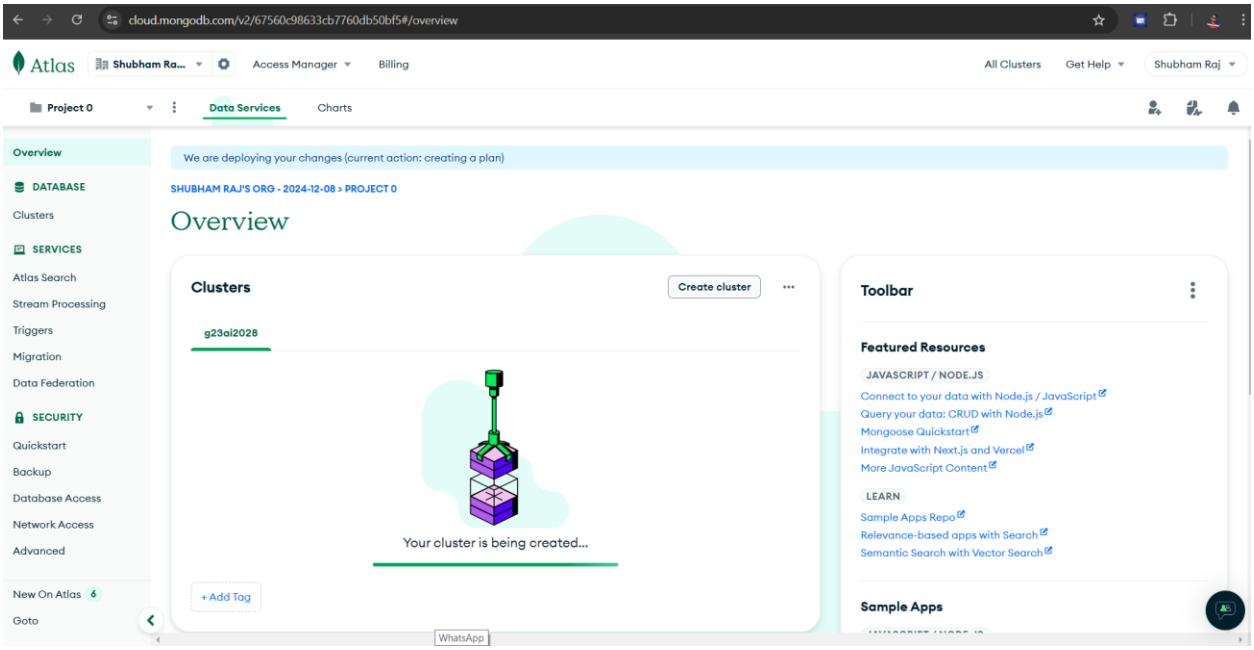
BIG DATA MANAGEMENT

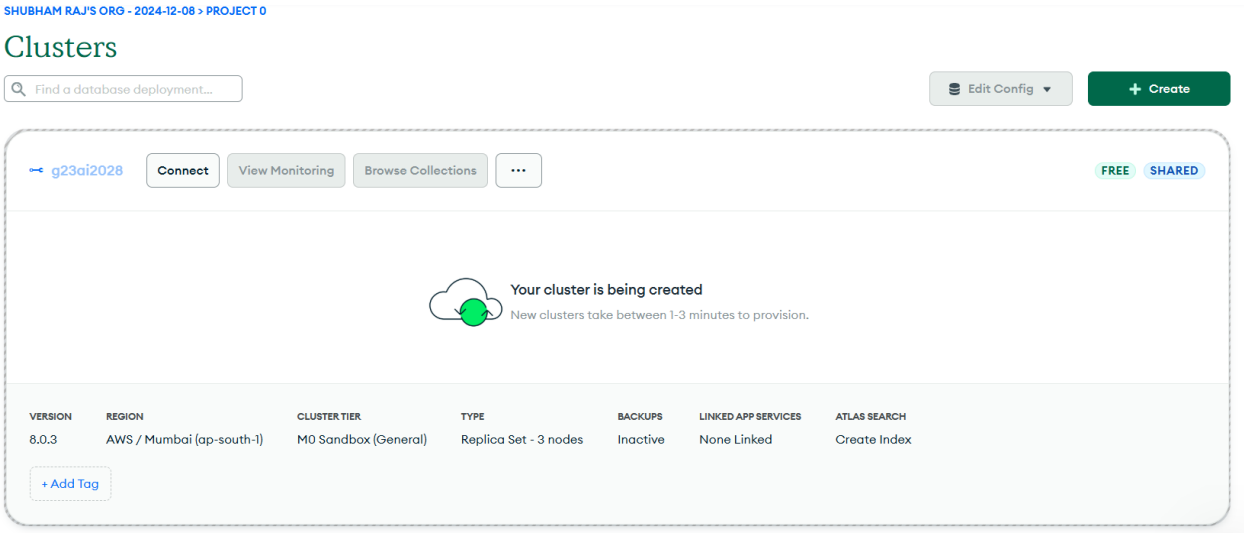
Assignment - 7

Name : Shubham Raj
Roll No : G23AI2028

