

# **BIG DATA MANAGEMENT**

## **POST GRADUATE DIPLOMA IN DATA ENGINEERING**

### **ASSIGNMENT - 7**

#### **SUBMITTED BY:**

**NIRAJ BHAGCHANDANI [G23AI2087]**



**SUBMISSION DATE: 15<sup>th</sup> December, 2024**

**DEPARTMENT OF AIDE  
INDIAN INSTITUTE OF TECHNOLOGY, JODHPUR**

Step 1: Name your project and optionally add tags to categorize it.

CODEMATE - 2020-12-16 &gt; PROJECTS

## Create a Project

Name Your Project > Add Members

### Name Your Project

Project names have to be unique within the organization (and other restrictions).

bdm-g23ai2087

### Add Tags (Optional)

Use tags to efficiently label and categorize your projects. A project can have a maximum of 50 tags. You can modify tags for the project later. [Learn more](#)

Key	Value	Actions
<input type="text" value="Select a key or enter your own"/>	: <input type="text" value="Select a value or enter your own"/>	
+ Add tag		
		0 TAGS

Step 2: Select a cluster type based on your requirements, such as free, serverless, or dedicated options.

## Deploy your cluster

Use a template below or set up advanced configuration options. You can also edit these configuration options once the cluster is created.

☐

**M10**

**\$0.08/hour**

Dedicated cluster for development environments and low-traffic applications.

STORAGE	RAM	vCPU
10 GB	2 GB	2 vCPUs

☐

**Serverless**

**\$0.12/1M reads**

For application development and testing, or workloads with variable traffic.

STORAGE	RAM	vCPU
Up to 1TB	Auto-scale	Auto-scale

☒

**M0**

**Free**

For learning and exploring MongoDB in a cloud environment.

STORAGE	RAM	vCPU
512 MB	Shared	Shared

✓ **Free forever!** Your free cluster is ideal for experimenting in a limited sandbox. You can upgrade to a production cluster anytime.

Step 3: Configure your cluster by selecting the provider, region, and optional settings like security setup and sample dataset preloading.

**Free forever!** Your free cluster is ideal for experimenting in a limited sandbox. You can upgrade to a production cluster anytime.

### Configurations

**Name**  
You cannot change the name once the cluster is created.

**Provider**

**Region**

**Mumbai (ap-south-1)** ★

★ Recommended ⓘ Low carbon emissions ⓘ

**Tag (optional)**  
Create your first tag to categorize and label your resources; more tags can be added later. [Learn more.](#)

:

### Quick setup

☒ Automate security setup ⓘ

☒ Preload sample dataset ⓘ

Step 4: Add members to your project, assign their permissions, and finalize by creating the project.

Access Manager ▼
 Billing

### Add Members and Set Permissions

Give your members access permissions below.

bhagchandani.niraj@gmail  
.com (you)

Project Owner ▼

Back

Cancel

Create Project

Step 5: Secure your cluster by adding a connection IP address and creating a database user with credentials.

## Connect to Cluster0

1

2

3

Set up connection securityChoose a connection methodConnect

You need to secure your MongoDB Atlas cluster before you can use it. Set which users and IP addresses can access your cluster now. [Read more](#)

- Add a connection IP address**

✓ Your current IP address (220.158.144.59) has been added to enable local connectivity. Only an IP address you add to your Access List will be able to connect to your project's clusters. Add more later in [Network Access](#).
- Create a database user**

This first user will have [atlasAdmin](#) permissions for this project.

We autogenerated a username and password. You can use this or create your own.

**i** You'll need your database user's credentials in the next step. Copy the database user password.

**Username**

g23ai2087

**Password**

iitj123

HIDE

Copy

Create Database User

Close

Choose a connection method

Step 6: Verify connection security by ensuring an IP address is added and a database user is created for accessing the cluster.

### Connect to Cluster0

1

2

3

Set up connection security

Choose a connection method

Connect

You need to secure your MongoDB Atlas cluster before you can use it. Set which users and IP addresses can access your cluster now. [Read more](#)

- Add a connection IP address**

✓ Your current IP address (220.158.144.59) has been added to enable local connectivity. Only an IP address you add to your Access List will be able to connect to your project's clusters. Add more later in [Network Access](#).
- Create a database user**

✓ A database user has been added to this project. Create another user later in [Database Access](#).

You'll need your database user's credentials in the next step.

Close

Choose a connection method

1. Write the method load() to load the TPC-H customer and orders data into separate collections (like how it would be stored in a relational model). The data files are in the data folder.

