
  }
> 23P_0559
```

5. Use findOne to retrieve:

A student where the math score is &gt;= 85 and the age is &lt; 22.

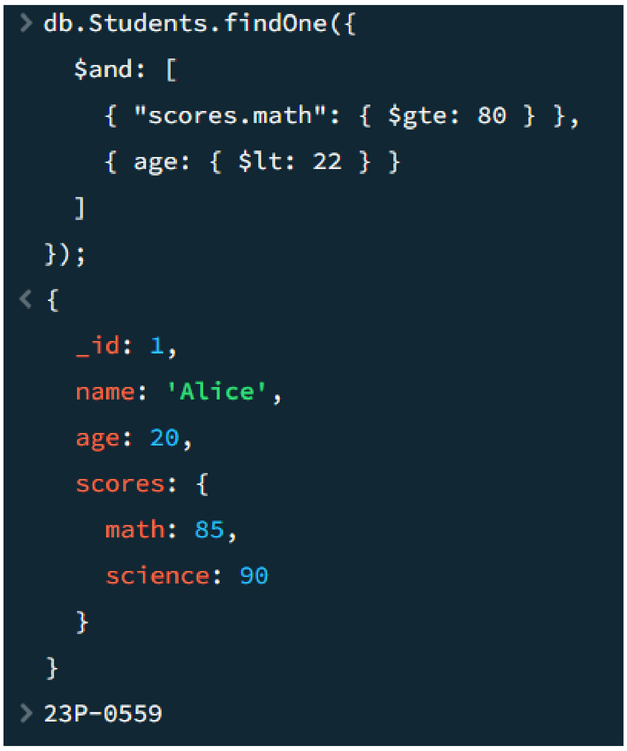
db.Students.findOne({

$and: [

{ "scores.math": { $gte: 80 } },

{ age: { $lt: 22 } }]

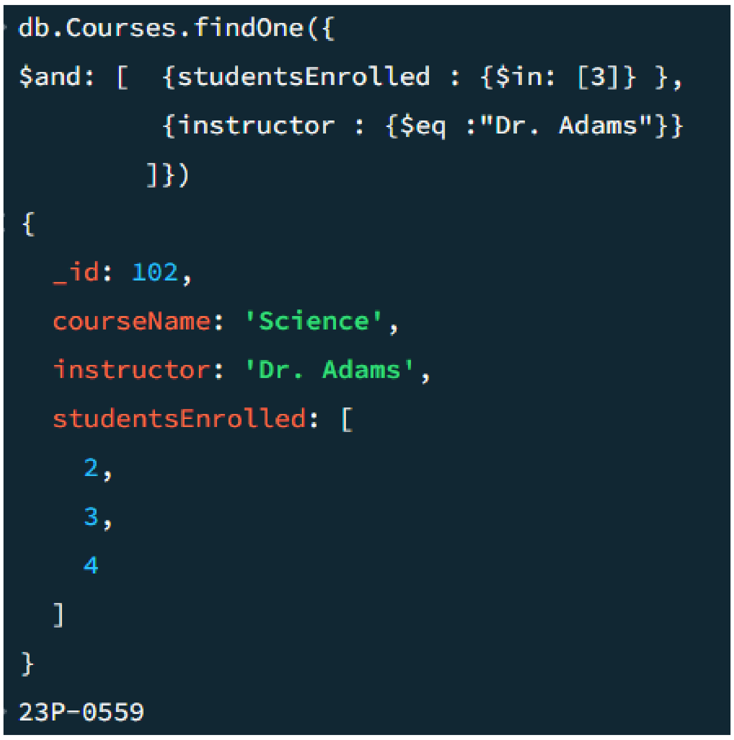
});



```
> db.Students.findOne({
    $and: [
      { "scores.math": { $gte: 80 } },
      { age: { $lt: 22 } }
    ]
  });
< {
    _id: 1,
    name: 'Alice',
    age: 20,
    scores: {
      math: 85,
      science: 90
    }
  }
> 23P-0559
```

A course where the studentsEnrolled array includes 3 and the instructor is &quot;Dr. Adams&quot;.

```
db.Courses.findOne({
$and: [  {studentsEnrolled : {$in: [3]} },
        {instructor : {$eq :"Dr. Adams"}}
       ]})
```

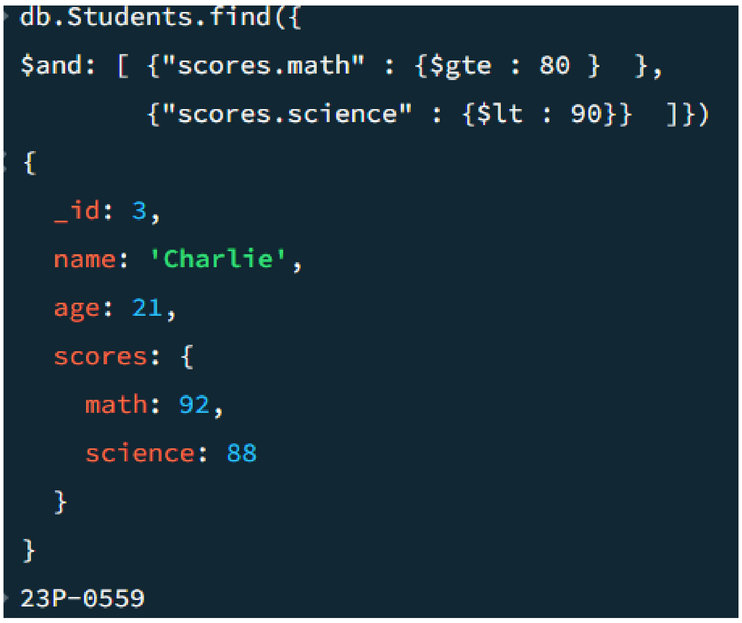


```
db.Courses.findOne({
$and: [  {studentsEnrolled : {$in: [3]} },
         {instructor : {$eq :"Dr. Adams"}}
        ]})
{
  _id: 102,
  courseName: 'Science',
  instructor: 'Dr. Adams',
  studentsEnrolled: [
    2,
    3,
    4
  ]
}
23P-0559
```

6. Use find to retrieve:

Students with math score &gt;= 80 and science score &lt; 90.

```
db.Students.find({
$and: [ {"scores.math" : {$gte : 80 }  },
       {"scores.science" : {$lt : 90}}  ]})
```



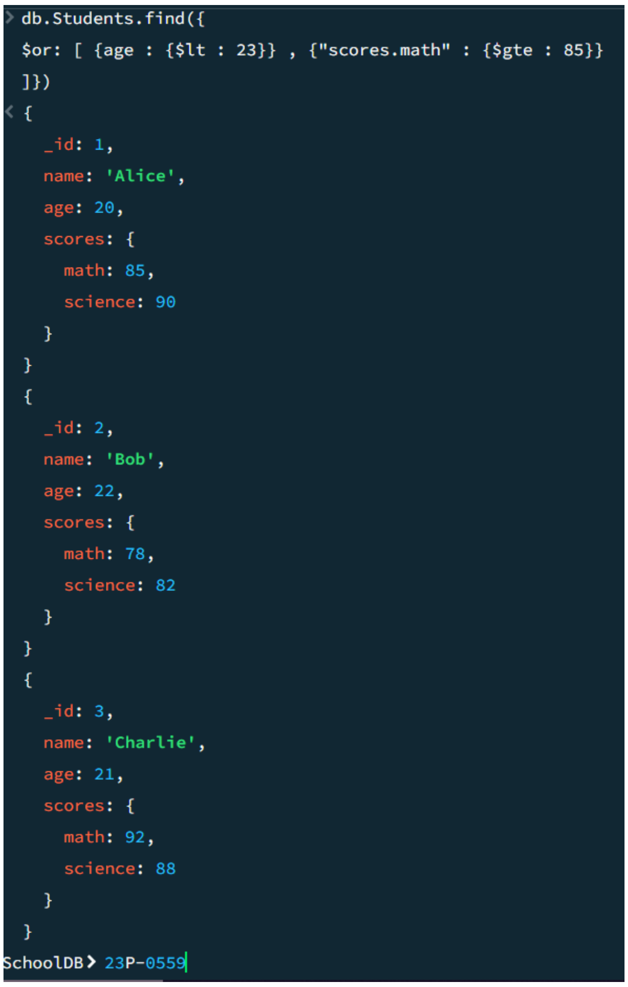
```
db.Students.find({
$and: [ {"scores.math" : {$gte : 80 }  },
        {"scores.science" : {$lt : 90}}  ]})
{
  _id: 3,
  name: 'Charlie',
  age: 21,
  scores: {
    math: 92,
    science: 88
  }
}
23P-0559
```

Students whose age is &lt; 23 or have a math score &gt;= 85.

db.Students.find({

$or: [{age : {$lt : 23}} , {"scores.math" : {$gte : 85}}

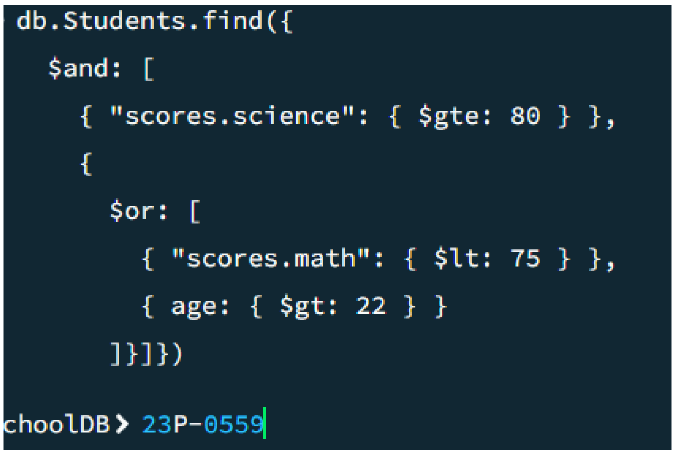
]})