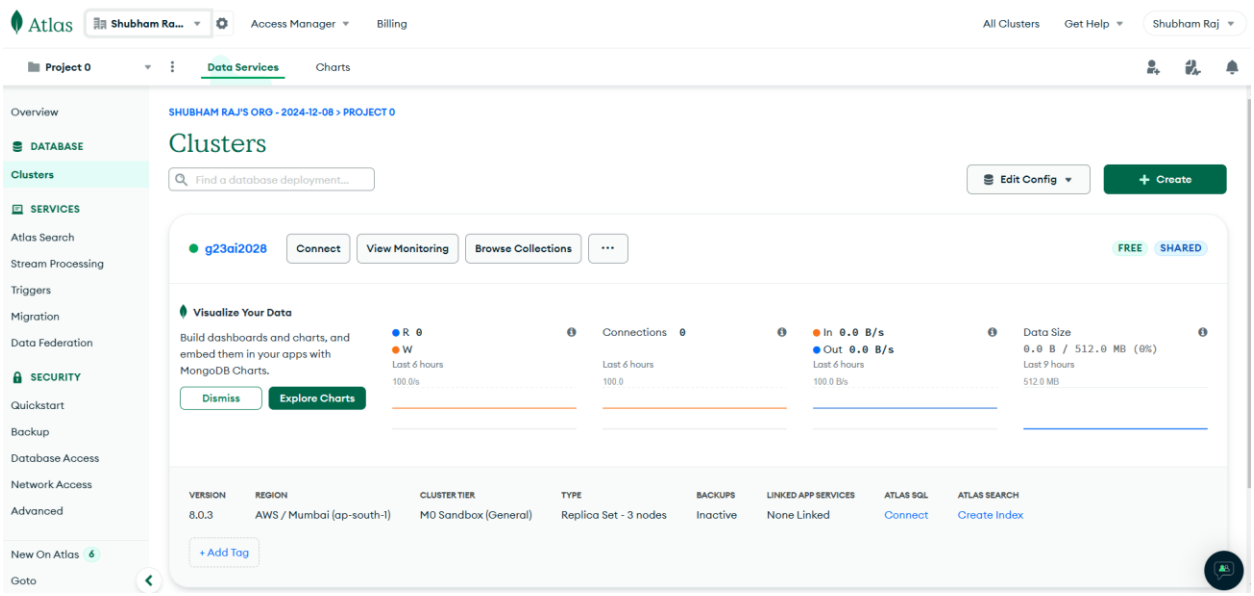
Github

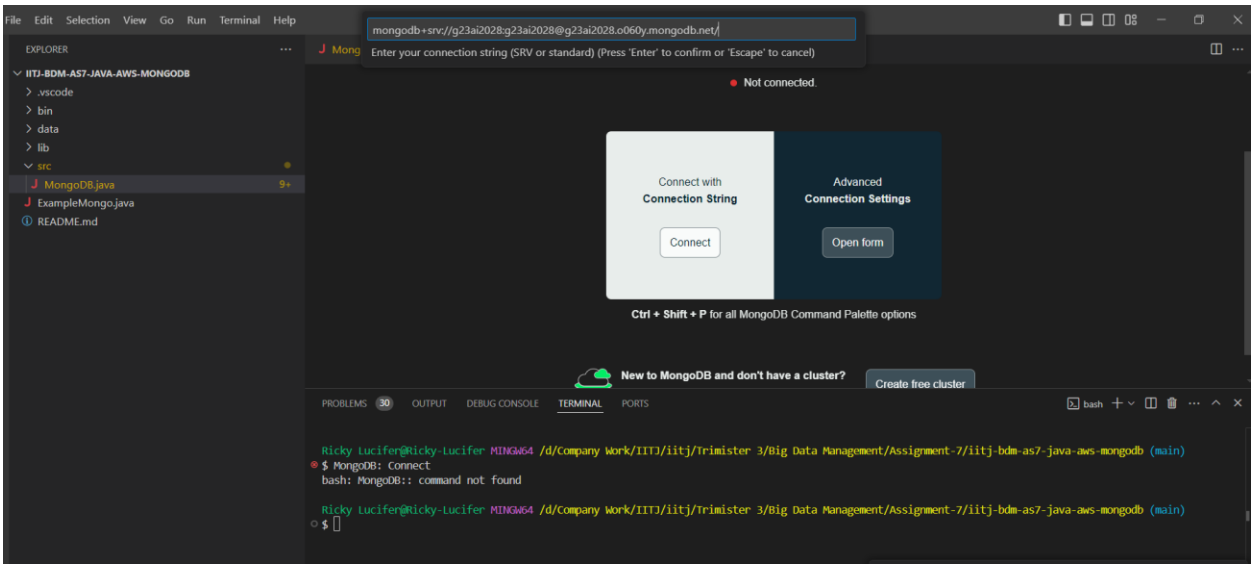
https://github.com/shubham14p3/iitj-bdm-as7-java-aws-mongodb