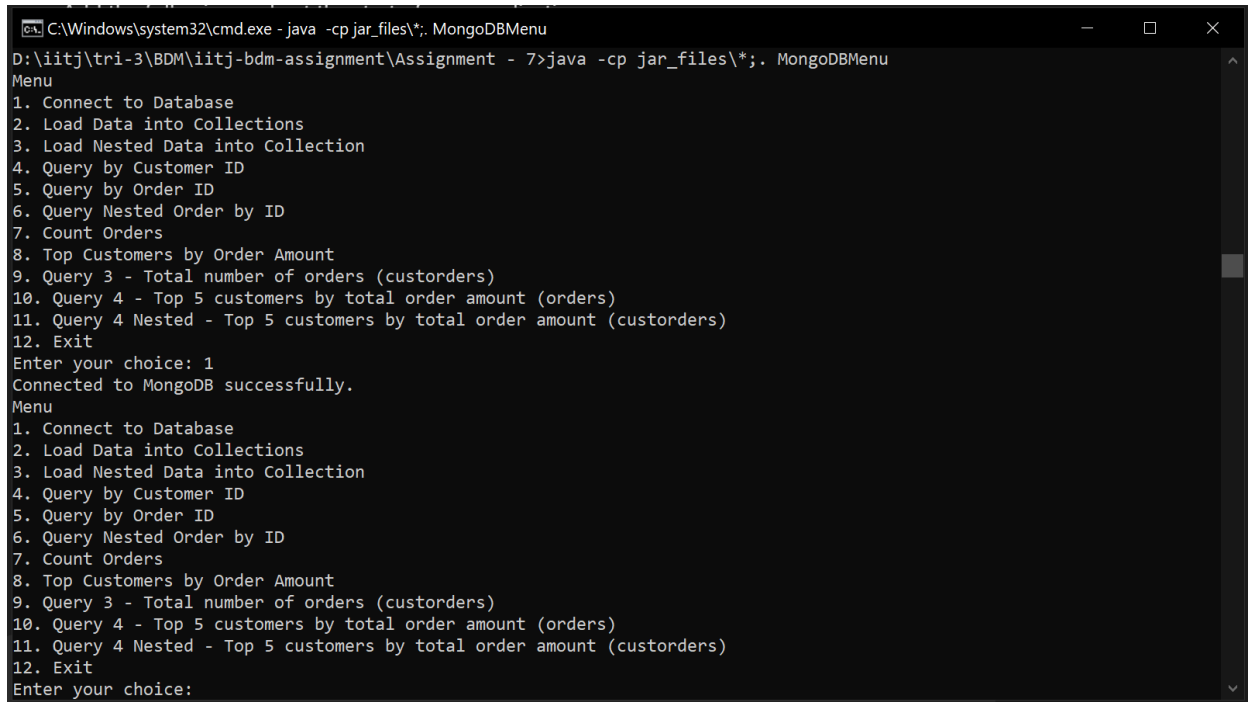
Code:

```
public void load() {
    try (BufferedReader customerReader = new BufferedReader(new
    FileReader("data/customer.tbl"));
        BufferedReader ordersReader = new BufferedReader(new
    FileReader("data/order.tbl"))) {

        MongoClient<Document> cc = database.getCollection("customer");
        customerReader.lines().forEach(line -> {
            String[] parts = line.split("\\|");
            Document cd = new Document("custkey", Integer.parseInt(parts[0]))
                .append("name", parts[1])
                .append("address", parts[2])
                .append("nationkey", Integer.parseInt(parts[3]))
                .append("phone", parts[4])
                .append("acctbal", Double.parseDouble(parts[5]))
                .append("mktsegment", parts[6])
                .append("comment", parts[7]);
            cc.insertOne(cd);
        });

        MongoClient<Document> oc = database.getCollection("orders");
        ordersReader.lines().forEach(line -> {
            String[] parts = line.split("\\|");
            Document orderDoc = new Document("orderkey", Integer.parseInt(parts[0]))
                .append("custkey", Integer.parseInt(parts[1]))
                .append("orderstatus", parts[2])
                .append("totalprice", Double.parseDouble(parts[3]))
                .append("orderdate", parts[4])
                .append("orderpriority", parts[5])
                .append("clerk", parts[6])
                .append("shippriority", Integer.parseInt(parts[7]))
                .append("comment", parts[8]);
            oc.insertOne(orderDoc);
        });

        System.out.println("Data loaded successfully.");
    } catch (Exception e) {
        System.out.println("Error while loading data: " + e.getMessage());
    }
}
```



```

C:\Windows\system32\cmd.exe - java -cp jar_files\*. MongoDBMenu
D:\iitj\tri-3\BDM\iitj-bdm-assignment\Assignment - 7>java -cp jar_files\*. MongoDBMenu
Menu
1. Connect to Database
2. Load Data into Collections
3. Load Nested Data into Collection
4. Query by Customer ID
5. Query by Order ID
6. Query Nested Order by ID
7. Count Orders
8. Top Customers by Order Amount
9. Query 3 - Total number of orders (custorders)
10. Query 4 - Top 5 customers by total order amount (orders)
11. Query 4 Nested - Top 5 customers by total order amount (custorders)
12. Exit
Enter your choice: 1
Connected to MongoDB successfully.
Menu
1. Connect to Database
2. Load Data into Collections
3. Load Nested Data into Collection
4. Query by Customer ID
5. Query by Order ID
6. Query Nested Order by ID
7. Count Orders
8. Top Customers by Order Amount
9. Query 3 - Total number of orders (custorders)
10. Query 4 - Top 5 customers by total order amount (orders)
11. Query 4 Nested - Top 5 customers by total order amount (custorders)
12. Exit
Enter your choice:

```

Fig. 7.1: Command-line interface demonstrating MongoDB menu operations, including database connection, data loading, and query execution options.

- Write the method `loadNest()` to load the TPC-H customer and order data into a nested collection called `custorders` where each document contains the customer information and all orders for that customer.

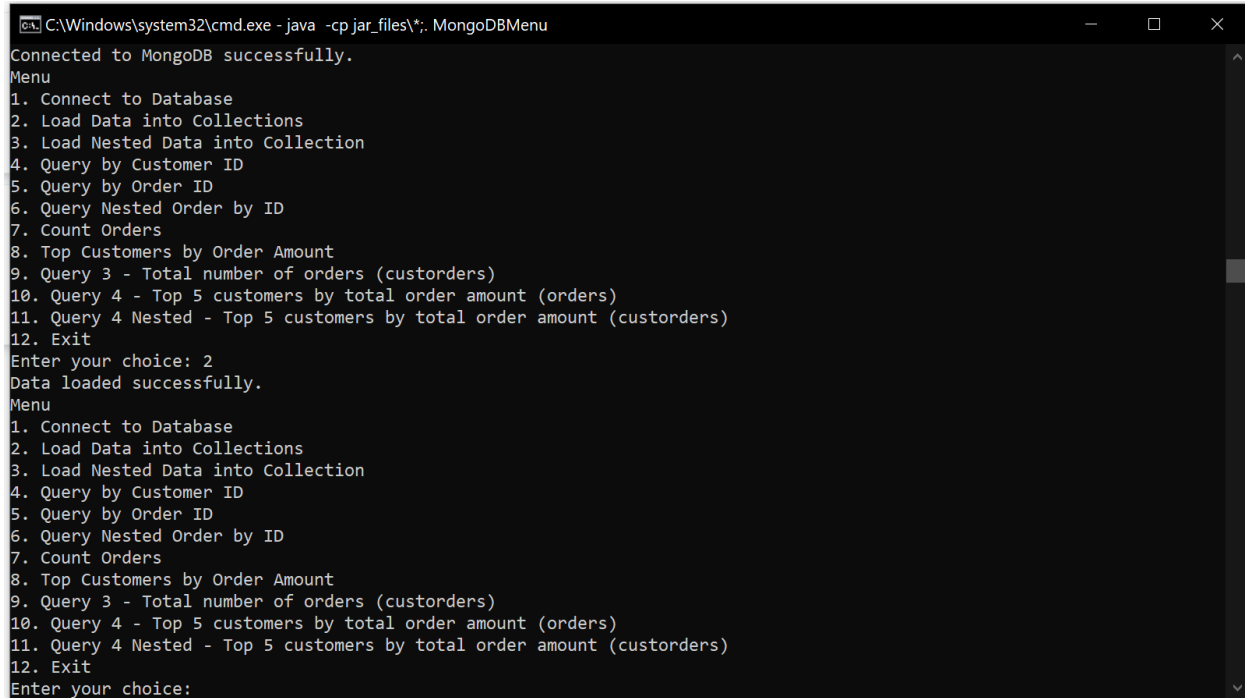
```

public void loadNestedData() throws Exception {
    try {
        List<Document> customers = loadDataFromFile("data/customer.tbl", true);
        List<Document> orders = loadDataFromFile("data/order.tbl", false);
        Map<Integer, List<Document>> customerOrdersMap = mapOrdCust(orders);

        List<Document> col = combineCustomerOrders(customers, customerOrdersMap);

        MongoCollection<Document> collection = database.getCollection("custorders");
        collection.insertMany(col);
        System.out.println("Nested data loaded successfully.");
    } catch (Exception e) {
        System.out.println("Error while loading nested data: " + e.getMessage());
        throw new Exception("Error loading nested customer and order data", e);
    }
}

```



```

C:\Windows\system32\cmd.exe - java -cp jar_files\*. MongoDBMenu
Connected to MongoDB successfully.
Menu
1. Connect to Database
2. Load Data into Collections
3. Load Nested Data into Collection
4. Query by Customer ID
5. Query by Order ID
6. Query Nested Order by ID
7. Count Orders
8. Top Customers by Order Amount
9. Query 3 - Total number of orders (custorders)
10. Query 4 - Top 5 customers by total order amount (orders)
11. Query 4 Nested - Top 5 customers by total order amount (custorders)
12. Exit
Enter your choice: 2
Data loaded successfully.
Menu
1. Connect to Database
2. Load Data into Collections
3. Load Nested Data into Collection
4. Query by Customer ID
5. Query by Order ID
6. Query Nested Order by ID
7. Count Orders
8. Top Customers by Order Amount
9. Query 3 - Total number of orders (custorders)
10. Query 4 - Top 5 customers by total order amount (orders)
11. Query 4 Nested - Top 5 customers by total order amount (custorders)
12. Exit
Enter your choice:

```

Fig. 7.2: Command-line interface showing successful data loading into MongoDB collections and available menu options for executing various database queries.



