Experiment 2:

a. **Develop a MongoDB query to select certain fields and ignore some fields of the documents from any collection.**
b. **Develop a MongoDB query to display the first 5 documents from the results obtained in a collection.**
   [use of limit and find]

a. Select and ignore fields

Develop a MongoDB query to select certain fields and ignore some fields of the documents from any collection.

To select certain fields and ignore others in MongoDB, you use projections in your queries. Projections allow you to specify which fields to include or exclude in the returned documents.

Create database and create the Collection:

```
test> use MoviesDB
switched to db MoviesDB
MoviesDB> db.createCollection("Movies")
{ ok: 1 }
MoviesDB> db.Movies.insertMany([
  { title: "Inception", director: "Christopher Nolan",
genre: "Science Fiction", year: 2010, ratings:
 { imdb: 8.8, rottenTomatoes: 87 } },
  { title: "The Matrix", director: "Wachowskis",
genre: "Science Fiction", year: 1999, ratings:
{ imdb: 8.7, rottenTomatoes: 87 } },
  { title: "The Godfather", director: "Francis Ford Coppola",
genre: "Crime", year: 1972, ratings: { imdb: 9.2, rottenTomatoes: 97 } }]);
```

Output:

```
{
  acknowledged: true,
  insertedIds: {
    '0': ObjectId('66523751d5449c3abf2202d8'),
    '1': ObjectId('66523751d5449c3abf2202d9'),
    '2': ObjectId('66523751d5449c3abf2202da')
  }
}
```

Basic Syntax for Projection

When using the `find()` method, the first parameter is the query filter, and the second parameter is the projection object. The projection object specifies the fields to include (using `1`) or exclude (using `0`).

Including Specific Fields

To include specific fields, set the fields you want to include to `1`:

To select only the `title` and `director` fields from the `Movies` collection:

```
MoviesDB> db.Movies.find({}, { title: 1, director: 1 })
```

Output:

```
[
  {
    _id: ObjectId('66523751d5449c3abf2202d8'),
    title: 'Inception',
    director: 'Christopher Nolan'
  },
  {
    _id: ObjectId('66523751d5449c3abf2202d9'),
    title: 'The Matrix',
    director: 'Wachowskis'
  },
  {
    _id: ObjectId('66523751d5449c3abf2202da'),
    title: 'The Godfather',
    director: 'Francis Ford Coppola'
  }
]
```

```
MoviesDB> db.Movies.find({}, { title: 1, director: 1, _id: 0 })
```

Output:

```
[
  { title: 'Inception', director: 'Christopher Nolan' },
  { title: 'The Matrix', director: 'Wachowskis' },
  { title: 'The Godfather', director: 'Francis Ford Coppola' }
]
```

In this query:

- The filter `{}` means we want to select all documents.
- The projection `{ title: 1, director: 1, _id: 0 }` means we include the `title` and `director` fields, and exclude the `_id` field (which is included by default unless explicitly excluded).

Excluding Specific Fields

To exclude specific fields, set the fields you want to exclude to `0`:

To exclude the `ratings` field from the results:

```
MoviesDB> db.Movies.find({}, { ratings: 0 })
```

Output:

```
[
  {
    _id: ObjectId('66523751d5449c3abf2202d8'),
    title: 'Inception',
    director: 'Christopher Nolan',
    genre: 'Science Fiction',
    year: 2010
  },
  {
    _id: ObjectId('66523751d5449c3abf2202d9'),
    title: 'The Matrix',
    director: 'Wachowskis',
    genre: 'Science Fiction',
    year: 1999
  },
  {
    _id: ObjectId('66523751d5449c3abf2202da'),
    title: 'The Godfather',
    director: 'Francis Ford Coppola',
    genre: 'Crime',
    year: 1972
  }
]
```

In this query:

- The filter `{}` means we want to select all documents.
- The projection `{ ratings: 0 }` means we exclude the `ratings` field.

Combining Filter and Projection

- You can also combine a query filter with a projection. For example, to find movies directed by "Christopher Nolan" and include only the `title` and `year` fields:

```
MoviesDB> db.Movies.find({ director: "Christopher Nolan" }, { title: 1, year: 1, _id: 0 })
```

`Output:`

Output:

```
[ { title: 'Inception', year: 2010 } ]
```

In this query:

- The filter `{ director: "Christopher Nolan" }` selects documents where the `director` is "Christopher Nolan".
- The projection `{ title: 1, year: 1, _id: 0 }` includes only the `title` and `year` fields and excludes the `_id` field.