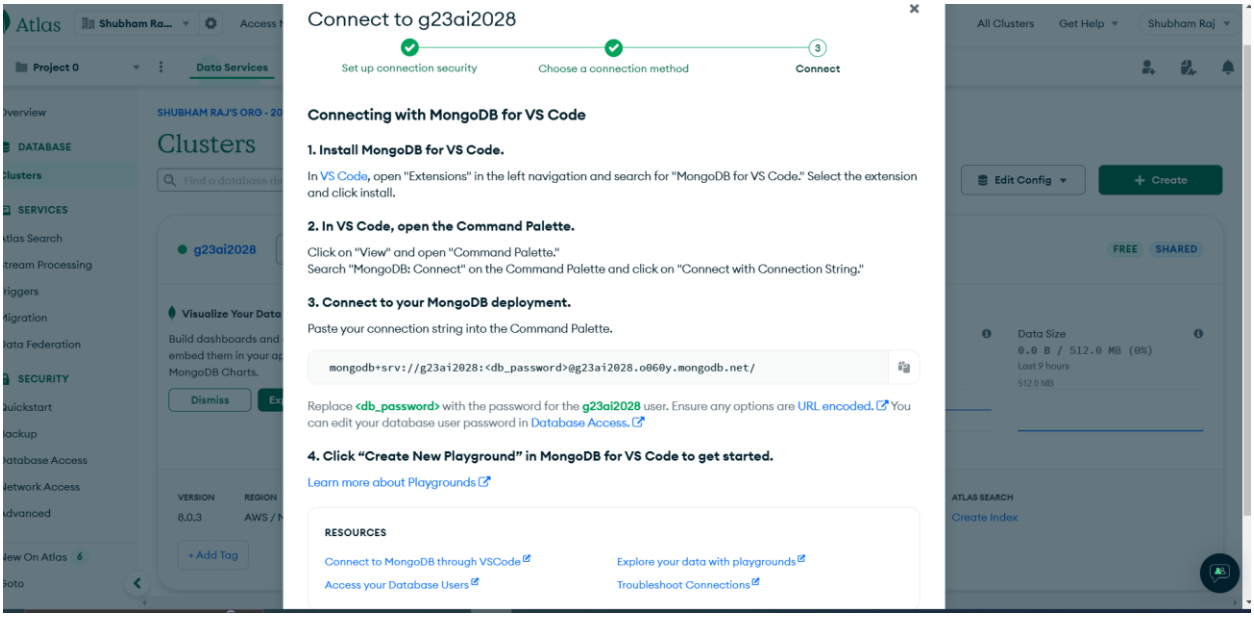




Once the cluster is up







For Connection

```
public MongoDatabase connect() {
        try {
            // Provide connection information to MongoDB server
            String url =
"mongodb+srv://g23ai2028:g23ai2028@g23ai2028.o060y.mongodb.net/?retryWrites=true&
w=majority";
            mongoClient = MongoClients.create(url);
            System.out.println("Connection to MongoDB established
successfully.");
        } catch (Exception ex) {
            System.out.println("Error: Unable to establish connection to
MongoDB.");
            System.out.println("Exception: " + ex);
            ex.printStackTrace();
        }
        // Provide database information to connect to
        // Note: If the database does not already exist, it will be created
        // automatically.
        db = mongoClient.getDatabase(DATABASE_NAME);
        return db;
    }
```

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.

```
/**
```

```
     * Loads TPC-H data into MongoDB.
     *
     * @throws Exception if a file I/O or database error occurs
     */
    public void load() throws Exception {
        // Paths to your data files
        String customerFilePath = "data/customer.tbl";
        String orderFilePath = "data/order.tbl";

        // Load customers data into MongoDB
        System.out.println("Loading customers...");
        List<Document> customers = loadFileToDocuments(customerFilePath, "|",
"customer");
        MongoCollection<Document> customerCollection =
db.getCollection("customer");
        customerCollection.insertMany(customers);
        System.out.println("Customers loaded successfully!");

        // Load orders data into MongoDB
        System.out.println("Loading orders...");
        List<Document> orders = loadFileToDocuments(orderFilePath, "|",
"orders");
        MongoCollection<Document> orderCollection = db.getCollection("orders");
        orderCollection.insertMany(orders);
        System.out.println("Orders loaded successfully!");
    }
```