Fig. 7.3: MongoDB cluster structure displaying databases and collections, including "db-g23ai2087" with collections "customer" and "orders."



	_id ObjectId	custkey Int32	name String	address String	nationkey Int32
1	ObjectId('675d4a9f791b565...')	1	"Customer#000000001"	"ANhzAAh6R3 gLS4Sx"	15
2	ObjectId('675d4a9f791b565...')	2	"Customer#000000002"	"MN0L30zNgyLx2"	13
3	ObjectId('675d4a9f791b565...')	3	"Customer#000000003"	"PSL745NCwwN20N66Lxgnw7mR...	1
4	ObjectId('675d4a9f791b565...')	4	"Customer#000000004"	"mknn1Sh0NPmZ1k5LwZ0B m0"	4
5	ObjectId('675d4a9f791b565...')	5	"Customer#000000005"	"yOww5znhPN150LQNPChkLx2B...	3
6	ObjectId('675d4a9f791b565...')	6	"Customer#000000006"	"n570ykL4n k51ik3R5wLNzjn...	20
7	ObjectId('675d4a9f791b565...')	7	"Customer#000000007"	"ChljB040gAizN6kQhRi7Ljj...	18
8	ObjectId('675d4a9f791b565...')	8	"Customer#000000008"	"kCRz0CknMw7mh4P50QjBnxSL...	17
9	ObjectId('675d4a9f791b565...')	9	"Customer#000000009"	"L4z65g2RN6PxM5krjnPB7k...	8
10	ObjectId('675d4a9f791b565...')	10	"Customer#000000010"	"L3jg3xAwI6A0B103B0Aymm"	5
11	ObjectId('675d4a9f791b565...')	11	"Customer#000000011"	"BCP1yzB0x4"	23
12	ObjectId('675d4a9f791b565...')	12	"Customer#000000012"	"LS2BNB2QxNc1AyLjMkCmx76...	13
13	ObjectId('675d4a9f791b565...')	13	"Customer#000000013"	"wMIAIC75QI6Ljx2N6C"	3
14	ObjectId('675d4a9f791b565...')	14	"Customer#000000014"	"l2Ajl Q3jizLz4L1 n2y lyB...	1

Fig. 7.4: Sample data from the "customer" collection in MongoDB, displaying fields such as \_id, custkey, name, address, and nationkey.

	_id ObjectId	orderkey Int32	custkey Int32	orderstatus String	totalprice Double
1	ObjectId('675d4b50791b565...')	1	781	"O"	172799.49
2	ObjectId('675d4b50791b565...')	2	1234	"O"	41048.98
3	ObjectId('675d4b50791b565...')	3	445	"F"	250870.73
4	ObjectId('675d4b50791b565...')	4	557	"O"	6705.3
5	ObjectId('675d4b50791b565...')	5	392	"F"	120227.38
6	ObjectId('675d4b50791b565...')	6	1301	"F"	2506.57
7	ObjectId('675d4b50791b565...')	7	670	"O"	235483.33
8	ObjectId('675d4b50791b565...')	32	611	"O"	197403.12
9	ObjectId('675d4b50791b565...')	33	1276	"F"	72247.39
10	ObjectId('675d4b50791b565...')	34	1153	"O"	131039.87
11	ObjectId('675d4b50791b565...')	35	862	"O"	171884.36
12	ObjectId('675d4b50791b565...')	36	1249	"O"	81904.33
13	ObjectId('675d4b50791b565...')	37	818	"F"	150846.36
14	ObjectId('675d4b51791b565...')	38	322	"O"	7219.48

Fig. 7.5: Sample data from the "orders" collection in MongoDB, displaying fields such as \_id, orderkey, custkey, orderstatus, and totalprice.

3. Write the method query1() that returns the customer name given a customer id using the customer collection.

Code:

```
public String query1(int customerKey) {
    MongoClient<Document> collection = database.getCollection("customer");
    Document customer = collection.find(eq("custkey", customerKey)).first();
    return customer != null ? customer.getString("name") : "Customer not found";
}
```

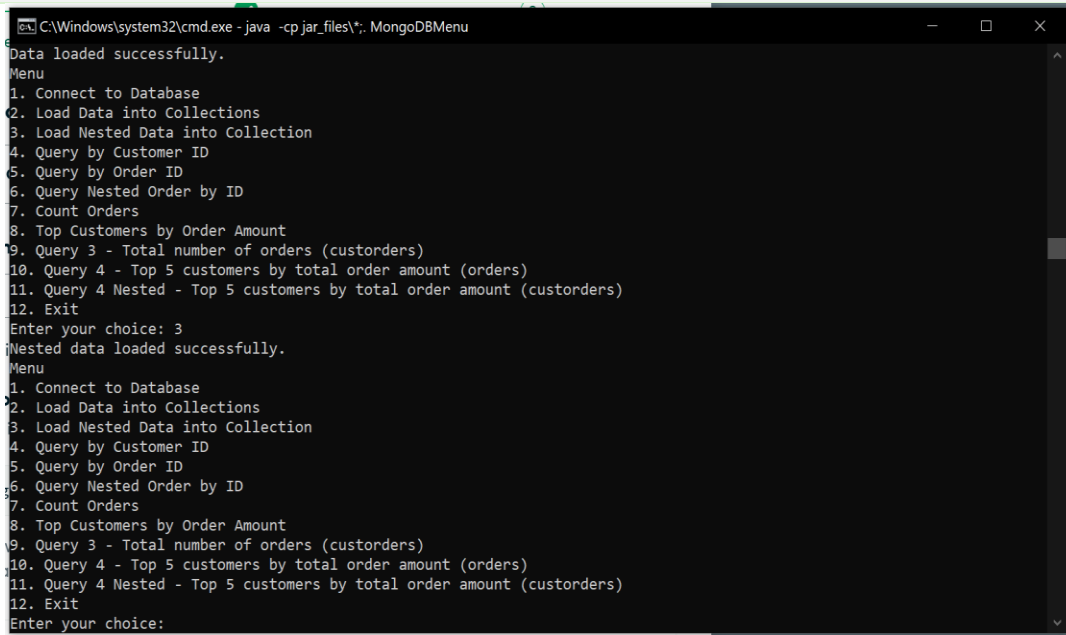


Fig. 7.6: Command-line interface illustrating the successful loading of nested data into MongoDB collections, with menu options available for further database queries and operations.

