**BIG DATA MANAGEMENT**

**POST GRADUATE DIPLOMA**

**IN DATA ENGINEERING**

## ASSIGNMENT 7

**SUBMITTED BY:**

**NIRAJ BHAGCHANDANI [G23AI2087]**

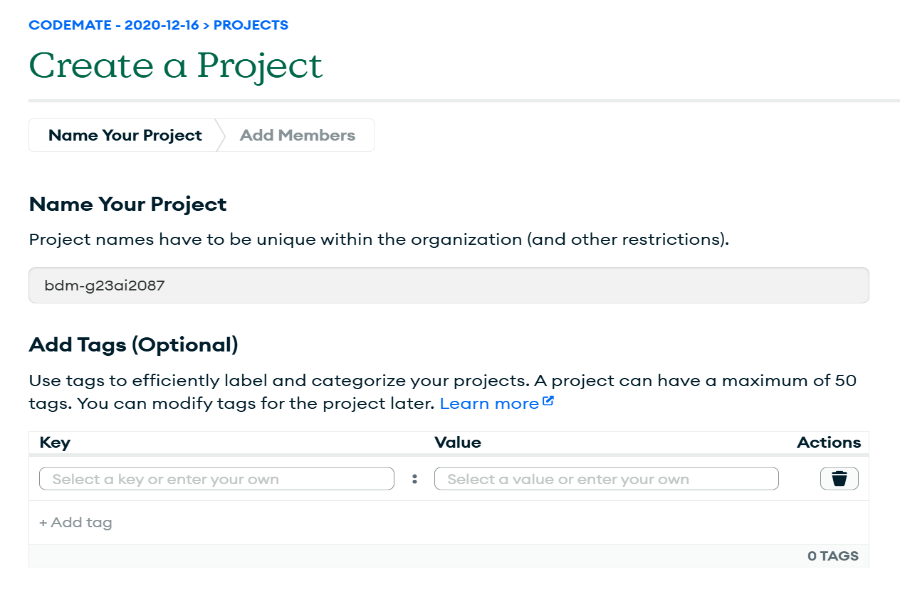
****

**SUBMISSION DATE: 15th December, 2024**

**DEPARTMENT OF AIDE**

**INDIAN INSTITUTE OF TECHNOLOGY, JODHPUR**

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



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

A screenshot of a computer

Description automatically generated

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

A screenshot of a computer

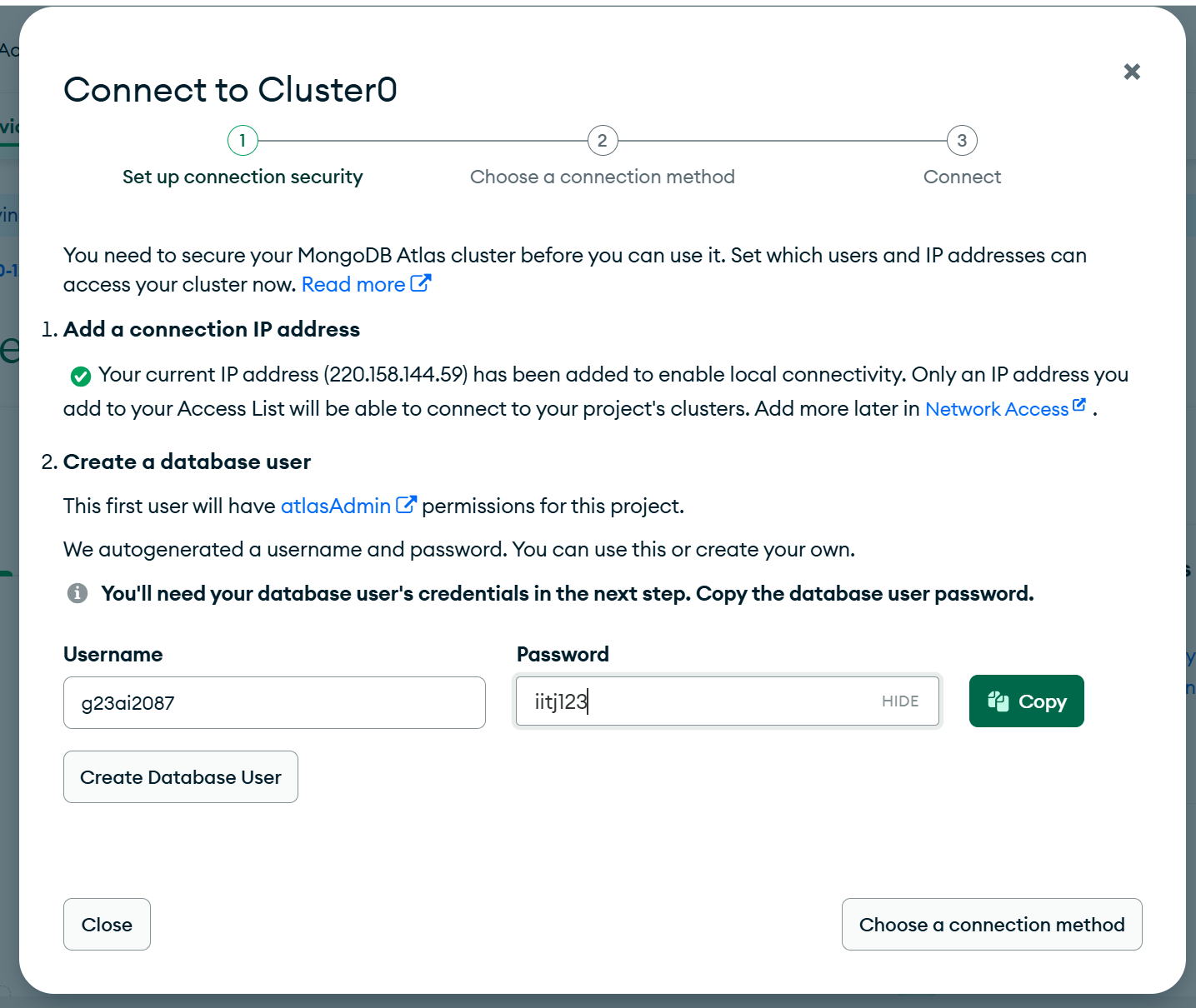
Description automatically generated

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

A screenshot of a computer

Description automatically generated

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



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

A screenshot of a computer

Description automatically generated

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"))) {

            MongoCollection<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);

            });

            MongoCollection<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);

            });

