1. Illustration of Where Clause, AND,OR operations in MongoDB.MongoDB : Insert, Query, Update, Delete and Projection.

Switch to created DB:

```
use ProgBooksDB
switched to db ProgBooksDB
```

Create Collection:

```
ProgBooksDB> db.createCollection("ProgrammingBooks")
```

Insert One Documents:

```
ProgBooksDB>  db.ProgrammingBooks.insertOne({
  title: "The Pragmatic Programmer: Your Journey to Mastery",
  author: "David Thomas, Andrew Hunt",
  category: "Software Development",
  year: 1999
})
```

Insert Many Documents:

```
ProgBooksDB> db.ProgrammingBooks.insertMany([
  {
    title: "Clean Code: A Handbook of Agile Software Craftsmanship",
    author: "Robert C. Martin",
    category: "Software Development",
    year: 2008
  },
  {
    title: "JavaScript: The Good Parts",
    author: "Douglas Crockford",
    category: "JavaScript",
    year: 2008
  },
  {
    title: "Design Patterns: Elements of Reusable Object-Oriented Software",
    author: "Erich Gamma, Richard Helm, Ralph Johnson, John Vlissides",
    category: "Software Design",
    year: 1994
  },
  {
    title: "Introduction to Algorithms",
    author: "Thomas H. Cormen, Charles E. Leiserson, Ronald L. Rivest, Clifford Stein",
    category: "Algorithms",
    year: 1990
  },
  {
    title: "Python Crash Course: A Hands-On, Project-Based Introduction to Programming",
    author: "Eric Matthes",
    category: "Python",
    year: 2015
  }
])
```

Find Documents:

```
ProgBooksDB> db.ProgrammingBooks.find().pretty()
[
  {
    _id: ObjectId('664ee3b1924a8039f62202d8'),
    title: 'The Pragmatic Programmer: Your Journey to Mastery',
    author: 'David Thomas, Andrew Hunt',
    category: 'Software Development',
    year: 1999
  },
  {
    _id: ObjectId('664ee452924a8039f62202d9'),
    title: 'Clean Code: A Handbook of Agile Software Craftsmanship',
    author: 'Robert C. Martin',
    category: 'Software Development',
    year: 2008
  },
  {
    _id: ObjectId('664ee452924a8039f62202da'),
    title: 'JavaScript: The Good Parts',
    author: 'Douglas Crockford',
    category: 'JavaScript',
    year: 2008
  },
  {
    _id: ObjectId('664ee452924a8039f62202db'),
    title: 'Design Patterns: Elements of Reusable Object-Oriented Software',
    author: 'Erich Gamma, Richard Helm, Ralph Johnson, John Vlissides',
    category: 'Software Design',
    year: 1994
  },
  {
    _id: ObjectId('664ee452924a8039f62202dc'),
    title: 'Introduction to Algorithms',
    author: 'Thomas H. Cormen, Charles E. Leiserson, Ronald L. Rivest, Clifford Stein',
    category: 'Algorithms',
    year: 1990
  },
  {
    _id: ObjectId('664ee452924a8039f62202dd'),
    title: 'Python Crash Course: A Hands-On, Project-Based Introduction to
Programming',
    author: 'Eric Matthes',
    category: 'Python',
    year: 2015
  }
]
```

Where Clause(Find The Documents with condition):

```
ProgBooksDB> db.ProgrammingBooks.find({ year: { $gt: 2000 } }).pretty()
[
  {
    _id: ObjectId('664ee452924a8039f62202d9'),
    title: 'Clean Code: A Handbook of Agile Software Craftsmanship',
    author: 'Robert C. Martin',
    category: 'Software Development',
    year: 2008
  },
  {
    _id: ObjectId('664ee452924a8039f62202da'),
    title: 'JavaScript: The Good Parts',
    author: 'Douglas Crockford',
    category: 'JavaScript',
    year: 2008
  },
  {
    _id: ObjectId('664ee452924a8039f62202dd'),
    title: 'Python Crash Course: A Hands-On, Project-Based Introduction to
Programming',
    author: 'Eric Matthes',
    category: 'Python',
    year: 2015
  }
]
```

