First practical

```
// Step 1: Create and switch to the "school" database
use school;

// Step 2: Insert documents (batch insert)
db.students.insertMany([
  { "_id": 1, "name": "John", "age": 21, "status": "active" },
  { "_id": 2, "name": "Alice", "age": 22, "status": "inactive" },
  { "_id": 3, "name": "Bob", "age": 20, "status": "active" },
  { "_id": 4, "name": "David", "age": 23, "status": "inactive" }
]);

// Step 3: Find a specific document using find()
print("Find John:");
printjson(db.students.find({ "name": "John" }).toArray());

// Step 4: Find one document using findOne()
print("Find one document with name Alice:");
printjson(db.students.findOne({ "name": "Alice" }));

// Step 5: Query with conditionals and operators ($gt, $lt)
print("Find students with age greater than 21:");
printjson(db.students.find({ "age": { $gt: 21 } }).toArray());

// Step 6: Query using $or
print("Find students who are either 'active' or age greater than 22:");
printjson(db.students.find({
  $or: [
    { "status": "active" },
    { "age": { $gt: 22 } }
  ]
}).toArray());

// Step 7: Update a document (updateOne with modifiers)
db.students.updateOne(
  { "name": "Bob" },
  { $set: { "status": "inactive" } }
);

// Step 8: Update multiple documents (updateMany)
db.students.updateMany(
  { "status": "inactive" },
  { $set: { "status": "active" } }
);

// Step 9: Remove a document (deleteOne)
db.students.deleteOne({ "name": "David" });

// Step 10: Remove multiple documents (deleteMany)
db.students.deleteMany({ "status": "active" });

// Step 11: Find with sorting and limiting
print("Find students sorted by age, limit to 2:");
printjson(db.students.find().sort({ "age": 1 }).limit(2).toArray());

// Step 12: Find with skip and limit (pagination)
print("Find students, skip first 1, limit to 2:");
printjson(db.students.find().skip(1).limit(2).toArray());
```

```
// Step 13: Using a $where query (find students where age is greater than 21)
print("Find students where age > 21 using $where:");
printjson(db.students.find({ $where: "this.age > 21" }).toArray());

// Step 14: Aggregate Example (group by status and sum ages)
print("Aggregate students by status and sum ages:");
printjson(db.students.aggregate([
    { $group: { _id: "$status", total_age: { $sum: "$age" } } }
]).toArray());
```

Map Reduce:

```
{
   "_id": 1, "name": "Shubham", "age": 20, "status": 0
},
{
   "_id": 2, "name": "Ram", "age": 25, "status": 1
},
{
   "_id": 3, "name": "Bebo", "age": 22, "status": 0
},
{
   "_id": 4, "name": "Foo", "age": 30, "status": 1
}


var mapFunction = function() {
   emit(this.status, this.age);
};


var reduceFunction = function(key, values) {
   return Array.sum(values);  // Sum up the values for each status
};


db.students.mapReduce(mapFunction, reduceFunction, { out: "total_age_by_status" });
```

Aggregate:

```
db.students.aggregate([
   {
      $group: {
         _id: "$status", // Group by 'status'
         total_age: { $sum: "$age" } // Calculate the sum of 'age' for each 'status'
      }
   }
])

db.students.aggregate([
   {
      $sort: { age: -1 } // Sort by 'age' in descending order
   },
   {
      $limit: 2 // Limit the result to the top 2 documents
   }
])

db.students.aggregate([
   {
      $project: {
         _id: 0, // Exclude '_id' field
         name: 1, // Include 'name' field
         age: 1   // Include 'age' field
      }
   }
])
```

```
db.students.aggregate([
    {
        $match: { age: { $gt: 20 } } // Filter students whose age is greater than 20
    },
    {
        $group: {
            _id: "$status", // Group by 'status'
            total_age: { $sum: "$age" } // Sum the ages for each group
        }
    }
])
```

Aggregation and indexing:

```
[
  { "_id": 1, "customer": "Alice", "total": 100 },
  { "_id": 2, "customer": "Bob", "total": 200 },
  { "_id": 3, "customer": "Alice", "total": 150 },
  { "_id": 4, "customer": "Alice", "total": 50 },
  { "_id": 5, "customer": "Bob", "total": 300 }
]

db.orders.aggregate([
    {
        $group: {
            _id: "$customer",  // Group by customer
            total_spent: { $sum: "$total" }  // Sum the 'total' field for each customer
        }
    }
])

db.orders.createIndex({ customer: 1, total: -1 })  // Ascending on 'customer', descending on 'total'

db.orders.createIndex({ customer: 1 })  // Index on the 'customer' field

db.orders.aggregate([
    {
        $match: { customer: "Alice" }  // This operation can benefit from the index
    },
    {
        $group: {
            _id: "$customer",
            total_spent: { $sum: "$total" }
        }
    }
])
```

Map Reducing and Indexing:

```
// Step 1: Create an `orders` collection and insert sample documents

db.orders.insertMany([
   { "_id": 1, "customer": "Alice", "total": 100 },
   { "_id": 2, "customer": "Bob", "total": 200 },
   { "_id": 3, "customer": "Alice", "total": 150 },
   { "_id": 4, "customer": "Alice", "total": 50 },
   { "_id": 5, "customer": "Bob", "total": 300 }
]);

// Step 2: Create an index on the 'customer' field to improve MapReduce performance
db.orders.createIndex({ customer: 1 });

// Step 3: Define the MapReduce functions

// Map function: Emits customer as the key and total as the value
var mapFunction = function() {
   emit(this.customer, this.total);
};

// Reduce function: Sums the total values for each customer
var reduceFunction = function(key, values) {
   return Array.sum(values);  // Sum the total for each customer
};

// Step 4: Perform the MapReduce operation and store the result in the 'totals_by_customer'
collection
db.orders.mapReduce(
   mapFunction,
   reduceFunction,
   { out: "totals_by_customer" }
);

// Step 5: Create an index on the '_id' field of the 'totals_by_customer' collection
db.totals_by_customer.createIndex({ _id: 1 });

// Step 6: Query the results from the 'totals_by_customer' collection
db.totals_by_customer.find();
```