In MongoDB, projections are used to control which fields are included or excluded in the returned documents. This is useful for optimizing queries and reducing the amount of data transferred over the network. You specify projections as the second parameter in the `find()` method.

b. Use of limit and find in MongoDB query

Develop a MongoDB query to display the first 5 documents from the results obtained in a. (illustrate use of limit and find)

To display the first 5 documents from a query result in MongoDB, you can use the `limit()` method in conjunction with the `find()` method. The `limit()` method restricts the number of documents returned by the query to the specified number.

Example Scenario

Assume we have the `Movies` collection as described previously:

```
test> use MoviesDB
switched to db MoviesDB
MoviesDB> db.createCollection("Movies")
{ ok: 1 }
MoviesDB>db.Movies.insertMany([
  { title: "Inception", director: "Christopher Nolan", genre: "Science Fiction", year: 2010, ratings:
{ imdb: 8.8, rottenTomatoes: 87 } },
  { title: "The Matrix", director: "Wachowskis", genre: "Science Fiction", year: 1999, ratings: {
imdb: 8.7, rottenTomatoes: 87 } },
  { title: "The Godfather", director: "Francis Ford Coppola", genre: "Crime", year: 1972, ratings: {
imdb: 9.2, rottenTomatoes: 97 } },
  { title: "Pulp Fiction", director: "Quentin Tarantino", genre: "Crime", year: 1994, ratings: { imdb:
8.9, rottenTomatoes: 92 } },
  { title: "The Shawshank Redemption", director: "Frank Darabont", genre: "Drama", year: 1994,
ratings: { imdb: 9.3, rottenTomatoes: 91 } },
  { title: "The Dark Knight", director: "Christopher Nolan", genre: "Action", year: 2008, ratings: {
imdb: 9.0, rottenTomatoes: 94 } },
  { title: "Fight Club", director: "David Fincher", genre: "Drama", year: 1999, ratings: { imdb: 8.8,
rottenTomatoes: 79 } }]);
```

Query with Projection and Limit

Suppose you want to display the first 5 documents from the `Movies` collection, including only the `title`, `director`, and `year` fields. Here's how you can do it:

```
MoviesDB> db.Movies.find({}, { title: 1, director: 1, year: 1, _id: 0 }).limit(5)
```

Output:

```
[
  { "title": "Inception", "director": "Christopher Nolan", "year": 2010 },
  { "title": "The Matrix", "director": "Wachowskis", "year": 1999 },
  { "title": "The Godfather", "director": "Francis Ford Coppola", "year": 1972 },
  { "title": "Pulp Fiction", "director": "Quentin Tarantino", "year": 1994 },
  { "title": "The Shawshank Redemption", "director": "Frank Darabont", "year": 1994 }
]
```

Explanation:

- `find({})`: This filter `{}` selects all documents in the collection.
- `{ title: 1, director: 1, year: 1, _id: 0 }`: This projection includes the `title`, `director`, and `year` fields, and excludes the `_id` field.
- `.limit(5)`: This method limits the query result to the first 5 documents.

By using the `find()` method with a projection and the `limit()` method, you can efficiently query and display a subset of documents from a MongoDB collection. This approach helps manage large datasets by retrieving only a specific number of documents, which is particularly useful for paginating results in applications.

Experiment 3:

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

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

a. Query selectors (comparison selectors, logical selectors)

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

Let's create a new collection called `Employees` and insert some documents into it. Then, we'll demonstrate the use of comparison selectors and logical selectors to query this collection.

Create the `Employees` Collection and Insert Documents

First, we need to create the `Employees` collection and insert some sample documents.

```
test> use companyDB

companyDB> db.Employees.insertMany([
  { name: "Alice", age: 30, department: "HR", salary: 50000, joinDate: new Date("2015-01-15") },
  { name: "Bob", age: 24, department: "Engineering", salary: 70000, joinDate: new Date("2019-03-10") },
  { name: "Charlie", age: 29, department: "Engineering", salary: 75000, joinDate: new Date("2017-06-23") },
  { name: "David", age: 35, department: "Marketing", salary: 60000, joinDate: new Date("2014-11-01") },
  { name: "Eve", age: 28, department: "Finance", salary: 80000, joinDate: new Date("2018-08-19") }
])
```