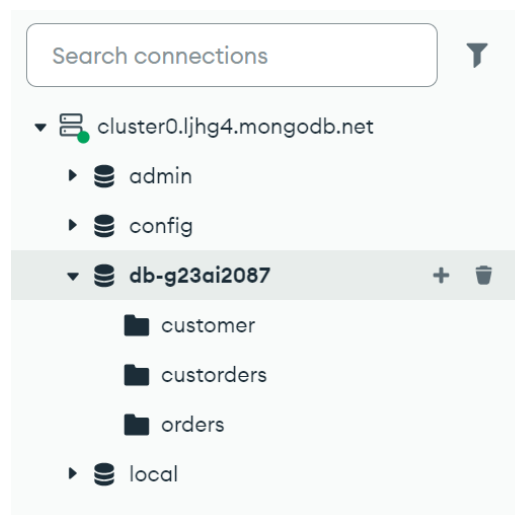


Fig. 7.7: MongoDB cluster structure displaying the database "db-g23ai2087" with collections including "customer," "custorders," and "orders," alongside administrative databases like "admin" and "config."

ADD DATA

EXPORT DATA

UPDATE

DELETE

100

1 - 100 of 1500

custorders

	_id	ObjectId	custkey	Int32	name	String	address	String	nationkey	Int32	
1	ObjectId('675d55e4791b565...	1	"Customer#000000001"	"ANhzAAh6R3 gLS4Sx"	15						
2	ObjectId('675d55e4791b565...	2	"Customer#000000002"	"MN0L30zNgyLx2"	13						
3	ObjectId('675d55e4791b565...	3	"Customer#000000003"	"PSL74SNCwwN2ON66Lxgnw7mR...	1						
4	ObjectId('675d55e4791b565...	4	"Customer#000000004"	"mknn1Sh0NPMz1k5Lw20B m0"	4						
5	ObjectId('675d55e4791b565...	5	"Customer#000000005"	"yOww5znHPN150LQNPChkLx2B...	3						
6	ObjectId('675d55e4791b565...	6	"Customer#000000006"	"nS70ykL4n k51ik3R5wLNzjn...	20						
7	ObjectId('675d55e4791b565...	7	"Customer#000000007"	"ChLjB040gAizN6kQhRi7LjJ...	18						
8	ObjectId('675d55e4791b565...	8	"Customer#000000008"	"kCRz0CknMw7mh4P50QjBnxSL...	17						
9	ObjectId('675d55e4791b565...	9	"Customer#000000009"	"L4z65g2RN6PxM5kRjnPB7k...	8						
10	ObjectId('675d55e4791b565...	10	"Customer#000000010"	"L3jg3xAwi6A0B103B0Aymm"	5						
11	ObjectId('675d55e4791b565...	11	"Customer#000000011"	"BCP1yzB0x4"	23						
12	ObjectId('675d55e4791b565...	12	"Customer#000000012"	"L52BNB2QxNxCiAyLjMkCmx76...	13						
13	ObjectId('675d55e4791b565...	13	"Customer#000000013"	"wM1A1C750i6Ljx2N6C"	3						
14	ObjectId('675d55e4791b565...	14	"Customer#000000014"	"L2AjL Q3jizLz4L1 n2y lyB...	1						

Fig.7.8: Sample data from the "custorders" collection in MongoDB, showcasing fields such as \_id, custkey, name, address, and nationkey.

4. Write the method query2() that returns the order date for a given order id using the orders collection.

Code:

```
public String query2(int orderId) {
    MongoCollection<Document> collection = database.getCollection("orders");
    Document order = collection.find(eq("orderkey", orderId)).first();
    return order != null ? order.getString("orderdate") : "Order not found";
}
```

```
C:\Windows\system32\cmd.exe - java -cp jar_files\*. MongoDBMenu
Menu
1. Connect to Database
2. Load Data into Collections
3. Load Nested Data into Collection
4. Query by Customer ID
5. Query by Order ID
6. Query Nested Order by ID
7. Count Orders
8. Top Customers by Order Amount
9. Query 3 - Total number of orders (custorders)
10. Query 4 - Top 5 customers by total order amount (orders)
11. Query 4 Nested - Top 5 customers by total order amount (custorders)
12. Exit
Enter your choice: 4
Enter Customer ID: 12
```

Fig. 7.9: Command-line interface executing Query 4 to retrieve the top 5 customers by total order amount from the "custorders" collection, with input for a specific customer ID.

5. Write the method query2Nest() that returns order date for a given order id using the custorders collection.

Code:

```
public String query2Nest(int orderId) {
    MongoClient<Document> collection = database.getCollection("custorders");
    List<Bson> pipeline = Arrays.asList(
        Aggregates.unwind("$orders"),
        Aggregates.match(Filters.eq("orders.orderkey", orderId)),
        Aggregates.project(Projections.fields(Projections.excludeId(),
        Projections.include("orders.orderdate")))
    );
    AggregateIterable<Document> result = collection.aggregate(pipeline);
    Document doc = result.first();
    return doc != null ? doc.get("orders", Document.class).getString("orderdate") :
    "Order not found";
}
```

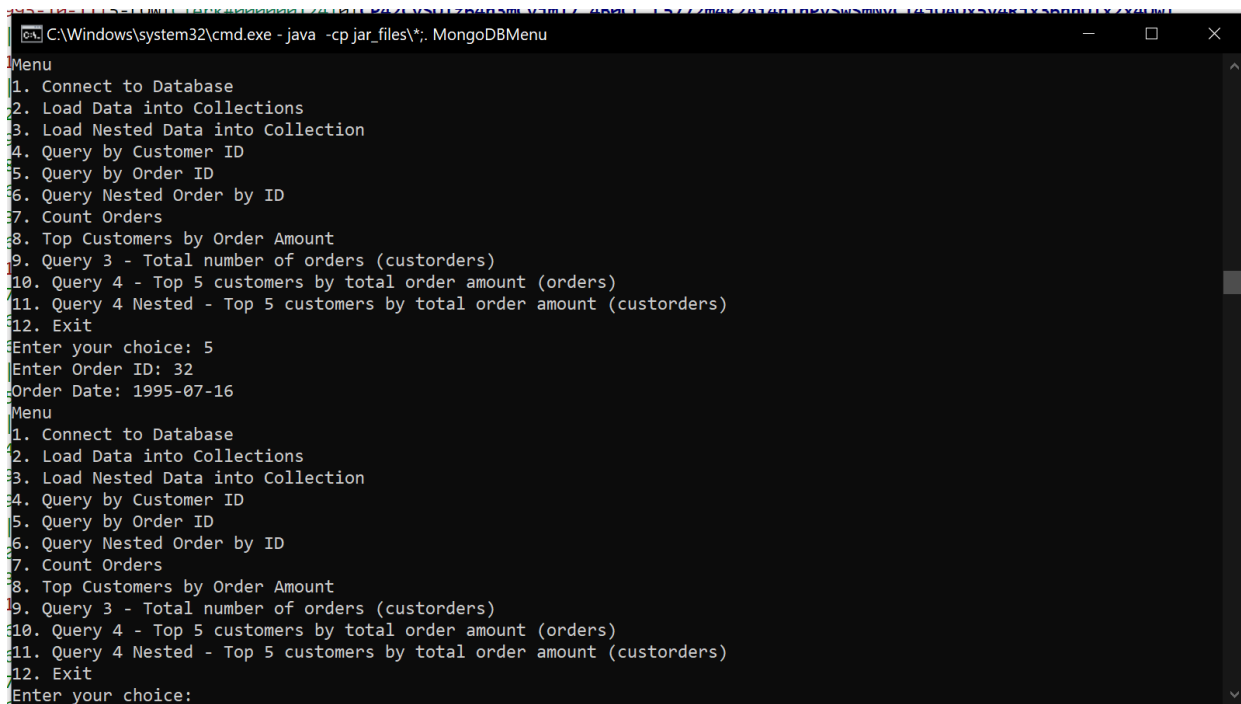


Fig. 7.10: Command-line interface executing Query 5 to retrieve nested order details by a specific Order ID, displaying the order date and returning to the menu for further operations.