```
> db.Students.find({
$or: [ {age : {$lt : 23}} , {"scores.math" : {$gte : 85}}
]})
< {
    _id: 1,
    name: 'Alice',
    age: 20,
    scores: {
      math: 85,
      science: 90
    }
  }
  {
    _id: 2,
    name: 'Bob',
    age: 22,
    scores: {
      math: 78,
      science: 82
    }
  }
  {
    _id: 3,
    name: 'Charlie',
    age: 21,
    scores: {
      math: 92,
      science: 88
    }
  }
SchoolDB> 23P-0559
```

Students with science score &gt;= 80 and (either math score &lt; 75 or age &gt; 22).

```
db.Students.find({
 $and: [
  { "scores.science": { $gte: 80 } },
  {
   $or: [
    { "scores.math": { $lt: 75 } },
    { age: { $gt: 22 } }
   ]}]})
```



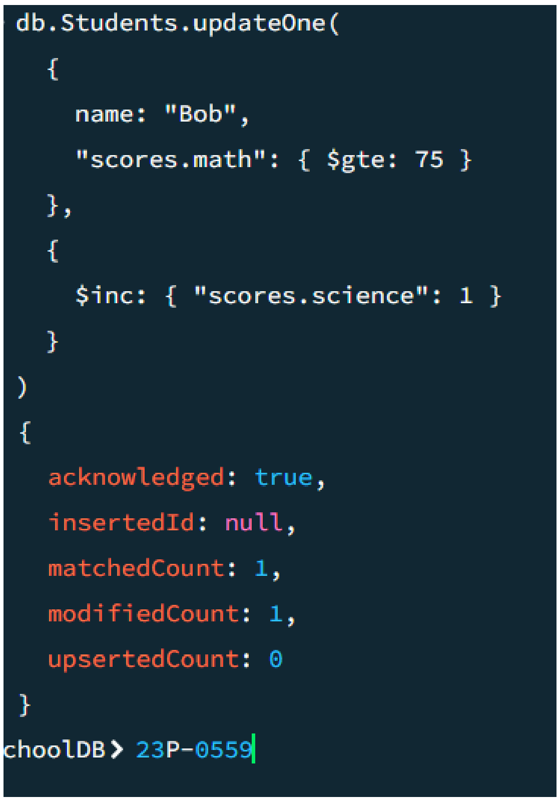
```
db.Students.find({
  $and: [
    { "scores.science": { $gte: 80 } },
    {
      $or: [
        { "scores.math": { $lt: 75 } },
        { age: { $gt: 22 } }
      ]}]})

choolDB> 23P-0559
```

7. Use updateOne to:

Increase the science score of the student where name is &quot;Bob&quot; and math score is &gt;= 75.

```
db.Students.updateOne(
 {name: "Bob",
  "scores.math": { $gte: 75 }
 },{
  $inc: { "scores.science": 1 }
 })
```



```
db.Students.updateOne(
  {
    name: "Bob",
    "scores.math": { $gte: 75 }
  },
  {
    $inc: { "scores.science": 1 }
  }
)
{
  acknowledged: true,
  insertedId: null,
  matchedCount: 1,
  modifiedCount: 1,
  upsertedCount: 0
}
choolDB> 23P-0559
```

8. Use updateMany to:

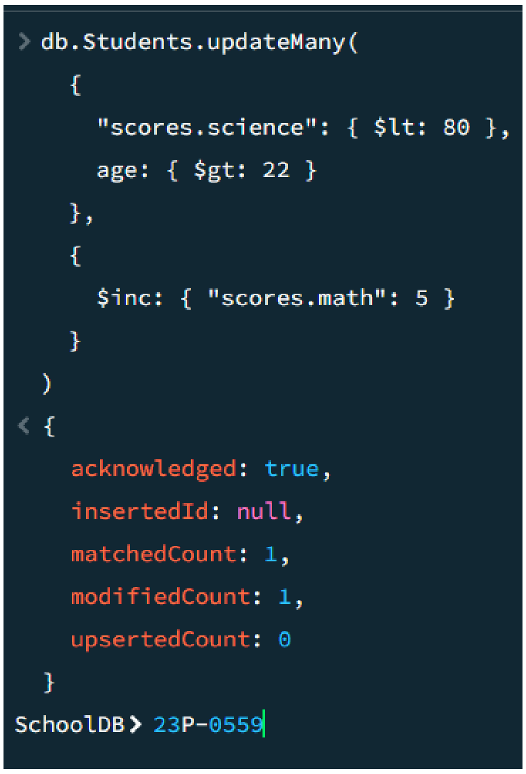
Increase the math score by 5 for students whose science score is &lt; 80 and age &gt; 22.

db.Students.updateMany(

{"scores.science": { $lt: 80 },

age: { $gt: 22 }},

{$inc: { "scores.math": 5 }})

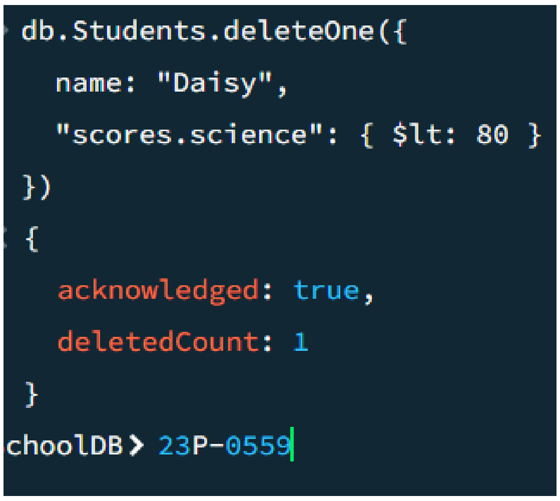


```
> db.Students.updateMany(
    {
      "scores.science": { $lt: 80 },
      age: { $gt: 22 }
    },
    {
      $inc: { "scores.math": 5 }
    }
  )
< {
    acknowledged: true,
    insertedId: null,
    matchedCount: 1,
    modifiedCount: 1,
    upsertedCount: 0
  }
SchoolDB> 23P-0559
```

9 Use deleteOne to:

o Remove a student where name is &quot;Daisy&quot; and their science score is &lt; 80.