Output

```
Ricky Lucifer@Ricky-Lucifer MINGW64 /d/Company Work/IITJ/iitj/Trimister 3/Big Data Management/Assignment-7/iitj-bdm-as7-java-aws-mongod
b (main)
$  cd d:\\Company\ Work\\IITJ\\iitj\\Trimister\ 3\\Big\ Data\ Management\\Assignment-7\\iitj-bdm-as7-java-aws-mongodb ; /usr/bin/env C:
\\Program\ Files\\Java\\jdk-23\\bin\\java.exe @C:\\Users\\RICKYL~1\\AppData\\Local\\Temp\\cp_bv2qe2xi6t29cckvpcvi8ttny.argfile MongoDB
\Local\\\\Temp\\\\cp_bv2qe2xi6t29cckvpcvi8ttny.argfile MongoDB ;5e5fd378-d8eb-4799-b18f-065ad6e0ab9e[main] INFO org.mongodb.driver.clus
[main] INFO org.mongodb.driver.cluster - Cluster created with settings {hosts=[127.0.0.1:27017], srvHost=g23ai2028.o060y.mongodb.net, m
ode=MULTIPLE, requiredClusterType=REPLICA_SET, serverSelectionTimeout='30000 ms', requiredReplicaSetName='atlas-ovdpb7-shard-0'}
Connection to MongoDB established successfully.
Loading customers...
[cluster-ClusterId{value='67569dc71af6aa2688f2382c', description='null'}-srv-g23ai2028.o060y.mongodb.net] INFO org.mongodb.driver.clust
er - Adding discovered server g23ai2028-shard-00-01.o060y.mongodb.net:27017 to client view of cluster
[main] INFO org.mongodb.driver.cluster - Cluster description not yet available. Waiting for 30000 ms before timing out
[cluster-ClusterId{value='67569dc71af6aa2688f2382c', description='null'}-srv-g23ai2028.o060y.mongodb.net] INFO org.mongodb.driver.clust
er - Adding discovered server g23ai2028-shard-00-02.o060y.mongodb.net:27017 to client view of cluster
[cluster-ClusterId{value='67569dc71af6aa2688f2382c', description='null'}-srv-g23ai2028.o060y.mongodb.net] INFO org.mongodb.driver.clust
er - Adding discovered server g23ai2028-shard-00-00.o060y.mongodb.net:27017 to client view of cluster
[main] INFO org.mongodb.driver.cluster - No server chosen by com.mongodb.client.internal.MongoClientDelegate$1@21a947fe from cluster de
scription ClusterDescription{type=REPLICA_SET, connectionMode=MULTIPLE, serverDescriptions=[ServerDescription{address=g23ai2028-shard-0
0-01.o060y.mongodb.net:27017, type=UNKNOWN, state=CONNECTING}, ServerDescription{address=g23ai2028-shard-00-02.o060y.mongodb.net:27017,
 type=UNKNOWN, state=CONNECTING}, ServerDescription{address=g23ai2028-shard-00-00.o060y.mongodb.net:27017, type=UNKNOWN, state=CONNECTI
NG}]}. Waiting for 30000 ms before timing out
[cluster-ClusterId{value='67569dc71af6aa2688f2382c', description='null'}-g23ai2028-shard-00-01.o060y.mongodb.net:27017] INFO org.mongod
b.driver.connection - Opened connection [connectionId{localValue:1, serverValue:83891}] to g23ai2028-shard-00-01.o060y.mongodb.net:2701
7
```

```
g{name='provider', value='AWS'}, Tag{name='region', value='AP_SOUTH_1'}, Tag{name='workloadType', value='OPERATIONAL'}]}, electionId=nu
ll, setVersion=290, topologyVersion=TopologyVersion{processId=67533a3da6d393519e23f24b, counter=4}, lastWriteDate=Mon Dec 09 13:05:38 I
ST 2024, lastUpdateTimeNanos=1374688395314900}
[cluster-ClusterId{value='67569dc71af6aa2688f2382c', description='null'}-g23ai2028-shard-00-02.o060y.mongodb.net:27017] INFO org.mongod
b.driver.cluster - Setting max election id to 7fffffff00000000000002de from replica set primary g23ai2028-shard-00-02.o060y.mongodb.net
:27017
[cluster-ClusterId{value='67569dc71af6aa2688f2382c', description='null'}-g23ai2028-shard-00-02.o060y.mongodb.net:27017] INFO org.mongod
b.driver.cluster - Setting max set version to 290 from replica set primary g23ai2028-shard-00-02.o060y.mongodb.net:27017
[cluster-ClusterId{value='67569dc71af6aa2688f2382c', description='null'}-g23ai2028-shard-00-02.o060y.mongodb.net:27017] INFO org.mongod
b.driver.cluster - Discovered replica set primary g23ai2028-shard-00-02.o060y.mongodb.net:27017
[main] INFO org.mongodb.driver.connection - Opened connection [connectionId{localValue:7, serverValue:88356}] to g23ai2028-shard-00-02.
o060y.mongodb.net:27017
Customers loaded successfully!
Loading orders...
Orders loaded successfully!
Loading customers...
Loading orders...
```