Order Key	Total Price	Order Date	Order Status	Clerk
1 781 0 172799.49 1996-01-02 5-LOW Clerk#000000951 0 A0xCm5ARNL mxjChn2kC64x4A4L6zBg205jhg M42izyPO QlymNiky5kmSiSgBAQA	172799.49	1996-01-02	5-LOW	Clerk#000000951
2 1234 0 41048.98 1996-12-01 1-URGENT Clerk#000000880 0 5PRxL1nM7xhQNZP2hnjhy1zz ykhg4P2A MMg5Px3OCN 0B0iyCRgiC2	41048.98	1996-12-01	1-URGENT	Clerk#000000880
3 445 F 250870.73 1993-10-14 5-LOW Clerk#000000955 0 nm0kygQBnw7R53AAA4k	250870.73	1993-10-14	5-LOW	Clerk#000000955
4 557 0 6705.30 1995-10-11 5-LOW Clerk#000000124 0 CP42CySQLz64n3mCyjm17 4B0CL L5772m4k2Ai4h1nPySwSmNyCl4j0AOx5y4Rjx36nh01x2x4Qw	6705.30	1995-10-11	5-LOW	Clerk#000000124
5 392 F 120227.38 1994-07-30 5-LOW Clerk#000000925 0  3PNC7zMP534MSizgy34Bxj6210C7n6PBk7	120227.38	1994-07-30	5-LOW	Clerk#000000925
6 1301 F 2506.57 1992-02-21 4-NOT SPECIFIED Clerk#000000058 0 1CN00NA0z75SwwCxMNB0MLNL	2506.57	1992-02-21	4-NOT SPECIFIED	Clerk#000000058
7 670 0 235483.33 1996-01-10 2-HIGH Clerk#000000470 0 gmiC6hj5L4 0ixCAQkmB6giC1614L16g	235483.33	1996-01-10	2-HIGH	Clerk#000000470
8 611 0 197403.12 1995-07-16 2-HIGH Clerk#000000616 0 7ihNSz00NCxA31PPx6RM4ih BPPlz4175Lk3SRA1zx0nlikRgjkk	197403.12	1995-07-16	2-HIGH	Clerk#000000616
9 1276 F 72247.39 1993-10-27 3-MEDIUM Clerk#000000409 0 jkACLH 0igMiy72n Sky0h0B6NB70j7Q	72247.39	1993-10-27	3-MEDIUM	Clerk#000000409
10 1153 0 131039.87 1998-07-21 3-MEDIUM Clerk#000000223 0 05k 2x242klm jyA wB0CBzzQnz5P1nAm15AL5jC 1g5	131039.87	1998-07-21	3-MEDIUM	Clerk#000000223
11 862 0 171884.36 1995-10-23 4-NOT SPECIFIED Clerk#000000259 0 6hjNmSm23PAmnw1jjy040nOx8x3RB6ny7yNwzm5 21mi B6z3mM4P	171884.36	1995-10-23	4-NOT SPECIFIED	Clerk#000000259

Fig.7.11: Query results displaying detailed order information from the "orders" collection, including fields such as orderkey, totalprice, orderdate, and clerk, highlighting specific records based on query conditions.

6. Write the method query3() that returns the total number of orders using the orders collection.

Code:

```
public long countOrders() {
    MongoCollection<Document> collection = database.getCollection("orders");
    return collection.countDocuments();
}
```

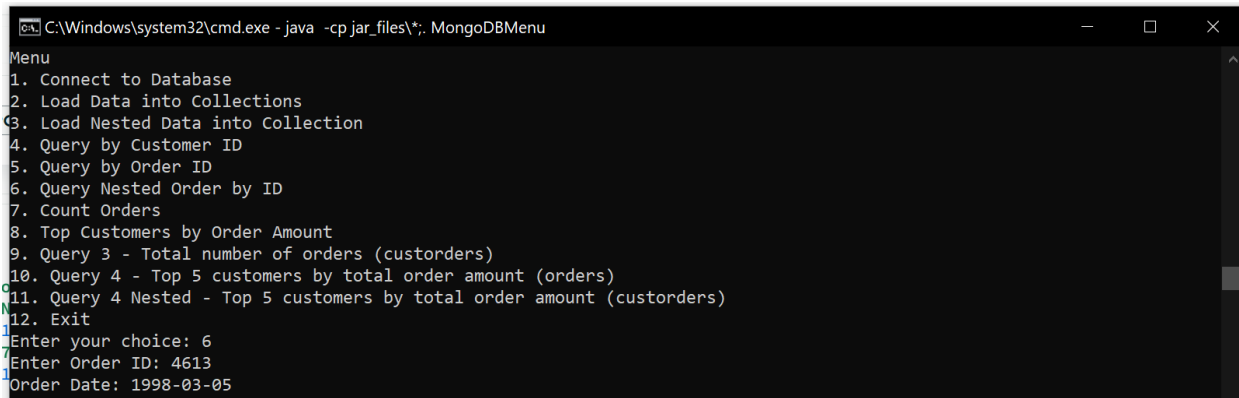


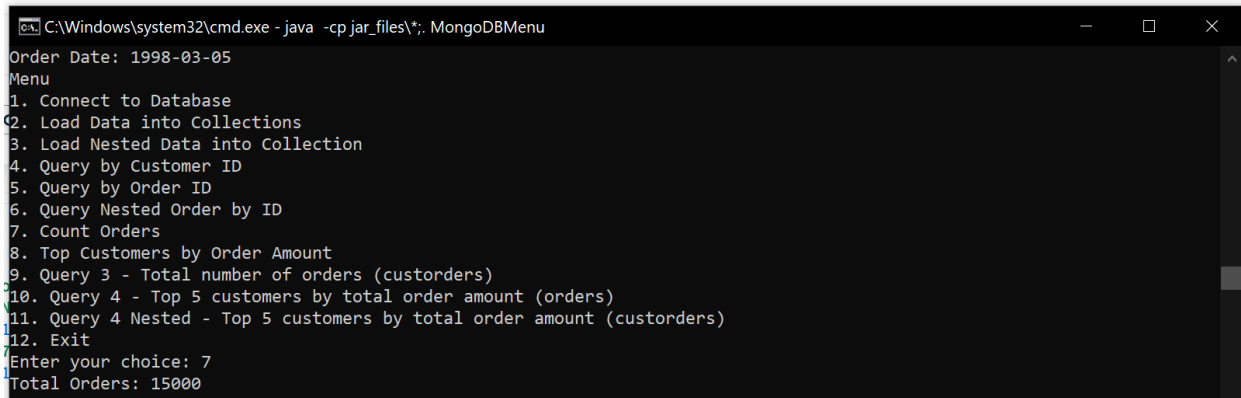
Fig. 7.12: Command-line interface executing Query 6 to fetch nested order details for a specific Order ID, displaying the corresponding order date and returning to the menu for additional operations.