```
db.Students.deleteOne({
  name: "Daisy",
  "scores.science": { $lt: 80 }
})
```

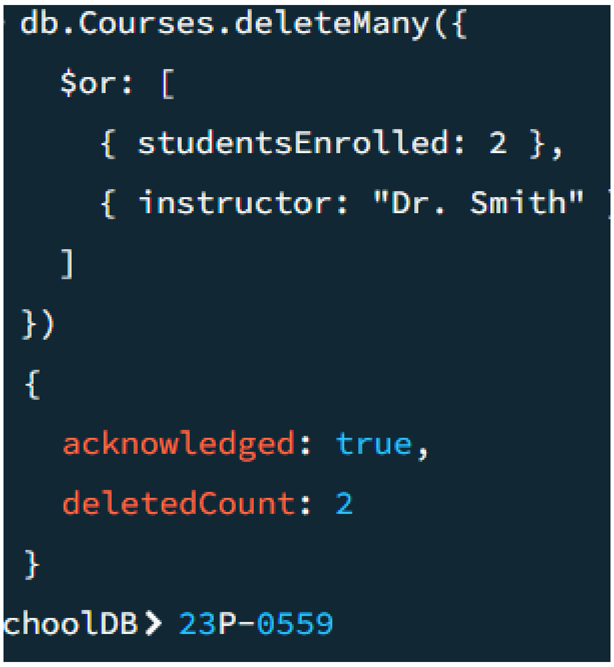


```
db.Students.deleteOne({
  name: "Daisy",
  "scores.science": { $lt: 80 }
})
{
  acknowledged: true,
  deletedCount: 1
}
choolDB> 23P-0559
```

10. Use deleteMany to:

o Remove courses where the studentsEnrolled array includes 2 or the instructor is &quot;Dr. Smith&quot;.

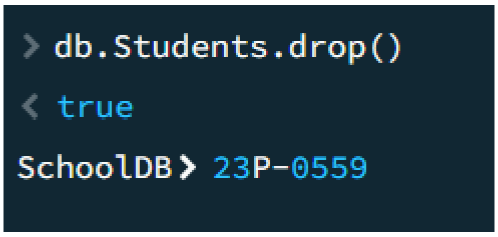
```
db.Courses.deleteMany({
  $or: [
    { studentsEnrolled: 2 },
    { instructor: "Dr. Smith" }
  ]
})
```



```
db.Courses.deleteMany({
  $or: [
    { studentsEnrolled: 2 },
    { instructor: "Dr. Smith"
  ]
})
{
  acknowledged: true,
  deletedCount: 2
}
choolDB> 23P-0559
```

11. Drop the Students collection.

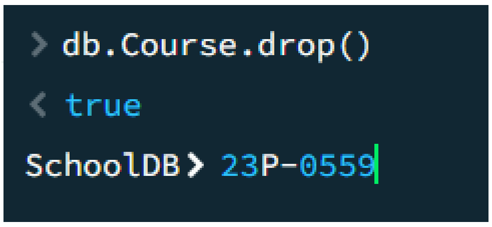
db.Students.drop()



```
> db.Students.drop()
< true
SchoolDB> 23P-0559
```

12. Drop the Courses collection.

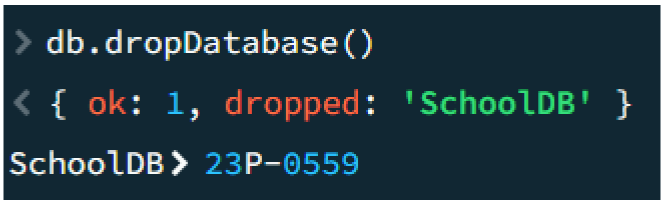
db.Students.drop()



```
> db.Course.drop()
< true
SchoolDB> 23P-0559
```

13. Finally, delete the SchoolDB database.

db.dropDatabase()



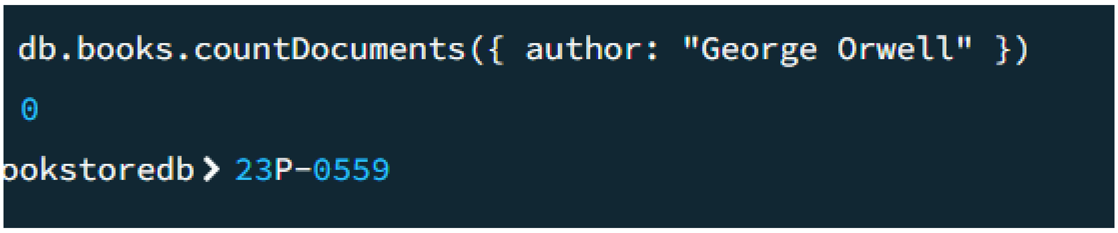
```
> db.dropDatabase()
< { ok: 1, dropped: 'SchoolDB' }
SchoolDB> 23P-0559
```

Part 2

1 Count Books by a Specific Author

□ Count the number of books written by &quot;George Orwell.&quot;

db.books.countDocuments({ author: "George Orwell" })

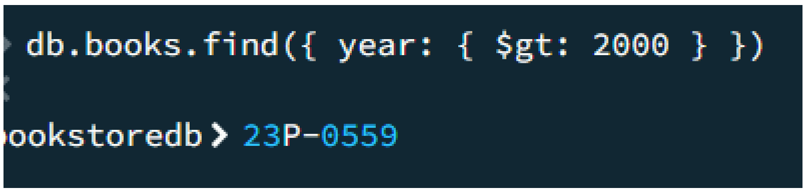


```
db.books.countDocuments({ author: "George Orwell" })
0
ookstoredb> 23P-0559
```

2. Find Books Published After a Certain Year

□ Retrieve all books published after the year 2000.

db.books.find({ year: { $gt: 2000 } })



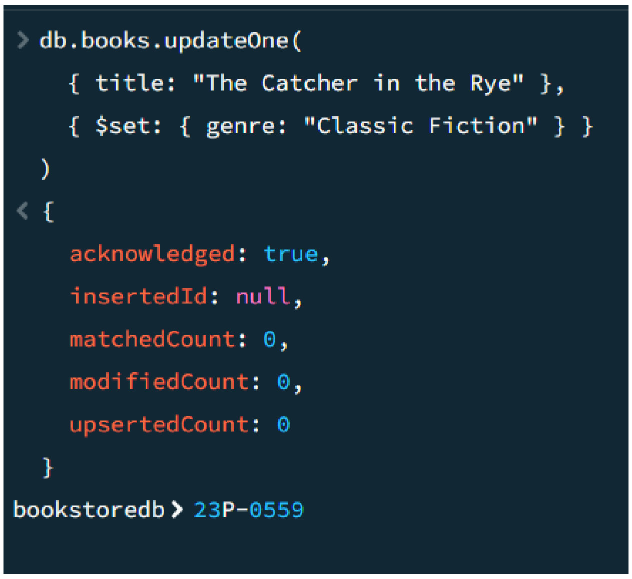
```
db.books.find({ year: { $gt: 2000 } })

ookstoredb> 23P-0559
```

3 Update the Genre of a Book

□ Change the genre of &quot;The Catcher in the Rye&quot; to &quot;Classic Fiction.&quot;

```
db.books.updateOne(
  { title: "The Catcher in the Rye" },
  { $set: { genre: "Classic Fiction" } }
)
```