Output:

```
{
  acknowledged: true,
  insertedIds: {
    '0': ObjectId('665356cff5b334bcf92202d8'),
    '1': ObjectId('665356cff5b334bcf92202d9'),
    '2': ObjectId('665356cff5b334bcf92202da'),
    '3': ObjectId('665356cff5b334bcf92202db'),
    '4': ObjectId('665356cff5b334bcf92202dc')
  }
}
```

Queries Using Comparison Selectors

1. $eq (Equal)

Find employees in the "Engineering" department.

companyDB> db.Employees.find({ department: { $eq: "Engineering" } }).pretty()

Output:

```
[
  {
    _id: ObjectId('665356cff5b334bcf92202d9'),
    name: 'Bob',
    age: 24,
    department: 'Engineering',
    salary: 70000,
    joinDate: ISODate('2019-03-10T00:00:00.000Z')
  },
  {
    _id: ObjectId('665356cff5b334bcf92202da'),
    name: 'Charlie',
    age: 29,
    department: 'Engineering',
    salary: 75000,
    joinDate: ISODate('2017-06-23T00:00:00.000Z')
  }
]
```

*2. **$ne** (Not Equal)*

Find employees who are not in the "HR" department.

companyDB> db.Employees.find({ department: { $ne: "HR" } }).pretty()

Output:

```
[
  {
    _id: ObjectId('665356cff5b334bcf92202d9'),
    name: 'Bob',
    age: 24,
    department: 'Engineering',
    salary: 70000,
    joinDate: ISODate('2019-03-10T00:00:00.000Z')
  },
  {
    _id: ObjectId('665356cff5b334bcf92202da'),
    name: 'Charlie',
    age: 29,
    department: 'Engineering',
    salary: 75000,
    joinDate: ISODate('2017-06-23T00:00:00.000Z')
  },
  {
    _id: ObjectId('665356cff5b334bcf92202db'),
    name: 'David',
    age: 35,
    department: 'Marketing',
    salary: 60000,
    joinDate: ISODate('2014-11-01T00:00:00.000Z')
  },
  {
    _id: ObjectId('665356cff5b334bcf92202dc'),
    name: 'Eve',
    age: 28,
    department: 'Finance',
    salary: 80000,
    joinDate: ISODate('2018-08-19T00:00:00.000Z')
  }
]
```

3. `$gt` (Greater Than)

Find employees who are older than 30.

```
companyDB> db.Employees.find({ age: { $gt: 30 } }).pretty()
Output:
[
  {
    _id: ObjectId('665356cff5b334bcf92202db'),
    name: 'David',
    age: 35,
    department: 'Marketing',
    salary: 60000,
    joinDate: ISODate('2014-11-01T00:00:00.000Z')
  }
]
```

4. `$lt` (Less Than)
Find employees with a salary less than 70000.

companyDB> db.Employees.find({ salary: { $lt: 70000 } }).pretty()

Output:

```
[
  {
    _id: ObjectId('665356cff5b334bcf92202d8'),
    name: 'Alice',
    age: 30,
    department: 'HR',
    salary: 50000,
    joinDate: ISODate('2015-01-15T00:00:00.000Z')
  },
  {
    _id: ObjectId('665356cff5b334bcf92202db'),
    name: 'David',
    age: 35,
    department: 'Marketing',
    salary: 60000,
    joinDate: ISODate('2014-11-01T00:00:00.000Z')
  }
]
```

5. `$gte` (Greater Than or Equal)
Find employees who joined on or after January 1, 2018.

companyDB> db.Employees.find({ joinDate: { $gte: new Date("2018-01-01") } }).pretty()

Output:

```
[
  {
    _id: ObjectId('665356cff5b334bcf92202d9'),
    name: 'Bob',
    age: 24,
    department: 'Engineering',
    salary: 70000,
    joinDate: ISODate('2019-03-10T00:00:00.000Z')
  },
  {
    _id: ObjectId('665356cff5b334bcf92202dc'),
    name: 'Eve',
    age: 28,
    department: 'Finance',
    salary: 80000,
    joinDate: ISODate('2018-08-19T00:00:00.000Z')
  }
]
```

6. `$lte` (Less Than or Equal)

Find employees who are 28 years old or younger.

companyDB> db.Employees.find({ age: { $lte: 28 } }).pretty()