7. Write the method query3Nest() that returns the total number of orders using the custorders collection.

Code:

```
public long query3Nest() {
    MongoCollection<Document> collection = database.getCollection("custorders");
    return collection.countDocuments(); // Returns the total number of documents in
the custorders collection
}
```

}



```

C:\Windows\system32\cmd.exe - java -cp jar_files\*. MongoDBMenu
Order Date: 1998-03-05
Menu
1. Connect to Database
2. Load Data into Collections
3. Load Nested Data into Collection
4. Query by Customer ID
5. Query by Order ID
6. Query Nested Order by ID
7. Count Orders
8. Top Customers by Order Amount
9. Query 3 - Total number of orders (custorders)
10. Query 4 - Top 5 customers by total order amount (orders)
11. Query 4 Nested - Top 5 customers by total order amount (custorders)
12. Exit
Enter your choice: 7
Total Orders: 15000

```

Fig. 7.13: Command-line interface executing Query 7 to count the total number of orders in the database, displaying the result as "Total Orders: 15000."

8. Write the method query4() that returns the top 5 customers based on total order amount using the customer and orders collections.

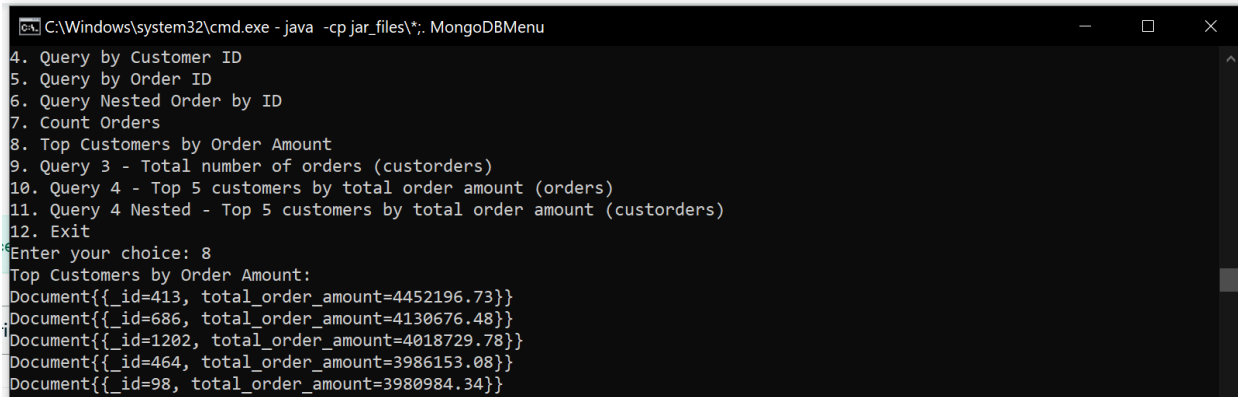
Code:

```

public List<Document> query4() {
    MongoCollection<Document> collection = database.getCollection("orders");
    List<Bson> pipeline = Arrays.asList(
        Aggregates.group("$custkey", Accumulators.sum("total_order_amount",
"$totalprice")), // Sum total order amounts
        Aggregates.sort(Sorts.descending("total_order_amount")),
        // Sort by total_order_amount
        Aggregates.limit(5)
        // Limit to top 5 customers
    );

    AggregateIterable<Document> result = collection.aggregate(pipeline);
    return result.into(new ArrayList<>());
}

```



```

C:\Windows\system32\cmd.exe - java -cp jar_files\*. MongoDBMenu
4. Query by Customer ID
5. Query by Order ID
6. Query Nested Order by ID
7. Count Orders
8. Top Customers by Order Amount
9. Query 3 - Total number of orders (custorders)
10. Query 4 - Top 5 customers by total order amount (orders)
11. Query 4 Nested - Top 5 customers by total order amount (custorders)
12. Exit
Enter your choice: 8
Top Customers by Order Amount:
Document{{_id=413, total_order_amount=4452196.73}}
Document{{_id=686, total_order_amount=4130676.48}}
Document{{_id=1202, total_order_amount=4018729.78}}
Document{{_id=464, total_order_amount=3986153.08}}
Document{{_id=98, total_order_amount=3980984.34}}

```

Fig.7.14: Command-line interface executing Query 8 to retrieve the top customers by total order amount, displaying the customer IDs and their corresponding total order amounts.

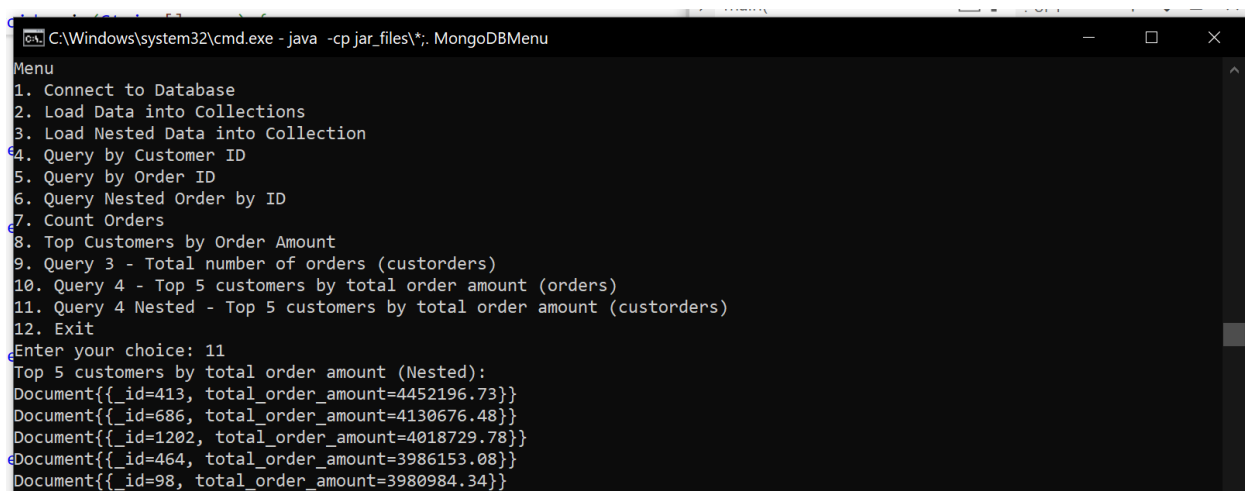
9. Write the method query4Nest() that returns the top 5 customers based on total order amount using the custorders collection.

