Experiment 1:

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

b. Execute the Commands of MongoDB and operations in MongoDB : Insert, Query, Update, Delete and Projection. (Note: use any collection)

a. Where Clause, AND,OR operations in MongoDB

In MongoDB, the equivalent of SQL's `WHERE` clause is achieved using query filters within the `find()` method. You can also combine multiple conditions using logical operators like `$and` and `$or`. Here's how you can illustrate the usage of these features:

Setting Up Example Data

First, let's assume we have a collection named **`ProgrammingBooks`** with the following documents:

```
ProgBooksDB> use newDB
switched to db newDB
newDB>db.getName()
newDB
newDB>db.createCollection("ProgrammingBooks")
{ ok: 1 }
newDB>db.ProgrammingBooks.insertMany([
  { title: "Clean Code",
    author: "Robert C. Martin",
    category: "Software Development", year: 2008 },
  { title: "JavaScript: The Good Parts",
    author: "Douglas Crockford",
   category: "JavaScript", year: 2008 },
  { title: "Design Patterns",
    author: "Erich Gamma",
    category: "Software Design", year: 1994 },
  { title: "Introduction to Algorithms",
    author: "Thomas H. Cormen",
   category: "Algorithms", year: 2009 },
  { title: "Python Crash Course",
    author: "Eric Matthes",
    category: "Python", year: 2015 }]);
```

Using the `WHERE` Clause Equivalent

To query documents with specific conditions, you can use the **`find()`** method with a filter object. For example, to find books published in the year 2008:

pretty() method is used to configure the cursor to display results in an easy-to-read format.

newDB> db.ProgrammingBooks.find({ year: 2008 }).pretty()
Output:

```
[
  {
    _id: ObjectId('6651daad9edbdf91e12202e2'),
    title: 'Clean Code',
    author: 'Robert C. Martin',
    category: 'Software Development',
    year: 2008
  },
  {
    _id: ObjectId('6651daad9edbdf91e12202e3'),
    title: 'JavaScript: The Good Parts',
    author: 'Douglas Crockford',
    category: 'JavaScript',
    year: 2008
  }
]
```

Using the `$and` Operator

The **`$and`** operator is used to combine multiple conditions that must all be true. Here's how to find books that are in the "Software Development" category and published in the year 2008:

newDB>db.ProgrammingBooks.find({ $and: [
{ category: "Software Development" },
{ year: 2008 }]}).pretty()
Output:

```
[
  {
    _id: ObjectId('6651daad9edbdf91e12202e2'),
    title: 'Clean Code',
    author: 'Robert C. Martin',
    category: 'Software Development',
    year: 2008
  }
]
```

- In this query both conditions must be met for a document to be included in the result.

Using the `$or` Operator

The **$or** operator is used to combine multiple conditions where at least one must be true. Here's how to find books that are either in the "JavaScript" category or published in the year 2015:

```
newDB> db.ProgrammingBooks.find({
  $or: [
    { category: "JavaScript" },
    { year: 2015 }
  ]
}).pretty()
```

Output:

```
[
  {
    _id: ObjectId('6651daad9edbdf91e12202e3'),
    title: 'JavaScript: The Good Parts',
    author: 'Douglas Crockford',
    category: 'JavaScript',
    year: 2008
  },
  {
    _id: ObjectId('6651daad9edbdf91e12202e6'),
    title: 'Python Crash Course',
    author: 'Eric Matthes',
    category: 'Python',
    year: 2015
  }
]
```

In this query a document will be included in the result if it meets either condition.