2. 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.

```java
    /**
     * Loads customer and orders TPC-H data into a single collection.
     *
     * @throws Exception if a file I/O or database error occurs
     */
    public void loadNest() throws Exception {
        // Paths to your data files
        String customerFilePath = "data/customer.tbl";
        String orderFilePath = "data/order.tbl";

        // Load customers and organize them into a map for nesting
        System.out.println("Loading customers...");
        List<Document> customers = loadFileToDocuments(customerFilePath, "|",
"customer");

        // Load orders and group them by customer key
        System.out.println("Loading orders...");
        List<Document> orders = loadFileToDocuments(orderFilePath, "|",
"orders");

        // Create a mapping of custkey to orders
        Map<Integer, List<Document>> ordersByCustomer = new HashMap<>();
        for (Document order : orders) {
            int custkey = order.getInteger("custkey");
            ordersByCustomer.putIfAbsent(custkey, new ArrayList<>());
            ordersByCustomer.get(custkey).add(order);
        }

        // Combine customers and their orders into a single nested document
        System.out.println("Combining customers and orders into nested
documents...");
        List<Document> custorders = new ArrayList<>();
        for (Document customer : customers) {
```

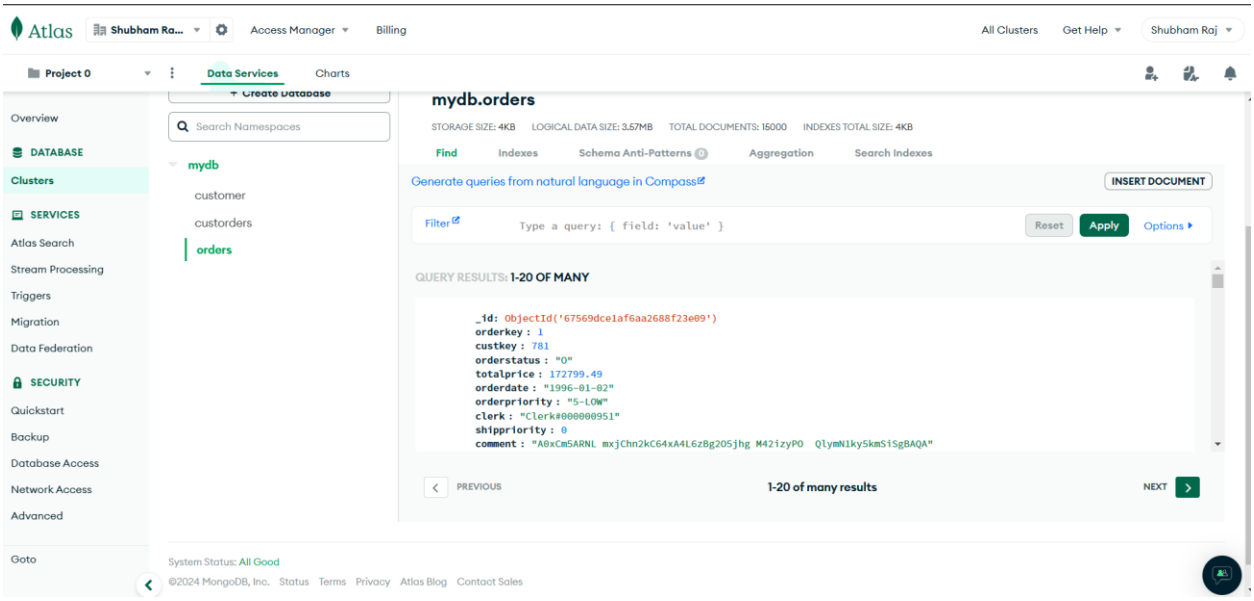
```
            int custkey = customer.getInteger("custkey");
            List<Document> customerOrders =
ordersByCustomer.getOrDefault(custkey, new ArrayList<>());
            customer.put("orders", customerOrders); // Nest the orders into the
customer document
            custorders.add(customer);
        }

        // Insert into the 'custorders' collection
        MongoCollection<Document> custordersCollection =
db.getCollection("custorders");
        custordersCollection.insertMany(custorders);
        System.out.println("Nested customers and orders loaded successfully!");
    }
```