Code:

```

public List<Document> query4Nest() {
    MongoCollection<Document> collection = database.getCollection("custorders");
    List<Bson> pipeline = Arrays.asList(
        Aggregates.unwind("$orders"),
        // Unwind orders array
        Aggregates.group("$custkey", Accumulators.sum("total_order_amount",
"$orders.totalprice")), // Sum total price in orders array
        Aggregates.sort(Sorts.descending("total_order_amount")),
        // Sort by total_order_amount
        Aggregates.limit(5)
        // Limit to top 5 customers
    );
}

```



```

C:\Windows\system32\cmd.exe - java -cp jar_files\*. MongoDBMenu
Menu
1. Connect to Database
2. Load Data into Collections
3. Load Nested Data into Collection
4. Query by Customer ID
5. Query by Order ID
6. Query Nested Order by ID
7. Count Orders
8. Top Customers by Order Amount
9. Query 3 - Total number of orders (custorders)
10. Query 4 - Top 5 customers by total order amount (orders)
11. Query 4 Nested - Top 5 customers by total order amount (custorders)
12. Exit
Enter your choice: 11
Top 5 customers by total order amount (Nested):
Document{{_id=413, total_order_amount=4452196.73}}
Document{{_id=686, total_order_amount=4130676.48}}
Document{{_id=1202, total_order_amount=4018729.78}}
Document{{_id=464, total_order_amount=3986153.08}}
Document{{_id=98, total_order_amount=3980984.34}}

```

Fig.7.15: Command-line interface executing Query 11 to retrieve the top 5 customers by total order amount using nested query logic, displaying customer IDs along with their total order amounts.



Full Code:

```
import com.mongodb.client.MongoClient;
import com.mongodb.client.MongoClients;
import com.mongodb.client.MongoCollection;
import com.mongodb.client.MongoDatabase;
import org.bson.Document;
import static com.mongodb.client.model.Filters.eq;
import com.mongodb.client.AggregateIterable;
import org.bson.conversions.Bson;
import com.mongodb.client.model.Aggregates;
import com.mongodb.client.model.Filters;
import com.mongodb.client.model.Projections;
import java.util.Arrays;
import java.util.List;
import java.util.ArrayList;
import java.util.HashMap;
import java.util.Map;
import java.util.Scanner;
import java.io.BufferedReader;
import java.io.FileReader;
import com.mongodb.client.model.Sorts;
import static com.mongodb.client.model.Accumulators.*;
import static com.mongodb.client.model.Accumulators.sum;
import com.mongodb.client.model.Accumulators;
import java.util.logging.Level;
import java.util.logging.Logger;

public class MongoDBMenu {
    private MongoClient client;
    private MongoDatabase database;

    public void MongoDB() {
        this.database = database;
    }

    public void connect() {
        try {
            String cs = "mongodb+srv://db-
g23ai2087:iitj123@cluster0.ljhg4.mongodb.net/?retryWrites=true&w=majority&appName=Cluster0
";
            this.client = MongoClients.create(cs);
            this.database = client.getDatabase("db-g23ai2087");
            System.out.println("Connected to MongoDB successfully.");
        }
    }
}
```



```

    } catch (Exception e) {
        System.out.println("Error while connecting: " + e.getMessage());
    }
}

public void load() {
    try (BufferedReader customerReader = new BufferedReader(new
FileReader("data/customer.tbl"));
        BufferedReader ordersReader = new BufferedReader(new
FileReader("data/order.tbl"))) {

        MongoClient<Document> cc = database.getCollection("customer");
        customerReader.lines().forEach(line -> {
            String[] parts = line.split("\\|");
            Document cd = new Document("custkey", Integer.parseInt(parts[0]))
                .append("name", parts[1])
                .append("address", parts[2])
                .append("nationkey", Integer.parseInt(parts[3]))
                .append("phone", parts[4])
                .append("acctbal", Double.parseDouble(parts[5]))
                .append("mktsegment", parts[6])
                .append("comment", parts[7]);
            cc.insertOne(cd);
        });

        MongoClient<Document> oc = database.getCollection("orders");
        ordersReader.lines().forEach(line -> {
            String[] parts = line.split("\\|");
            Document orderDoc = new Document("orderkey", Integer.parseInt(parts[0]))
                .append("custkey", Integer.parseInt(parts[1]))
                .append("orderstatus", parts[2])
                .append("totalprice", Double.parseDouble(parts[3]))
                .append("orderdate", parts[4])
                .append("orderpriority", parts[5])
                .append("clerk", parts[6])
                .append("shippriority", Integer.parseInt(parts[7]))
                .append("comment", parts[8]);
            oc.insertOne(orderDoc);
        });

        System.out.println("Data loaded successfully.");
    } catch (Exception e) {
        System.out.println("Error while loading data: " + e.getMessage());
    }
}

```

```

public void loadNestedData() throws Exception {
    try {
        List<Document> customers = loadDataFromFile("data/customer.tbl", true);
        List<Document> orders = loadDataFromFile("data/order.tbl", false);
        Map<Integer, List<Document>> customerOrdersMap = mapOrdCust(orders);

        List<Document> col = combineCustomerOrders(customers, customerOrdersMap);

        MongoCollection<Document> collection = database.getCollection("custorders");
        collection.insertMany(col);
        System.out.println("Nested data loaded successfully.");
    } catch (Exception e) {
        System.out.println("Error while loading nested data: " + e.getMessage());
        throw new Exception("Error loading nested customer and order data", e);
    }
}

private List<Document> loadDataFromFile(String fileName, boolean isCustomerData)
throws Exception {
    List<Document> dl = new ArrayList<>();
    try (BufferedReader reader = new BufferedReader(new FileReader(fileName))) {
        String line;
        while ((line = reader.readLine()) != null) {
            String[] data = line.split("\\|");
            try {
                Document document = isCustomerData ? createCustomerDocument(data) :
createOrderDocument(data);
                dl.add(document);
            } catch (Exception e) {
                System.out.println("Skipping invalid record: " + line);
            }
        }
    }
    return dl;
}

private Document createCustomerDocument(String[] data) {
    return new Document()
        .append("custkey", parseIntSafe(data[0]))
        .append("name", data[1])
        .append("address", data[2])
        .append("nationkey", parseIntSafe(data[3]))
        .append("phone", data[4])
        .append("acctbal", parseDoubleSafe(data[5]))
        .append("mktsegment", data[6])
        .append("comment", data[7]);
}

```

```

    }

    private Document createOrderDocument(String[] data) {
        return new Document()
            .append("orderkey", parseIntSafe(data[0]))
            .append("custkey", parseIntSafe(data[1]))
            .append("orderstatus", data[2])
            .append("totalprice", parseDoubleSafe(data[3]))
            .append("orderdate", data[4])
            .append("orderpriority", data[5])
            .append("clerk", data[6])
            .append("shippriority", parseIntSafe(data[7]))
            .append("comment", data[8]);
    }

    private Map<Integer, List<Document>> mapOrdCust(List<Document> orders) {
        Map<Integer, List<Document>> map = new HashMap<>();
        for (Document order : orders) {
            int custKey = order.getInteger("custkey");
            map.computeIfAbsent(custKey, k -> new ArrayList<>()).add(order);
        }
        return map;
    }

    private List<Document> combineCustomerOrders(List<Document> customers, Map<Integer,
List<Document>> customerOrdersMap) {
        List<Document> combinedList = new ArrayList<>();
        for (Document customer : customers) {
            int custKey = customer.getInteger("custkey");
            List<Document> orders = customerOrdersMap.get(custKey);
            if (orders != null) {
                customer.append("orders", orders);
            }
            combinedList.add(customer);
        }
        return combinedList;
    }

    private int parseIntSafe(String value) {
        try {
            return Integer.parseInt(value);
        } catch (NumberFormatException e) {
            System.out.println("Invalid integer: " + value);
            return 0;
        }
    }
}