Output:

```
[
  {
    _id: ObjectId('665356cff5b334bcf92202d9'),
    name: 'Bob',
    age: 24,
    department: 'Engineering',
    salary: 70000,
    joinDate: ISODate('2019-03-10T00:00:00.000Z')
  },
  {
    _id: ObjectId('665356cff5b334bcf92202dc'),
    name: 'Eve',
    age: 28,
    department: 'Finance',
    salary: 80000,
    joinDate: ISODate('2018-08-19T00:00:00.000Z')
  }
]
```

Queries Using Logical Selectors

1. `$and` (Logical AND)

Find employees who are in the “Engineering” department and have a salary greater than 70000.

```
companyDB> db.Employees.find({
  $and: [
    { department: "Engineering" },
    { salary: { $gt: 70000 } }
  ]
}).pretty()
Output:

[
  {
    _id: ObjectId('665356cff5b334bcf92202da'),
    name: 'Charlie',
    age: 29,
    department: 'Engineering',
    salary: 75000,
    joinDate: ISODate('2017-06-23T00:00:00.000Z')
  }
]
```

*2. **$or** (Logical OR)*

Find employees who are either in the “HR” department or have a salary less than 60000.

```
companyDB> db.Employees.find({
  $or: [
    { department: "HR" },
    { salary: { $lt: 60000 } }
  ]
}).pretty()

Output:
[
  {
    _id: ObjectId('665356cff5b334bcf92202d8'),
    name: 'Alice',
    age: 30,
    department: 'HR',
    salary: 50000,
    joinDate: ISODate('2015-01-15T00:00:00.000Z')
  }
]
```

3. `$not` (Logical NOT)

Find employees who are not in the “Engineering” department.

```
companyDB> db.Employees.find({
  department: {
    $not: { $eq: "Engineering" }
  }
}).pretty()
```

Output:

```
[
  {
    _id: ObjectId('665356cff5b334bcf92202d8'),
    name: 'Alice',
    age: 30,
    department: 'HR',
    salary: 50000,
    joinDate: ISODate('2015-01-15T00:00:00.000Z')
  },
  {
    _id: ObjectId('665356cff5b334bcf92202db'),
    name: 'David',
    age: 35,
    department: 'Marketing',
    salary: 60000,
    joinDate: ISODate('2014-11-01T00:00:00.000Z')
  },
  {
    _id: ObjectId('665356cff5b334bcf92202dc'),
    name: 'Eve',
    age: 28,
    department: 'Finance',
    salary: 80000,
    joinDate: ISODate('2018-08-19T00:00:00.000Z')
  }
]
```

4. `$nor` (Logical NOR)

Find employees who are neither in the “HR” department nor have a salary greater than 75000.

```
companyDB> db.Employees.find({
  $nor: [
    { department: "HR" },
    { salary: { $gt: 75000 } }
  ]
}).pretty()
```
Output:
```
[
  {
    _id: ObjectId('665356cff5b334bcf92202d9'),
    name: 'Bob',
    age: 24,
    department: 'Engineering',
    salary: 70000,
    joinDate: ISODate('2019-03-10T00:00:00.000Z')
  },
  {
    _id: ObjectId('665356cff5b334bcf92202da'),
    name: 'Charlie',
    age: 29,
    department: 'Engineering',
    salary: 75000,
    joinDate: ISODate('2017-06-23T00:00:00.000Z')
  },
  {
    _id: ObjectId('665356cff5b334bcf92202db'),
    name: 'David',
    age: 35,
    department: 'Marketing',
    salary: 60000,
    joinDate: ISODate('2014-11-01T00:00:00.000Z')
  }
]
```

b. Query selectors (Geospatial selectors, Bitwise selectors)

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

Let’s extend our MongoDB examples to include queries using geospatial selectors and bitwise selectors. We will create a new collection called `Places` for geospatial queries and a collection called Devices for bitwise queries.

```
test>use geoDatabase

geoDatabase>db.Places.insertMany([

  { name: "Central Park", location: { type: "Point", coordinates: [-73.9654, 40.7829] } },

  { name: "Times Square", location: { type: "Point", coordinates: [-73.9851, 40.7580] } },

  { name: "Brooklyn Bridge", location: { type: "Point", coordinates: [-73.9969, 40.7061] }
},

  { name: "Empire State Building", location: { type: "Point", coordinates: [-73.9857,
40.7488] } },

  { name: "Statue of Liberty", location: { type: "Point", coordinates: [-74.0445, 40.6892]
} }

])
```

1. `$near` (Find places near a certain point)