```
> db.books.updateOne(
    { title: "The Catcher in the Rye" },
    { $set: { genre: "Classic Fiction" } }
  )
< {
    acknowledged: true,
    insertedId: null,
    matchedCount: 0,
    modifiedCount: 0,
    upsertedCount: 0
  }
bookstoredb > 23P-0559
```

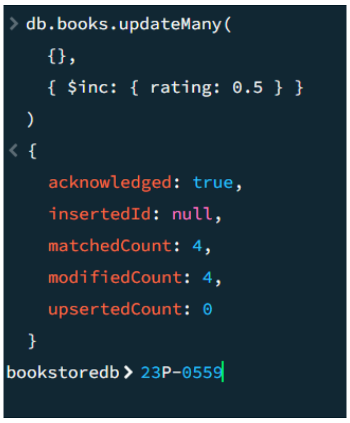
4. Increase Rating for All Books by 0.5

□ Increase the rating field of all books by 0.5 points.

db.books.updateMany(

{},

{ $inc: { rating: 0.5 } })



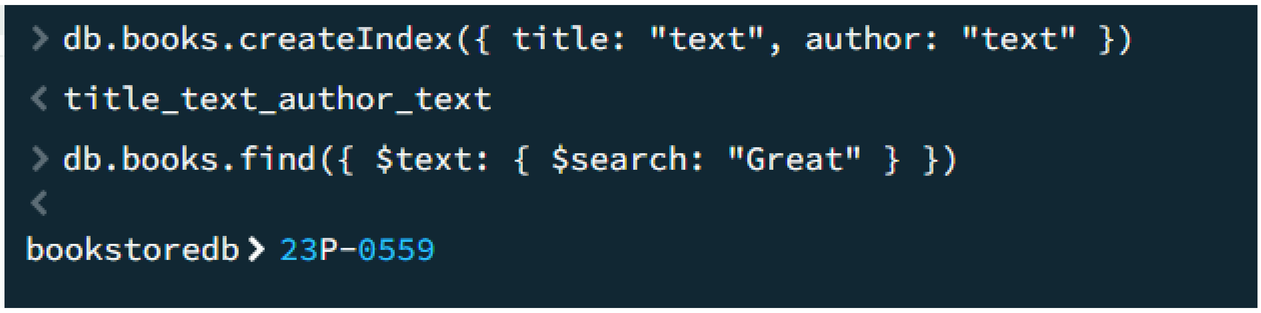
```
> db.books.updateMany(
    {},
    { $inc: { rating: 0.5 } }
  )
< {
    acknowledged: true,
    insertedId: null,
    matchedCount: 4,
    modifiedCount: 4,
    upsertedCount: 0
  }
bookstoredb > 23P-0559
```

5. Find Books Matching a Keyword

□ Perform a text search for books that contain the keyword &quot;Great&quot; in the title or author.

db.books.createIndex({ title: "text", author: "text" })

db.books.find({ $text: { $search: "Great" } })

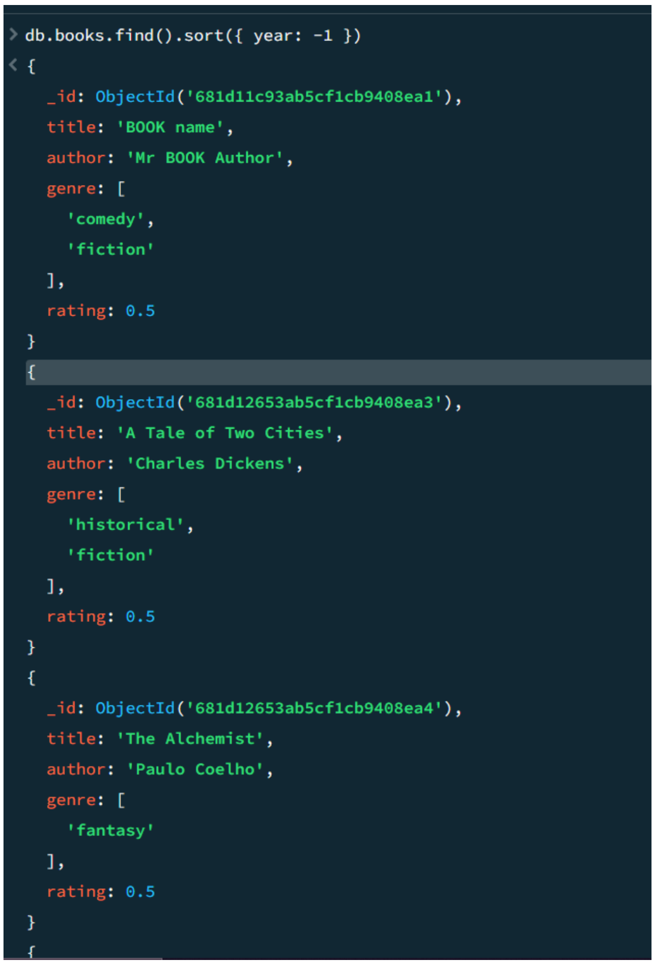


```
> db.books.createIndex({ title: "text", author: "text" })
< title_text_author_text
> db.books.find({ $text: { $search: "Great" } })
<
bookstoredb > 23P-0559
```

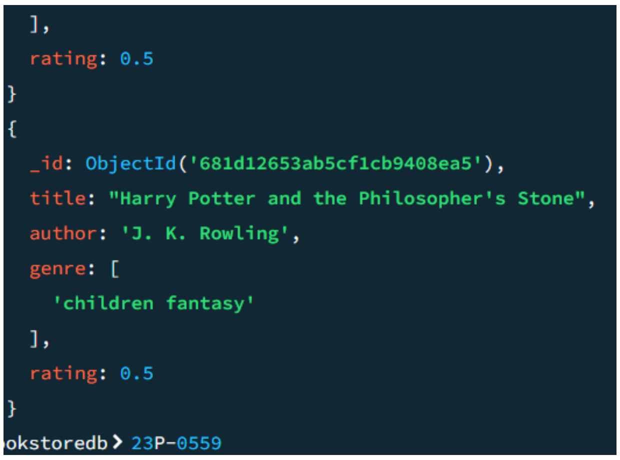
6. Sort Books by Publication Year

 Retrieve all books, sorted in descending order by publication year.

db.books.find().sort({ year: -1 })



```
> db.books.find().sort({ year: -1 })
< {
    _id: ObjectId('681d11c93ab5cf1cb9408ea1'),
    title: 'BOOK name',
    author: 'Mr BOOK Author',
    genre: [
      'comedy',
      'fiction'
    ],
    rating: 0.5
  }
  {
    _id: ObjectId('681d12653ab5cf1cb9408ea3'),
    title: 'A Tale of Two Cities',
    author: 'Charles Dickens',
    genre: [
      'historical',
      'fiction'
    ],
    rating: 0.5
  }
  {
    _id: ObjectId('681d12653ab5cf1cb9408ea4'),
    title: 'The Alchemist',
    author: 'Paulo Coelho',
    genre: [
      'fantasy'
    ],
    rating: 0.5
  }
  {
```

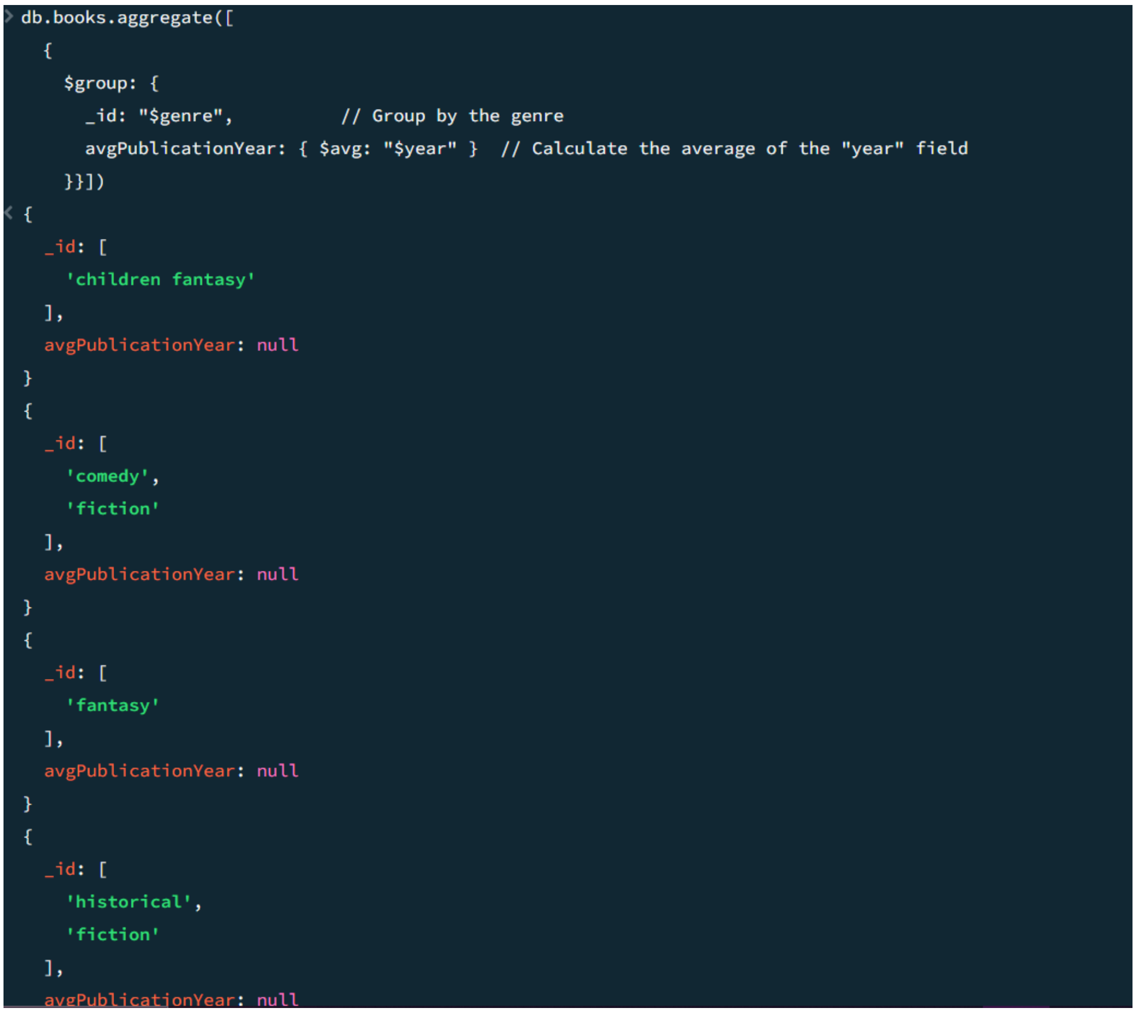


```
  ],
  rating: 0.5
}
{
  _id: ObjectId('681d12653ab5cf1cb9408ea5'),
  title: "Harry Potter and the Philosopher's Stone",
  author: 'J. K. Rowling',
  genre: [
    'children fantasy'
  ],
  rating: 0.5
}
okstoredb > 23P-0559
```

7. Get the Average Publication Year by Genre

 Calculate the average publication year of books for each genre.