```

```

private double parseDoubleSafe(String value) {
    try {
        return Double.parseDouble(value);
    } catch (NumberFormatException e) {
        System.out.println("Invalid double: " + value);
        return 0.0;
    }
}

public String query1(int customerKey) {
    MongoCollection<Document> collection = database.getCollection("customer");
    Document customer = collection.find(eq("custkey", customerKey)).first();
    return customer != null ? customer.getString("name") : "Customer not found";
}

public String query2(int orderId) {
    MongoCollection<Document> collection = database.getCollection("orders");
    Document order = collection.find(eq("orderkey", orderId)).first();
    return order != null ? order.getString("orderdate") : "Order not found";
}

public String query2Nest(int orderId) {
    MongoCollection<Document> collection = database.getCollection("custorders");
    List<Bson> pipeline = Arrays.asList(
        Aggregates.unwind("$orders"),
        Aggregates.match(Filters.eq("orders.orderkey", orderId)),
        Aggregates.project(Projections.fields(Projections.excludeId(),
Projections.include("orders.orderdate"))))
    );
    AggregateIterable<Document> result = collection.aggregate(pipeline);
    Document doc = result.first();
    return doc != null ? doc.get("orders", Document.class).getString("orderdate") :
"Order not found";
}

public long query3() {
    MongoCollection<Document> collection = database.getCollection("orders");
    return collection.countDocuments();
}

public long countNestedOrders() {
    MongoCollection<Document> collection = database.getCollection("custorders");
    List<Bson> pipeline = Arrays.asList(
        Aggregates.unwind("$orders"),
        Aggregates.count("totalOrders")
    );
}

```

```

    );
    AggregateIterable<Document> result = collection.aggregate(pipeline);
    Document doc = result.first();
    return doc != null ? doc.getLong("totalOrders") : 0;
}

public List<Document> topCustomersByOrderAmount() {
    MongoCollection<Document> collection = database.getCollection("orders");

    List<Bson> pipeline = Arrays.asList(
        Aggregates.group("$custkey", sum("total_order_amount", "$totalprice")),
        Aggregates.sort(Sorts.descending("total_order_amount")),
        Aggregates.limit(5)
    );

    AggregateIterable<Document> result = collection.aggregate(pipeline);
    return result.into(new ArrayList<>());
}

public List<Document> topNestedCustomersByOrderAmount() {
    MongoCollection<Document> collection = database.getCollection("custorders");

    List<Bson> pipeline = Arrays.asList(
        Aggregates.unwind("$orders"),
        Aggregates.group("$custkey", sum("total_order_amount", "$orders.totalprice")),
        Aggregates.sort(Sorts.descending("total_order_amount")),
        Aggregates.limit(5)
    );

    AggregateIterable<Document> result = collection.aggregate(pipeline);
    return result.into(new ArrayList<>());
}

public long query3Nest() {
    MongoCollection<Document> collection = database.getCollection("custorders");
    return collection.countDocuments();
}

// 8. Query 4 - Top 5 customers by total order amount using customer and orders
collections
public List<Document> query4() {
    MongoCollection<Document> collection = database.getCollection("orders");
    List<Bson> pipeline = Arrays.asList(
        Aggregates.group("$custkey", Accumulators.sum("total_order_amount",
"$totalprice")),

```

```

        Aggregates.sort(Sorts.descending("total_order_amount")),

        Aggregates.limit(5)

    );

    AggregateIterable<Document> result = collection.aggregate(pipeline);
    return result.into(new ArrayList<>());
}

// 9. Query 4 Nested - Top 5 customers by total order amount using custorders
collection
public List<Document> query4Nest() {
    MongoCollection<Document> collection = database.getCollection("custorders");
    List<Bson> pipeline = Arrays.asList(
        Aggregates.unwind("$orders"),

        Aggregates.group("$custkey", Accumulators.sum("total_order_amount",
"$orders.totalprice")),
        Aggregates.sort(Sorts.descending("total_order_amount")),

        Aggregates.limit(5)

    );

    AggregateIterable<Document> result = collection.aggregate(pipeline);
    return result.into(new ArrayList<>());
}

public static void main(String[] args) {
    Scanner scanner = new Scanner(System.in);
    MongoDBMenu app = new MongoDBMenu();
    Logger mongoLogger = Logger.getLogger("org.mongodb.driver");
    mongoLogger.setLevel(Level.OFF);
    int choice = 0;
    while (choice != 9) {
        System.out.println("Menu");
        System.out.println("1. Connect to Database");
        System.out.println("2. Load Data into Collections");
        System.out.println("3. Load Nested Data into Collection");
        System.out.println("4. Query by Customer ID");
        System.out.println("5. Query by Order ID");
        System.out.println("6. Query Nested Order by ID");
        System.out.println("7. Count Orders");
        System.out.println("8. Top Customers by Order Amount");
        System.out.println("9. Query 3 - Total number of orders (custorders)");
    }
}

```

```

        System.out.println("10. Query 4 - Top 5 customers by total order amount
(orders)");
        System.out.println("11. Query 4 Nested - Top 5 customers by total order amount
(custorders)");
        System.out.println("12. Exit");
        System.out.print("Enter your choice: ");
        choice = scanner.nextInt();
        switch (choice) {
            case 1:
                app.connect();
                break;
            case 2:
                app.load();
                break;
            case 3:
                try {
                    app.loadNestedData();
                } catch (Exception e) {
                    System.out.println(e.getMessage());
                }
                break;
            case 4:
                System.out.print("Enter Customer ID: ");
                int customerId = scanner.nextInt();
                System.out.println("Customer Name: " + app.query1(customerId));
                break;
            case 5:
                System.out.print("Enter Order ID: ");
                int orderId = scanner.nextInt();
                System.out.println("Order Date: " + app.query2(orderId));
                break;
            case 6:
                System.out.print("Enter Order ID: ");
                int orderIdNested = scanner.nextInt();
                System.out.println("Order Date: " + app.query2Nest(orderIdNested));
                break;
            case 7:
                System.out.println("Total Orders: " + app.query3());
                break;
            case 8:
                System.out.println("Top Customers by Order Amount: ");
                List<Document> topCustomers = app.topCustomersByOrderAmount();
                topCustomers.forEach(System.out::println);
                break;
            case 9:
                long totalOrders = app.query3Nest();

```

```
        System.out.println("Total number of orders: " + totalOrders);
        break;
    case 10:
        topCustomers = app.query4();
        System.out.println("Top 5 customers by total order amount:");
        topCustomers.forEach(System.out::println);
        break;
    case 11:
        List<Document> topNestedCustomers = app.query4Nest();
        System.out.println("Top 5 customers by total order amount (Nested):");
        topNestedCustomers.forEach(System.out::println);
        break;
    case 12:
        System.out.println("Exiting... Goodbye!");
        break;
    default:
        System.out.println("Invalid choice. Please try again.");
    }
}
scanner.close();
}
```