Update One Documents:

```
ProgBooksDB>db.ProgrammingBooks.updateOne(
  { title: "Clean Code: A Handbook of Agile Software Craftsmanship" },
  { $set: { author: "Robert C. Martin (Uncle Bob)" } }
)

//verify by displaying books published in year 2008
ProgBooksDB> db.ProgrammingBooks.find({ year: { $eq: 2008 } }).pretty()
[
  {
    _id: ObjectId('663eaaebae582498972202df'),
    title: 'Clean Code: A Handbook of Agile Software Craftsmanship',
    author: 'Robert C. Martin (Uncle Bob)',
    category: 'Software Development',
    year: 2008
  },
  {
    _id: ObjectId('663eaaebae582498972202e0'),
    title: 'JavaScript: The Good Parts',
    author: 'Douglas Crockford',
    category: 'JavaScript',
    year: 2008
  }
]

//another way to verify
ProgBooksDB> db.ProgrammingBooks.find({ author: { $regex: "Robert*" }
}).pretty()
[
  {
    _id: ObjectId('664ee452924a8039f62202d9'),
    title: 'Clean Code: A Handbook of Agile Software Craftsmanship',
    author: 'Robert C. Martin (Uncle Bob)',
    category: 'Software Development',
    year: 2008
  }
]
```

Update Many Documents:

```
ProgBooksDB> db.ProgrammingBooks.updateMany(
  { year: { $lt: 2010 } },
  { $set: { category: "Classic Programming Books" } }
)

//verify the update operation by displaying books published before year 2010
ProgBooksDB> db.ProgrammingBooks.find({ year: { $lt: 2010 } }).pretty()
[
  {
    _id: ObjectId('663eaaebae582498972202df'),
    title: 'Clean Code: A Handbook of Agile Software Craftsmanship',
    author: 'Robert C. Martin (Uncle Bob)',
    category: 'Classic Programming Books',
    year: 2008
  },
  {
    _id: ObjectId('663eaaebae582498972202e0'),
    title: 'JavaScript: The Good Parts',
    author: 'Douglas Crockford',
    category: 'Classic Programming Books',
    year: 2008
  },
  {
    _id: ObjectId('663eaaebae582498972202e1'),
    title: 'Design Patterns: Elements of Reusable Object-Oriented Software',
    author: 'Erich Gamma, Richard Helm, Ralph Johnson, John Vlissides',
    category: 'Classic Programming Books',
    year: 1994
  },
  {
    _id: ObjectId('663eaaebae582498972202e2'),
    title: 'Introduction to Algorithms',
    author: 'Thomas H. Cormen, Charles E. Leiserson, Ronald L. Rivest,
Clifford Stein',
    category: 'Classic Programming Books',
    year: 1990
  },
  {
    _id: ObjectId('663eab05ae582498972202e4'),
    title: 'The Pragmatic Programmer: Your Journey to Mastery',
    author: 'David Thomas, Andrew Hunt',
    category: 'Classic Programming Books',
    year: 1999
  }
]
```

Delete operation Documents:

```
ProgBooksDB> db.ProgrammingBooks.deleteOne({ title: "JavaScript: The Good
Parts" })

{ acknowledged: true, deletedCount: 1 }

//Verify to see document is deleted
ProgBooksDB> db.ProgrammingBooks.find({ title: "JavaScript: The Good Parts"
}).pretty()
```

Experiment 2:

a. Select & ignore Fields

Create Database & 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 } },
  { 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 } }
])
```

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 })

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

Excluding Specific Field:

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

Combining Filter & Projection:

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

b. Use of Limit and find in MongoDB query

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

Experiment 3: Query selectors (comparison selectors, logical selectors)

```
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") }
])
```

a. Queries Using Comparison Selectors

1. $eq (Equal)

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

2. $ne (Not Equal)

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

3. $gt (Greater Than)

```
companyDB> db.Employees.find({ age: { $gt: 30 } }).pretty()
```

4. $lt (Less Than)

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

5. $gte (Greater Than or Equal)

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

6. $lte (Less Than or Equal)

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

Queries Using Logical Selectors:

1. $and (Logical AND)

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

2. $or (Logical OR)

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

3. $not (Logical NOT)

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

4. $nor (Logical NOR)

```
companyDB> db.Employees.find({   $nor: [ { department: "HR" }, { salary: { $gt: 75000 } }  ]
).pretty()
```

b. Query selectors (Geospatial selectors, Bitwise selectors)

Geospatial Selectors:

```
test> use geoDatabase
switched to db 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] } }
])