```
db.books.aggregate([
 {
   $group: {
     _id: "$genre",        // Group by the genre
     avgPublicationYear: { $avg: "$year" }  // Calculate the average of the "year" field }}])
```

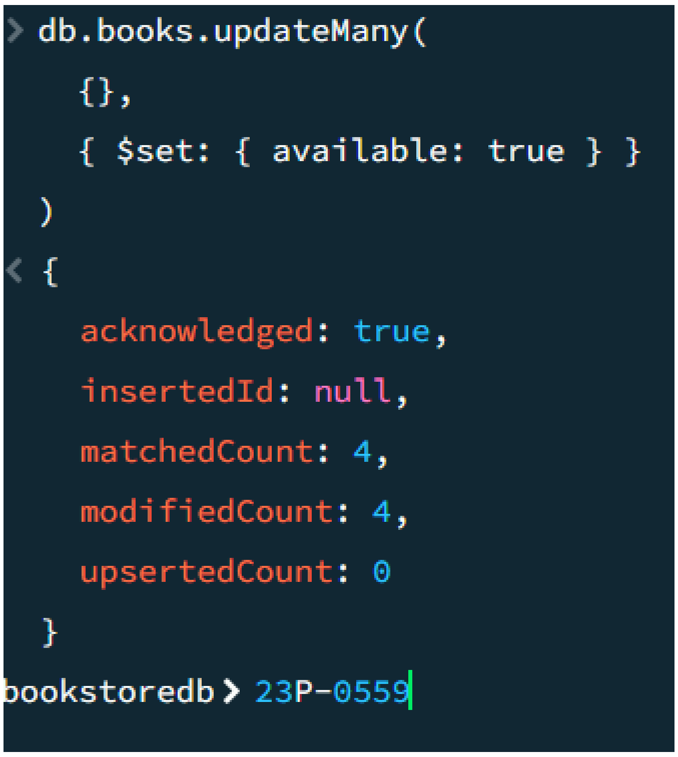


```
> db.books.aggregate([
    {
      $group: {
        _id: "$genre",          // Group by the genre
        avgPublicationYear: { $avg: "$year" }  // Calculate the average of the "year" field
      }}])
< {
    _id: [
      'children fantasy'
    ],
    avgPublicationYear: null
  }
  {
    _id: [
      'comedy',
      'fiction'
    ],
    avgPublicationYear: null
  }
  {
    _id: [
      'fantasy'
    ],
    avgPublicationYear: null
  }
  {
    _id: [
      'historical',
      'fiction'
    ],
    avgPublicationYear: null
```

8. Add a New Field to All Documents

 Add a new field available (boolean) set to true for all books.

```
db.books.updateMany(
  {},
  { $set: { available: true } }
)
```

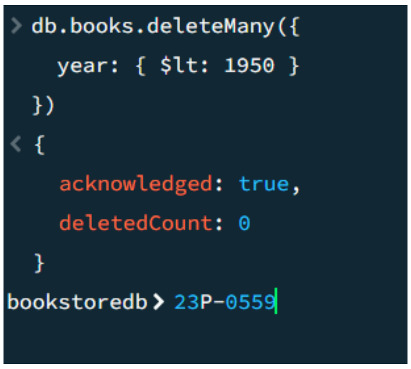


```
> db.books.updateMany(
    {},
    { $set: { available: true } }
  )
< {
    acknowledged: true,
    insertedId: null,
    matchedCount: 4,
    modifiedCount: 4,
    upsertedCount: 0
  }
bookstoredb> 23P-0559
```

9. Delete Books Published Before a Certain Year

 Delete all books published before the year 1950.

```
db.books.deleteMany({
  year: { $lt: 1950 }
})
```

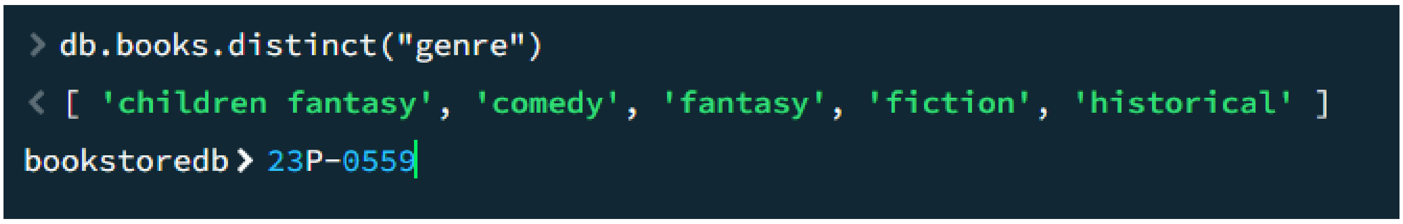


```
> db.books.deleteMany({
    year: { $lt: 1950 }
  })
< {
    acknowledged: true,
    deletedCount: 0
  }
bookstoredb> 23P-0559
```

10. List All Unique Genres

 Retrieve a list of all unique genres in the collection without duplicates.

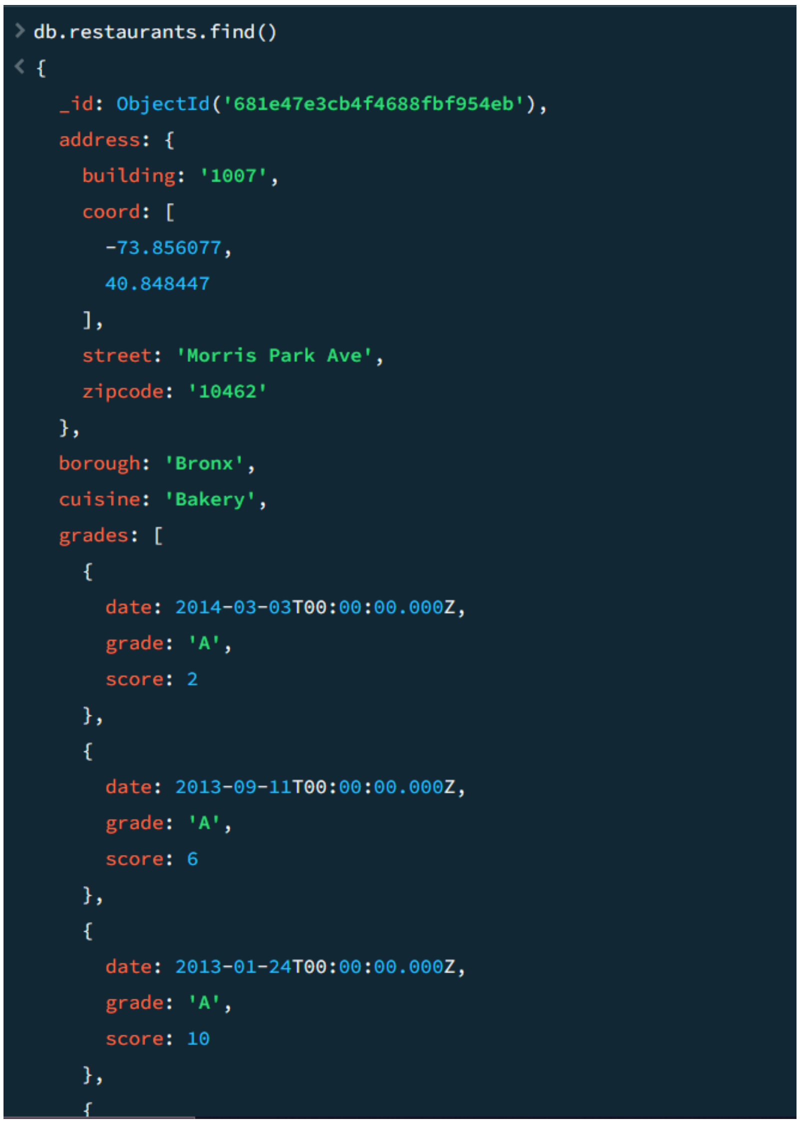
db.books.distinct("genre")



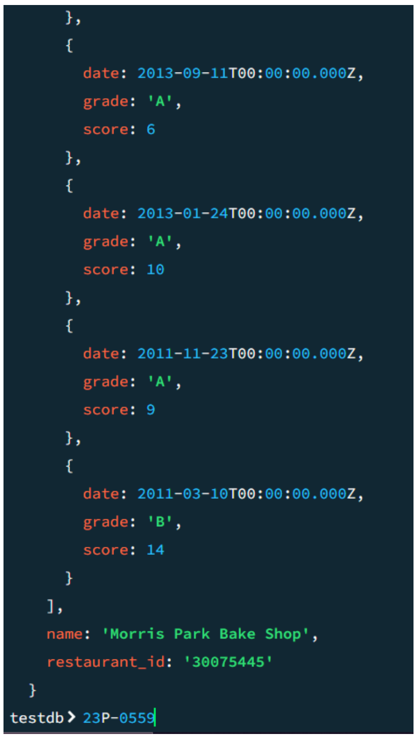
```
> db.books.distinct("genre")
< [ 'children fantasy', 'comedy', 'fantasy', 'fiction', 'historical' ]
bookstoredb> 23P-0559
```

Additional Tasks

1. Write a MongoDB query to display all the documents in the collection restaurants.