Output

```
g{name='provider', value='AWS'}, Tag{name='region', value='AP_SOUTH_1'}, Tag{name='workloadType', value='OPERATIONAL'}]}, electionId=nu
ll, setVersion=290, topologyVersion=TopologyVersion{processId=67533a3da6d393519e23f24b, counter=4}, lastWriteDate=Mon Dec 09 13:05:38 I
ST 2024, lastUpdateTimeNanos=1374688395314900}
[cluster-ClusterId{value='67569dc71af6aa2688f2382c', description='null'}-g23ai2028-shard-00-02.o060y.mongodb.net:27017] INFO org.mongod
b.driver.cluster - Setting max election id to 7fffffff00000000000002de from replica set primary g23ai2028-shard-00-02.o060y.mongodb.net
:27017
[cluster-ClusterId{value='67569dc71af6aa2688f2382c', description='null'}-g23ai2028-shard-00-02.o060y.mongodb.net:27017] INFO org.mongod
b.driver.cluster - Setting max set version to 290 from replica set primary g23ai2028-shard-00-02.o060y.mongodb.net:27017
[cluster-ClusterId{value='67569dc71af6aa2688f2382c', description='null'}-g23ai2028-shard-00-02.o060y.mongodb.net:27017] INFO org.mongod
b.driver.cluster - Discovered replica set primary g23ai2028-shard-00-02.o060y.mongodb.net:27017
[main] INFO org.mongodb.driver.connection - Opened connection [connectionId{localValue:7, serverValue:88356}] to g23ai2028-shard-00-02.
o060y.mongodb.net:27017
Customers loaded successfully!
Loading orders...
Orders loaded successfully!
Loading customers...
Loading orders...
```

```
o060y.mongodb.net:27017
Customers loaded successfully!
Loading orders...
Orders loaded successfully!
Loading customers...
Loading orders...
Combining customers and orders into nested documents...
Nested customers and orders loaded successfully!
```

In mongodb

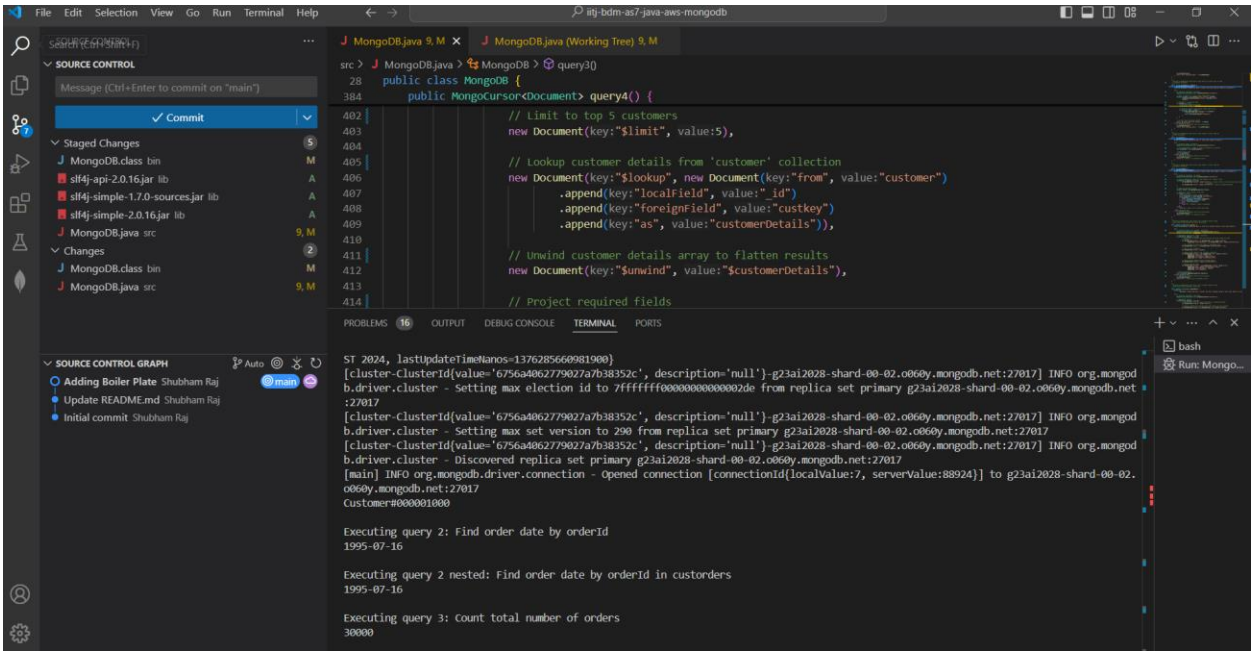


3. Write the method query1() that returns the customer name given a customer id using the customer collection.

```
  /**
     * Performs a MongoDB query that returns customer name given a customer id
     * using customer collection.
     *
     * @param custkey customer id (custkey) to search for.
     * @return name of customer or a message indicating no customer was
     *         found.
     */
    public String query1(int custkey) {
        System.out.println("\nExecuting query 1: Find customer name by custkey");
        // TODO: Writing query
        try {
            // Accessing 'customer' collection
            MongoCollection<Document> col = db.getCollection("customer");
            // See: https://docs.mongodb.com/drivers/java/sync/current/usage-
examples/find/

            // Query collection for given custkey
            Document customer = col.find(eq("custkey",
custkey)).projection(fields(include("name"), exclude("_id")))
                    .first();

            // Checking if a result was found
            if (customer != null) {
                // Returning customer name
                return customer.getString("name");
            } else {
                // No customer found with given custkey
```

```
                return "No customer found with custkey: " + custkey;
            }
        } catch (Exception ex) {
            // Handling any exceptions that occur during  query
            ex.printStackTrace();
            return "Error executing query: " + ex.getMessage();
        }
    }
```