Find places near a specific coordinate, for example, near Times Square.

```
geoDatabase>db.Places.find({
  location: {
    $near: {
      $geometry: {
        type: "Point",
        coordinates: [-73.9851, 40.7580]
      },
      $maxDistance: 5000 // distance in meters
    }
  }
}).pretty()

Output:
[
  {
    _id: ObjectId('66536a9799cad9cd2b2202d9'),
    name: 'Times Square',
    location: { type: 'Point', coordinates: [ -73.9851, 40.758 ] }
  },
  {
    _id: ObjectId('66536a9799cad9cd2b2202db'),
    name: 'Empire State Building',
    location: { type: 'Point', coordinates: [ -73.9857, 40.7488 ] }
  },
  {
    _id: ObjectId('66536a9799cad9cd2b2202d8'),
    name: 'Central Park',
    location: { type: 'Point', coordinates: [ -73.9654, 40.7829 ] }
  }
]
```

`2. '$geoWithin'` (Find places within a specific area)

Find places within a specific polygon, for example, an area covering part of Manhattan.

```
geoDatabase>db.Places.find({
  location: {
    $geoWithin: {
      $geometry: {
        type: "Polygon",
        coordinates: [
          [
            [-70.016, 35.715],
            [-74.014, 40.717],
            [-73.990, 40.730],
            [-73.990, 40.715],
            [-70.016, 35.715]
          ]
        ]
      }
    }
  }
}).pretty()
```

Output:

```
[
  {
    _id: ObjectId('66536a9799cad9cd2b2202da'),
    name: 'Brooklyn Bridge',
    location: { type: 'Point', coordinates: [ -73.9969, 40.7061 ] }
  }
]
```

Bitwise Selectors

Next, let’s create a `Devices` collection for bitwise operations.

Create the `Devices` Collection and Insert Documents

```
test>use techDB

techDB>db.Devices.insertMany([
  { name: "Device A", status: 5 }, // Binary: 0101
  { name: "Device B", status: 3 }, // Binary: 0011
  { name: "Device C", status: 12 }, // Binary: 1100
  { name: "Device D", status: 10 }, // Binary: 1010
  { name: "Device E", status: 7 }  // Binary: 0111
])
```

1. ‘$bitsAllSet’ (Find documents where all bits are set)
Find devices where the binary status has both the 1st and 3rd bits set (binary mask 0101, or decimal 5).

```
techDB>db.Devices.find({
  status: { $bitsAllSet: [0, 2] }
}).pretty()
```

Output:

```
[
  {
    _id: ObjectId('6653703d4e38f292e52202d8'),
    name: 'Device A',
    status: 5
  },
  {
    _id: ObjectId('6653703d4e38f292e52202dc'),
    name: 'Device E',
    status: 7
  }
]
```

2. $bitsAnySet (Find documents where any of the bits are set)
Find devices where the binary status has at least the 2nd bit set (binary mask 0010, or decimal 2).

```
techDB>db.Devices.find({
  status: { $bitsAnySet: [1] }
}).pretty()
```

Output:

```
[
  {
    _id: ObjectId('6653703d4e38f292e52202d9'),
    name: 'Device B',
    status: 3
  },
  {
    _id: ObjectId('6653703d4e38f292e52202db'),
    name: 'Device D',
    status: 10
  },
  {
    _id: ObjectId('6653703d4e38f292e52202dc'),
    name: 'Device E',
    status: 7
  }
]
```

3. '$bitsAllClear' (Find documents where all bits are clear)
Find devices where the binary status has both the 2nd and 4th bits clear (binary mask 1010, or decimal 10).

```
techDB>b.Devices.find({
  status: { $bitsAllClear: [1, 3] }
}).pretty()
```

Output:

```
[
  {
    _id: ObjectId('6653703d4e38f292e52202d8'),
    name: 'Device A',
    status: 5
  }
]
```

4. $bitsAnyClear (Find documents where any of the bits are clear)
Find devices where the binary status has at least the 1st bit clear (binary mask 0001, or decimal 1).

```
techDB>db.Devices.find({
  status: { $bitsAnyClear: [0] }
}).pretty()
```

Output:

```
[
  {
    _id: ObjectId('6653703d4e38f292e52202da'),
    name: 'Device C',
    status: 12
  },
  {
    _id: ObjectId('6653703d4e38f292e52202db'),
    name: 'Device D',
    status: 10
  }
]
```