```
> db.restaurants.find()
< {
    _id: ObjectId('681e47e3cb4f4688fbf954eb'),
    address: {
      building: '1007',
      coord: [
        -73.856077,
        40.848447
      ],
      street: 'Morris Park Ave',
      zipcode: '10462'
    },
    borough: 'Bronx',
    cuisine: 'Bakery',
    grades: [
      {
        date: 2014-03-03T00:00:00.000Z,
        grade: 'A',
        score: 2
      },
      {
        date: 2013-09-11T00:00:00.000Z,
        grade: 'A',
        score: 6
      },
      {
        date: 2013-01-24T00:00:00.000Z,
        grade: 'A',
        score: 10
      },
      {
```



```
    },
    {
      date: 2013-09-11T00:00:00.000Z,
      grade: 'A',
      score: 6
    },
    {
      date: 2013-01-24T00:00:00.000Z,
      grade: 'A',
      score: 10
    },
    {
      date: 2011-11-23T00:00:00.000Z,
      grade: 'A',
      score: 9
    },
    {
      date: 2011-03-10T00:00:00.000Z,
      grade: 'B',
      score: 14
    }
  ],
  name: 'Morris Park Bake Shop',
  restaurant_id: '30075445'
}
testdb> 23P-0559
```

2. Write a MongoDB query to display the fields restaurant_id, name, borough and cuisine for all the

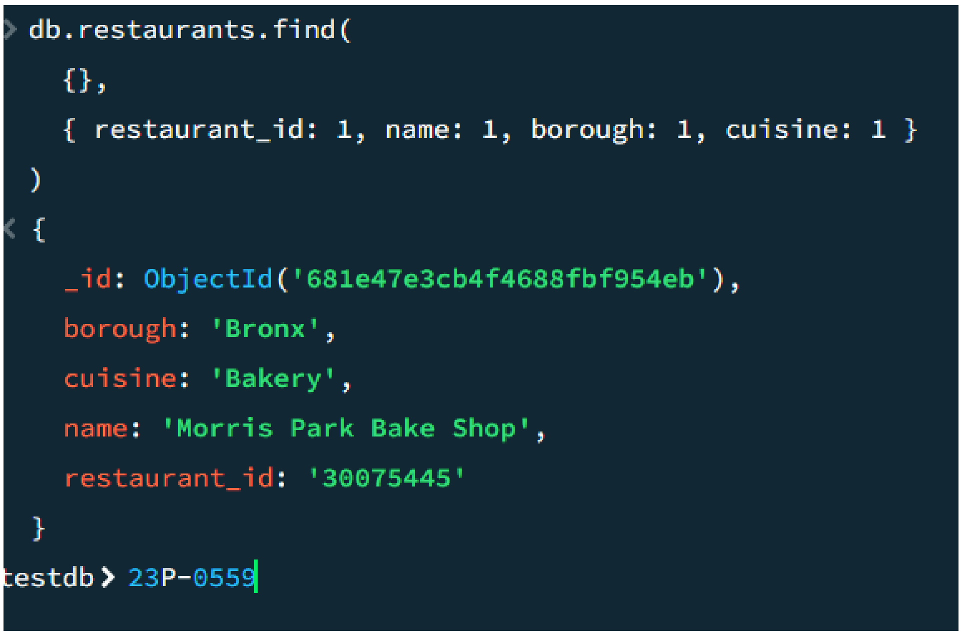
documents in the collection restaurant.

db.restaurants.find(

{},

{ restaurant_id: 1, name: 1, borough: 1, cuisine: 1 }

)



```
> db.restaurants.find(
    {},
    { restaurant_id: 1, name: 1, borough: 1, cuisine: 1 }
  )
< {
    _id: ObjectId('681e47e3cb4f4688fbf954eb'),
    borough: 'Bronx',
    cuisine: 'Bakery',
    name: 'Morris Park Bake Shop',
    restaurant_id: '30075445'
  }
testdb> 23P-0559
```

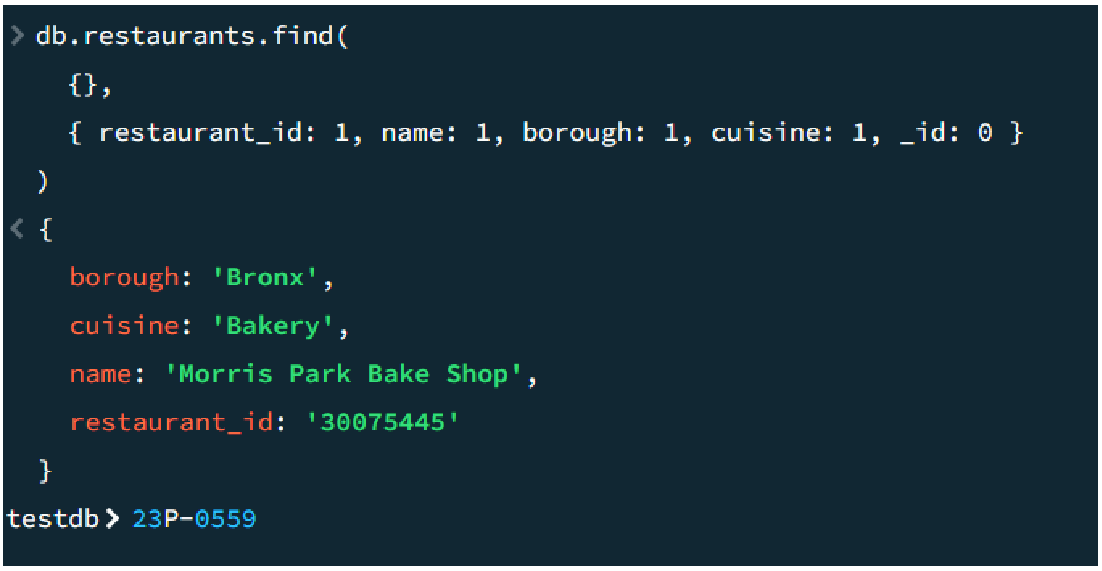
3. Write a MongoDB query to display the fields restaurant_id, name, borough and cuisine, but exclude the field _id for all the documents in the collection restaurant.

db.restaurants.find(

{},

{ restaurant_id: 1, name: 1, borough: 1, cuisine: 1, _id: 0 }

)