// Create a geospatial index
geoDatabase> db.Places.createIndex({ location: "2dsphere" })
```

Geospatial Queries:

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

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

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

```
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()
```

Bitwise Selectors:

```
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
                                       ])
```

Bitwise Queries:

1. **$bitsAllSet (Find documents where all bits are set)**

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

2. **$bitsAnySet (Find documents where any of the bits are set)**

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

3. **$bitsAllClear (Find documents where all bits are clear)**

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

4. **$bitsAnyClear (Find documents where any of the bits are clear)**

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

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

Creating Product:

```
test> use retailDB
switched to db retailDB
retailDB> db.Products.insertMany([
  {
    name: "Laptop",
    brand: "BrandA",
    features: [
      { name: "Processor", value: "Intel i7" },
      { name: "RAM", value: "16GB" },
      { name: "Storage", value: "512GB SSD" }
    ],
    reviews: [
      { user: "Alice", rating: 5, comment: "Excellent!" },
      { user: "Bob", rating: 4, comment: "Very good" },
      { user: "Charlie", rating: 3, comment: "Average" }
    ]
  },
  {
    name: "Smartphone",
    brand: "BrandB",
    features: [
      { name: "Processor", value: "Snapdragon 888" },
      { name: "RAM", value: "8GB" },
      { name: "Storage", value: "256GB" }
    ],
    reviews: [
      { user: "Dave", rating: 4, comment: "Good phone" },
      { user: "Eve", rating: 2, comment: "Not satisfied" }
    ]
  }
])
```

1. The `$` Projection Operator

Example: Find the product named “Laptop” and project the review from the user “Alice”.

```
retailDB> db.Products.find(
  { name: "Laptop", "reviews.user": "Alice" },
  { "reviews.$": 1 }
).pretty()
```

2. The `$elemMatch` Projection Operator

Example: Find the product named “Laptop” and project the review where the rating is greater than 4.

```
retailDB> db.Products.find(
  { name: "Laptop" },
  { reviews: { $elemMatch: { rating: { $gt: 4 } } } }
).pretty()
```

3. The `$slice` Projection Operator

Example: Find the product named “Smartphone” and project the first review.

```
retailDB> db.Products.find(
  { name: "Smartphone" },
  { reviews: { $slice: 1 } }
).pretty()
```

5. Execute Aggregation operations ($avg, $min,$max, $push, $addToSet etc.). students encourage to execute several queries to demonstrate various aggregation operators)

Create the `Sales` Collection and Insert Documents

```
test> use salesDB
salesDB> db.Sales.insertMany([
  { date: new Date("2024-01-01"), product: "Laptop", price: 1200, quantity: 1, customer: "Amar" },
  { date: new Date("2024-01-02"), product: "Laptop", price: 1200, quantity: 2, customer: "Babu" },
  { date: new Date("2024-01-03"), product: "Mouse", price: 25, quantity: 5, customer: "Chandra" },
  { date: new Date("2024-01-04"), product: "Keyboard", price: 45, quantity: 3, customer: "Amar" },
  { date: new Date("2024-01-05"), product: "Monitor", price: 300, quantity: 1, customer: "Babu" },
  { date: new Date("2024-01-06"), product: "Laptop", price: 1200, quantity: 1, customer: "Deva" }
])
```

1. $avg (Average)

```
salesDB> db.Sales.aggregate([
  {
    $group: {
      _id: "$product",
      averagePrice: { $avg: "$price" }
    }
  } ]).pretty()
```

2. `$min` (Minimum)

```
salesDB> db.Sales.aggregate([
  {
    $group: {
      _id: "$product",
      minPrice: { $min: "$price" }
    }
  } ]).pretty()
```

3. `$max` (Maximum)

```
salesDB> db.Sales.aggregate([
  {
    $group: {
      _id: "$product",
      maxPrice: { $max: "$price" }
    }
  } ]).pretty()
```

4. `$push` (Push Values to an Array)

```
 salesDB> db.Sales.aggregate([
  {
    $group: {
      _id: "$customer",
      products: { $push: "$product" }
    }
  } ]).pretty()
```

5. `$addToSet` (Add Unique Values to an Array)

```
salesDB> db.Sales.aggregate([
  {
    $group: {
      _id: "$customer",
      uniqueProducts: { $addToSet: "$product" }
    }
  } ]).pretty()
```