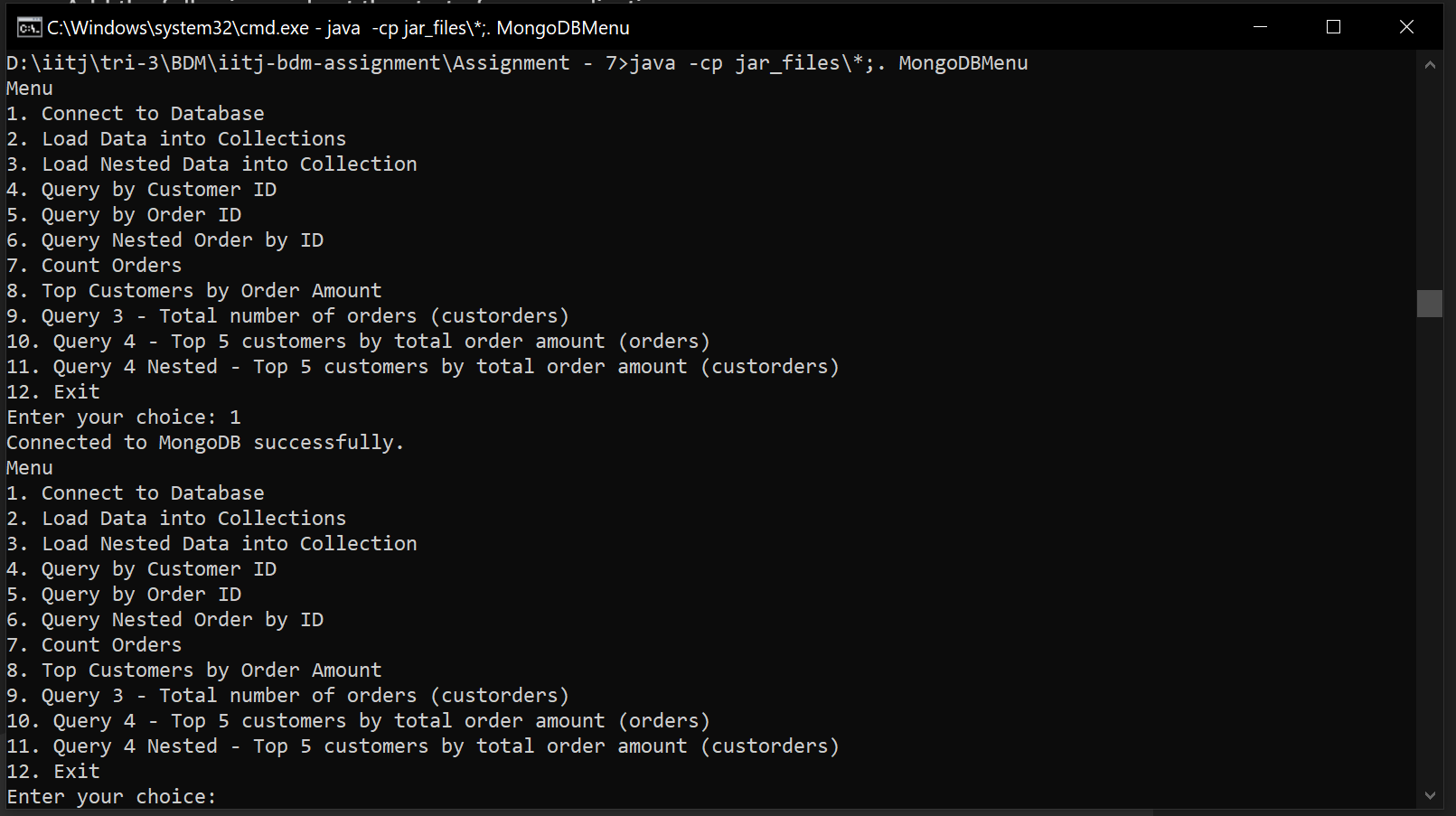
            System.out.println("Data loaded successfully.");

        } catch (Exception e) {

            System.out.println("Error while loading data: " + e.getMessage());

        }

    }



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

1. 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);

        }

    }

A screenshot of a computer

Description automatically generated

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

A screenshot of a computer

Description automatically generated

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

A screenshot of a computer

Description automatically generated

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

A screenshot of a computer

Description automatically generated

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

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

Code:

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";

    }

A screenshot of a computer

Description automatically generated

*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.*

A screenshot of a search box

Description automatically generated

*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."*

A screenshot of a computer

Description automatically generated

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

1. 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";

    }

A screen shot of a computer

Description automatically generated

*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.*

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

Code:

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";

    }

A screenshot of a computer program

Description automatically generated

*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.*

A screenshot of a computer program

Description automatically generated

*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.*

1. 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();

A computer screen with white text

Description automatically generated    }

*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.*

1. 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

    }

A screen shot of a computer

Description automatically generated

*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."*

1. 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<>());

    }

A screenshot of a computer

Description automatically generated

*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.*

1. 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

        );

A screen shot of a computer

Description automatically generated

*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"))) {

            MongoCollection<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);

            });

            MongoCollection<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();

    }

}