Ouput

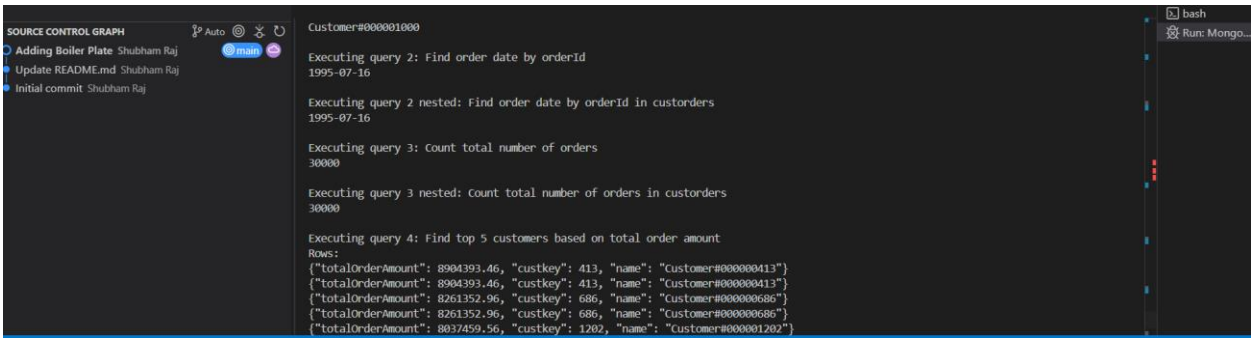


4. Write the method query2() that returns the order date for a given order id using the orders collection.

```
  /**
     * Performs a MongoDB query that returns order date for a given order id
using
     * orders collection.
     */
    public String query2(int orderId) {
        // TODO: Write a MongoDB query
        System.out.println("\nExecuting query 2: Find order date by orderId");

        try {
            // Accessing 'orders' collection
            MongoCollection<Document> col = db.getCollection("orders");

            // Query collection for given orderId
```

```
            Document order = col.find(eq("orderkey",
orderId)).projection(fields(include("orderdate"), exclude("_id")))
                    .first();

            // Checking if a result was found
            if (order != null) {
                // Return order date
                return order.getString("orderdate");
            } else {
                // No order found with given orderId
                return "No order found with orderId: " + orderId;
            }
        } catch (Exception ex) {
            // Handling any exceptions that occur during query
            ex.printStackTrace();
            return "Error executing query: " + ex.getMessage();
        }
    }
```

Output

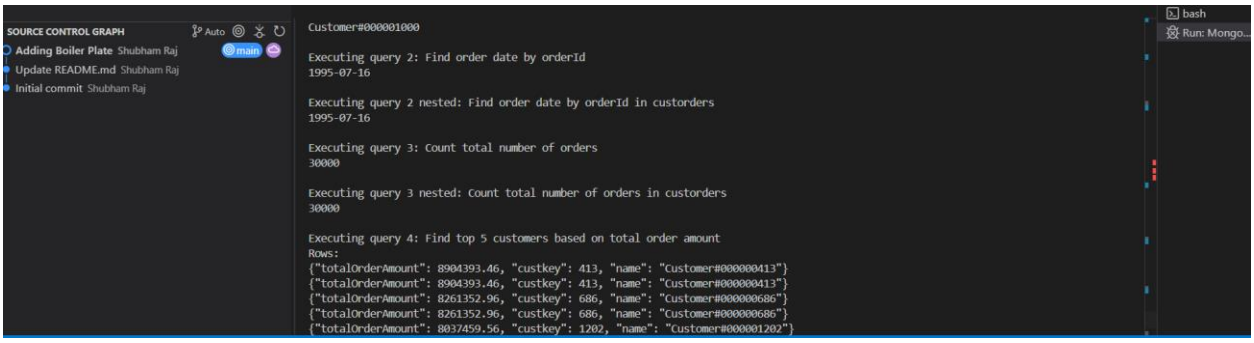


5. Write the method query2Nest() that returns order date for a given order id using the custorders collection.

```
 /**
     * Performs a MongoDB query that returns order date for a given order id
using
     * custorders collection.
     */
    public String query2Nest(int orderId) {
        // TODO: Write a MongoDB query
        System.out.println("\nExecuting query 2 nested: Find order date by
orderId in custorders");

        try {
            // Accessing 'custorders' collection
```

```
            MongoCollection<Document> col = db.getCollection("custorders");

            // Query to search for an order within nested orders array
            Document customer = col.find(eq("orders.orderkey", orderId))
                    .projection(fields(include("orders"), exclude("_id")))
                    .first();

            // Checking if a result was found
            if (customer != null) {
                // Get orders array from document
                List<Document> orders = (List<Document>) customer.get("orders");

                // Search for specific order within nested orders array
                for (Document order : orders) {
                    if (order.getInteger("orderkey") == orderId) {
                        // Return order date if found
                        return order.getString("orderdate");
                    }
                }
            }

            // No order found with given orderId
            return "No order found with orderId: " + orderId;
        } catch (Exception ex) {
            // Handling any exceptions that occur during query
            ex.printStackTrace();
            return "Error executing query: " + ex.getMessage();
        }
    }
```