```
> db.restaurants.find(
    {},
    { restaurant_id: 1, name: 1, borough: 1, cuisine: 1, _id: 0 }
  )
< {
    borough: 'Bronx',
    cuisine: 'Bakery',
    name: 'Morris Park Bake Shop',
    restaurant_id: '30075445'
  }
testdb> 23P-0559
```

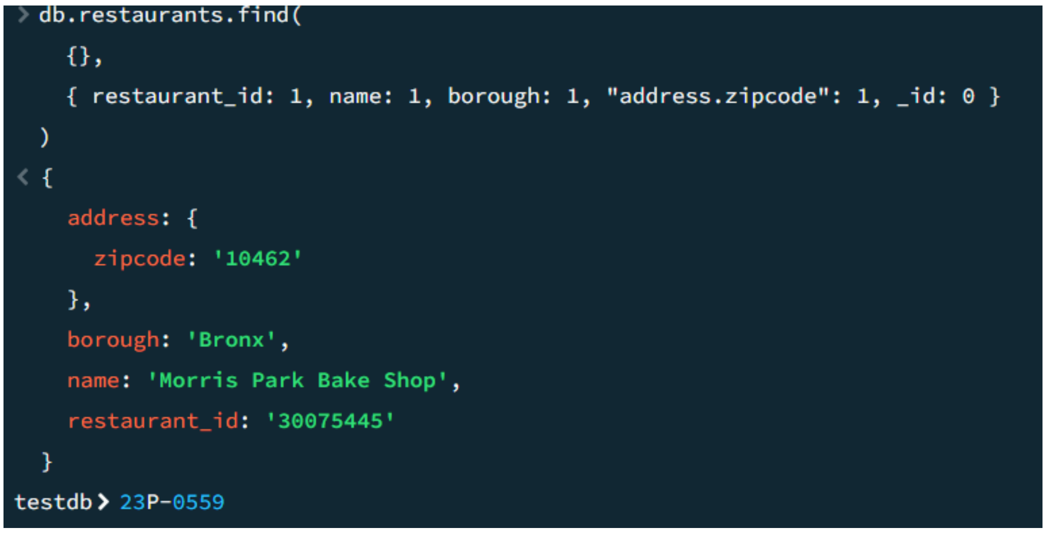
4. Write a MongoDB query to display the fields restaurant_id, name, borough and zip code, but exclude the field _id for all the documents in the collection restaurant.

db.restaurants.find(

{},

{ restaurant_id: 1, name: 1, borough: 1, "address.zipcode": 1, _id: 0 }

)



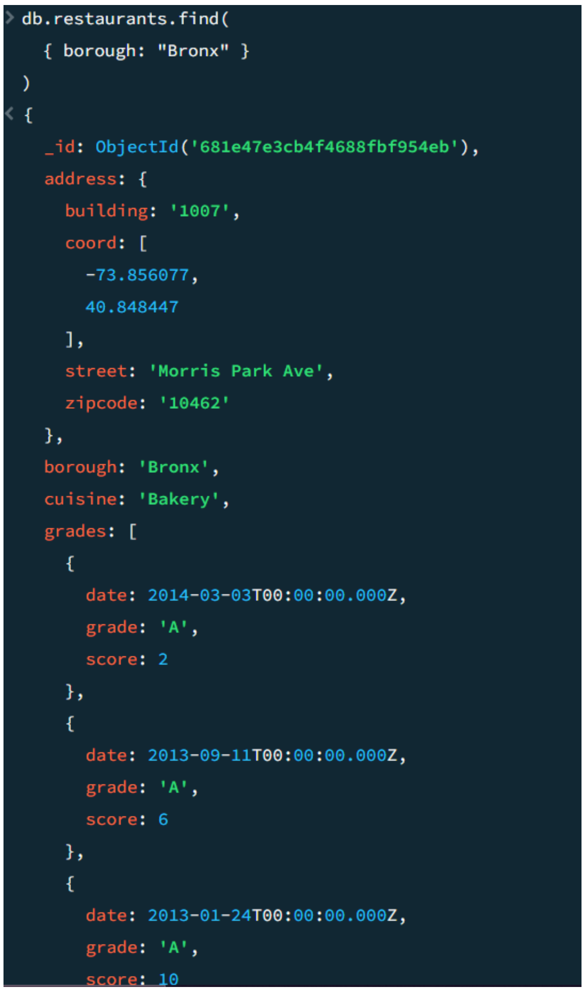
```
> db.restaurants.find(
    {},
    { restaurant_id: 1, name: 1, borough: 1, "address.zipcode": 1, _id: 0 }
  )
< {
    address: {
      zipcode: '10462'
    },
    borough: 'Bronx',
    name: 'Morris Park Bake Shop',
    restaurant_id: '30075445'
  }
testdb> 23P-0559
```

5. Write a MongoDB query to display all the restaurant which is in the borough Bronx.

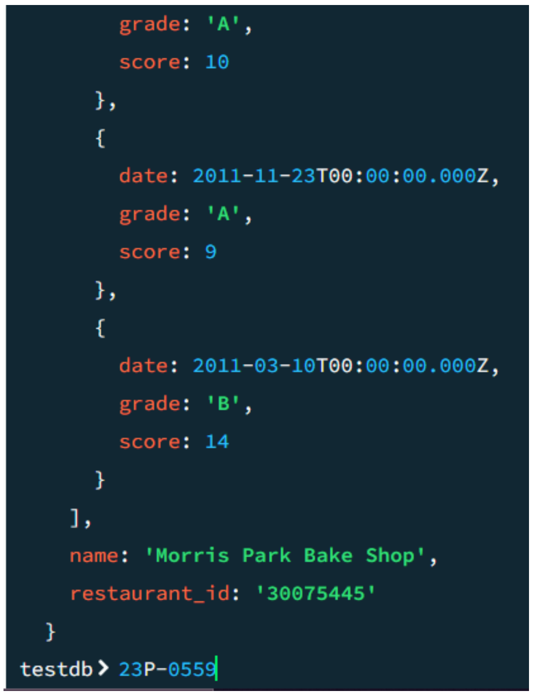
db.restaurants.find(

{ borough: "Bronx" }

)



```
> db.restaurants.find(
    { borough: "Bronx" }
  )
< {
    _id: ObjectId('681e47e3cb4f4688fbf954eb'),
    address: {
      building: '1007',
      coord: [
        -73.856077,
        40.848447
      ],
      street: 'Morris Park Ave',
      zipcode: '10462'
    },
    borough: 'Bronx',
    cuisine: 'Bakery',
    grades: [
      {
        date: 2014-03-03T00:00:00.000Z,
        grade: 'A',
        score: 2
      },
      {
        date: 2013-09-11T00:00:00.000Z,
        grade: 'A',
        score: 6
      },
      {
        date: 2013-01-24T00:00:00.000Z,
        grade: 'A',
        score: 10
```



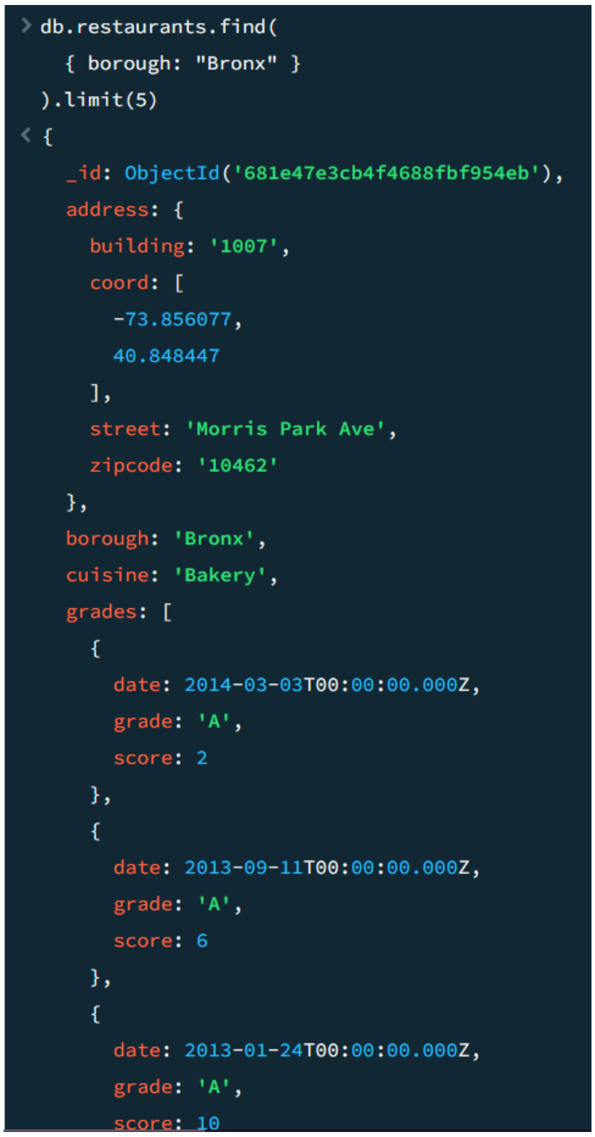
```
        grade: 'A',
        score: 10
      },
      {
        date: 2011-11-23T00:00:00.000Z,
        grade: 'A',
        score: 9
      },
      {
        date: 2011-03-10T00:00:00.000Z,
        grade: 'B',
        score: 14
      }
    ],
    name: 'Morris Park Bake Shop',
    restaurant_id: '30075445'
  }
testdb> 23P-0559
```

6. Write a MongoDB query to display the first 5 restaurant which is in the borough Bronx.

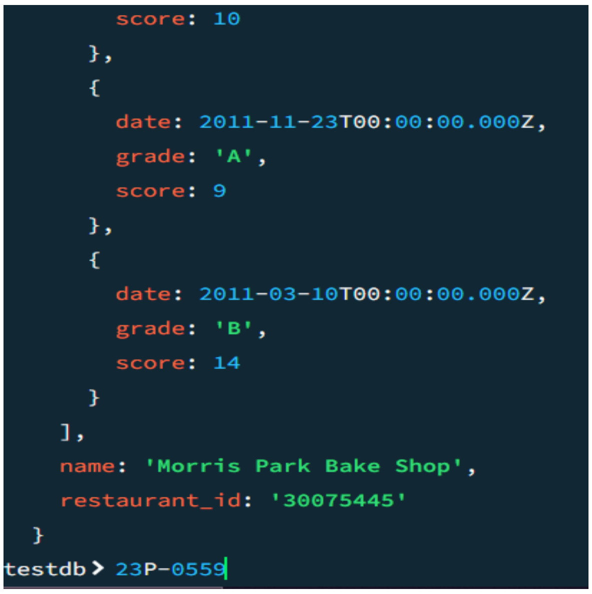
db.restaurants.find(

{ borough: "Bronx" }

).limit(5)