Combining $and and $or Operators

You can combine **$and** and **$or** operators for more complex queries. For example, to find books that are either in the "Software Development" category and published after 2007, or in the "Python" category:

```
newDB> db.ProgrammingBooks.find({
  $or: [
    {
      $and: [
        { category: "Software Development" },
        { year: { $gt: 2007 } }
      ]
    },
    { category: "Python" }
  ]
}).pretty()
```

Output:

```
[
 {
  _id: ObjectId('6651daad9edbdf91e12202e2'),
  title: 'Clean Code',
  author: 'Robert C. Martin',
  category: 'Software Development',
  year: 2008
 },
 {
  _id: ObjectId('6651daad9edbdf91e12202e6'),
  title: 'Python Crash Course',
  author: 'Eric Matthes',
  category: 'Python',
  year: 2015
 }
]
```

a. Execute the Commands of MongoDB and operations in MongoDB : Insert, Query, Update, Delete and Projection. (Note: use any collection)

MongoDB Operations

Switch to a Database

If you want to use a specific database, switch to that database using the **`use`** command. If the database doesn't exist, MongoDB will create it implicitly when you insert data into it:

test> use ProgBooksDB
switched to db ProgBooksDB
ProgBooksDB>

Create the `ProgrammingBooks` Collection:

To create the **`ProgrammingBooks`** collection, use the **`createCollection()`** method. This step is optional because MongoDB will automatically create the collection when you insert data into it, but you can explicitly create it if needed:

ProgBooksDB> db.createCollection("ProgrammingBooks")

Insert operations

Insert a Single Document into `ProgrammingBooks`:

Use the **`insertOne()`** method to insert a new document into the **`ProgrammingBooks`** collection:

ProgBooksDB> db.ProgrammingBooks.insertOne({

```
  title: "The Pragmatic Programmer: Your Journey to Mastery",
  author: "David Thomas, Andrew Hunt",
  category: "Software Development",
  year: 1999
})
```

Insert multiple Documents into the `ProgrammingBooks` Collection :

Now, insert 5 documents representing programming books into the **`ProgrammingBooks`** collection using the **`insertMany()`** method:

```
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
  }
])
```

Query operations

Find All Documents

To retrieve all documents from the **`ProgrammingBooks`** collection:

```
ProgBooksDB> db.ProgrammingBooks.find().pretty()
```

Find Documents Matching a Condition

To find books published after the year 2000:

```
ProgBooksDB> db.ProgrammingBooks.find({ year: { $gt: 2000 } }).pretty()
```

Update Operations

a. Update a Single Document

To update a specific book (e.g., change the author of a book):

```
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()
```

b. Update Multiple Documents

To update multiple books (e.g., update the category of books published before 2010):

```
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()
```

Delete Operations

Delete a Single Document

To delete a specific book from the collection (e.g., delete a book by title):

```
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()
```

Delete Multiple Documents

To delete multiple books based on a condition (e.g., delete all books published before 1995):

```
ProgBooksDB> db.ProgrammingBooks.deleteMany({ year: { $lt: 1995 } })
{ acknowledged: true, deletedCount: 2 }
```

Delete All Documents in the Collection:

To delete all documents in a collection (e.g., `ProgrammingBooks`), use the `deleteMany()` method with an empty filter '`{}`':

```
ProgBooksDB> db.ProgrammingBooks.deleteMany({})
{ acknowledged: true, deletedCount: 3 }

//verify by displaying the collection
ProgBooksDB> db.ProgrammingBooks.find().pretty()
```

Projection Operations

In MongoDB, a projection refers to the mechanism of specifying which fields (or columns) should be returned from a query result. When querying a collection, you can use projection to control the shape of the returned documents by specifying which fields to include or exclude.In MongoDB, projection is typically specified as the second parameter to the `find()` method. The projection parameter takes an object where keys represent the fields to include or exclude, with values of `1` (include) or `0` (exclude).

Include Specific Fields:

Use `1` to include a field in the result:

```
ProgBooksDB> db.ProgrammingBooks.find({}, { title: 1, author: 1 } )
```

Exclude Specific Fields: Use `0` to exclude a field from the result:

```
ProgBooksDB> db.ProgrammingBooks.find({}, {year: 0}
```