```
> db.restaurants.find(
    { borough: "Bronx" }
  ).limit(5)
< {
    _id: ObjectId('681e47e3cb4f4688fbf954eb'),
    address: {
      building: '1007',
      coord: [
        -73.856077,
        40.848447
      ],
      street: 'Morris Park Ave',
      zipcode: '10462'
    },
    borough: 'Bronx',
    cuisine: 'Bakery',
    grades: [
      {
        date: 2014-03-03T00:00:00.000Z,
        grade: 'A',
        score: 2
      },
      {
        date: 2013-09-11T00:00:00.000Z,
        grade: 'A',
        score: 6
      },
      {
        date: 2013-01-24T00:00:00.000Z,
        grade: 'A',
        score: 10
```



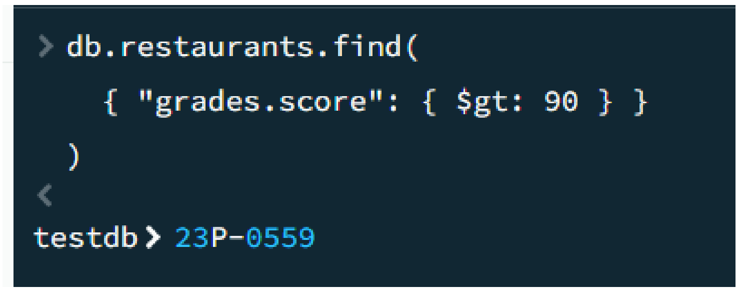
```
        score: 10
      },
      {
        date: 2011-11-23T00:00:00.000Z,
        grade: 'A',
        score: 9
      },
      {
        date: 2011-03-10T00:00:00.000Z,
        grade: 'B',
        score: 14
      }
    ],
    name: 'Morris Park Bake Shop',
    restaurant_id: '30075445'
  }
testdb> 23P-0559
```

7. **Write a MongoDB query to find the restaurants who achieved a score more than 90**

   db.restaurants.find(

   { "grades.score": { $gt: 90 } }

)



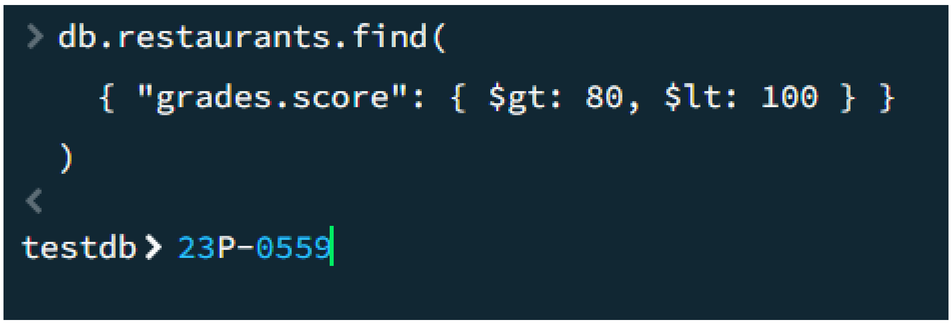
```
> db.restaurants.find(
    { "grades.score": { $gt: 90 } }
  )
<
testdb > 23P-0559
```

8. Write a MongoDB query to find the restaurants that achieved a score, more than 80 but less than 100.

db.restaurants.find(

{ "grades.score": { $gt: 80, $lt: 100 } }

)



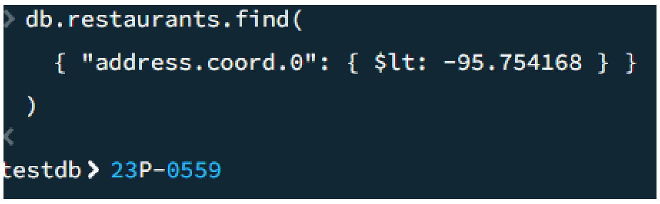
```
> db.restaurants.find(
    { "grades.score": { $gt: 80, $lt: 100 } }
  )
<
testdb > 23P-0559
```

9. **Write a MongoDB query to find the restaurants which locate in latitude value less than -95.754168.**

   db.restaurants.find(

   { "address.coord.0": { $lt: -95.754168 } }

)



```
> db.restaurants.find(
    { "address.coord.0": { $lt: -95.754168 } }
  )
<
testdb > 23P-0559
```

10. **Write a MongoDB query to find the restaurants which belong to the borough Bronx and prepared either American or Chinese dish.**

db.restaurants.find(

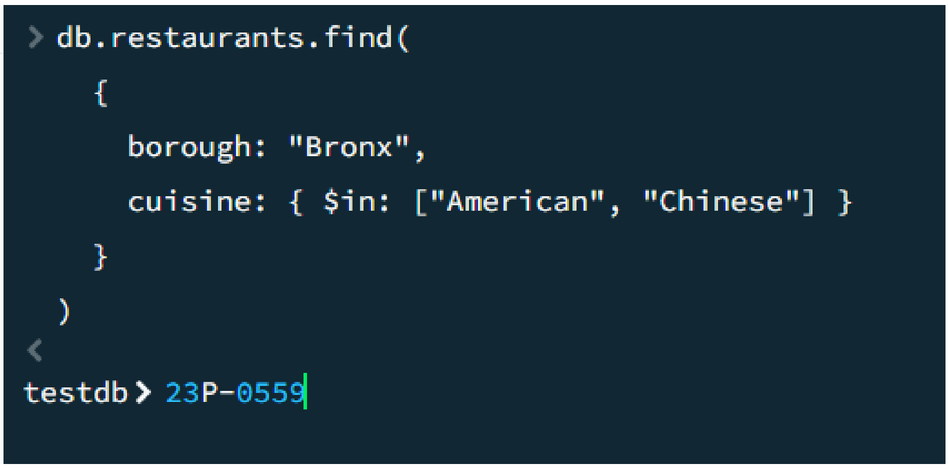
{

borough: "Bronx",

cuisine: { $in: ["American", "Chinese"] }

}

)



```
> db.restaurants.find(
    {
      borough: "Bronx",
      cuisine: { $in: ["American", "Chinese"] }
    }
  )
<
testdb> 23P-0559
```