output



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

```java
    /**
     * Performs a MongoDB query that returns total number of orders using the
     * orders collection.
     */
    public long query3() {
        // TODO: Write a MongoDB query
        System.out.println("\nExecuting query 3: Count total number of orders");

        try {
            // Accessing 'orders' collection
            MongoCollection<Document> col = db.getCollection("orders");

            // Use countDocuments() method to count all documents in collection
            long totalOrders = col.countDocuments();

            // Return total count
            return totalOrders;
        } catch (Exception ex) {
            // Handling any exceptions that occur during query
            ex.printStackTrace();
            return -1; // Return -1 to indicate an error
        }
    }
```

Output

```
Executing query 3: Count total number of orders
30000
```

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

```java
    /**
     * Performs a MongoDB query that returns total number of orders using the
     * custorders collection.
     */
    public long query3Nest() {
        // TODO: Write a MongoDB query
        System.out.println("\nExecuting query 3 nested: Count total number of
orders in custorders");

        try {
            // Accessing 'custorders' collection
            MongoCollection<Document> col = db.getCollection("custorders");
```

```
            // Using an aggregation pipeline to sum lengths of all 'orders'
arrays
            List<Document> pipeline = List.of(
                    new Document("$unwind", "$orders"), // Unwind 'orders' array
to process each order as a separate
                                                        // document
                    new Document("$count", "totalOrders") // Count all unwound
documents
            );

            // Executing aggregation query
            Document result = col.aggregate(pipeline).first();

            // Checking if result contains total count
            if (result != null) {
                // Handling result as either Integer or Long
                Object totalOrders = result.get("totalOrders");
                if (totalOrders instanceof Integer) {
                    return ((Integer) totalOrders).longValue();
                } else if (totalOrders instanceof Long) {
                    return (Long) totalOrders;
                } else {
                    return 0; // Fallback if type is unexpected
                }
            } else {
                return 0; // Return 0 if no orders were found
            }
        } catch (Exception ex) {
            // Handling any exceptions that occur during query
            ex.printStackTrace();
            return -1; // Return -1 to indicate an error
        }
    }
```

Output

```
Executing query 3 nested: Count total number of orders in custorders
30000
```

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

```
 /**
     * Performs a MongoDB query that returns top 5 customers based on total
     * order amount using customer and orders collections.
```

```
     */
    public MongoCursor<Document> query4() {
        System.out.println("\nExecuting query 4: Find top 5 customers based on
total order amount");

        try {
            // Accessing 'orders' and 'customer' collections
            MongoCollection<Document> ordersCol = db.getCollection("orders");
            MongoCollection<Document> customerCol = db.getCollection("customer");

            // Aggregation pipeline for 'orders' collection
            List<Document> pipeline = List.of(
                    // Group orders by customer key and calculate total order
amount for each
                    // customer
                    new Document("$group", new Document("_id", "$custkey")
                            .append("totalOrderAmount", new Document("$sum",
"$totalprice"))),

                    // Sort customers by total order amount in descending order
                    new Document("$sort", new Document("totalOrderAmount", -1)),

                    // Limit to top 5 customers
                    new Document("$limit", 5),

                    // Lookup customer details from 'customer' collection
                    new Document("$lookup", new Document("from", "customer")
                            .append("localField", "_id")
                            .append("foreignField", "custkey")
                            .append("as", "customerDetails")),

                    // Unwind customer details array to flatten results
                    new Document("$unwind", "$customerDetails"),

                    // ProjectING required fields
                    new Document("$project", new Document("custkey", "$_id")
                            .append("name", "$customerDetails.name")
                            .append("totalOrderAmount", 1)
                            .append("_id", 0)));

            // Executing aggregation
            return ordersCol.aggregate(pipeline).iterator();
        } catch (Exception ex) {
            // Handling any exceptions that occur during query
            ex.printStackTrace();
            return null; // Return null to indicate an error
        }
```

```
    }
```

Output

```
Executing query 4: Find top 5 customers based on total order amount
Rows:
{"totalOrderAmount": 8904393.46, "custkey": 413, "name": "Customer#000000413"}
{"totalOrderAmount": 8904393.46, "custkey": 413, "name": "Customer#000000413"}
{"totalOrderAmount": 8261352.96, "custkey": 686, "name": "Customer#000000686"}
{"totalOrderAmount": 8261352.96, "custkey": 686, "name": "Customer#000000686"}
{"totalOrderAmount": 8037459.56, "custkey": 1202, "name": "Customer#000001202"}
{"totalOrderAmount": 8037459.56, "custkey": 1202, "name": "Customer#000001202"}
{"totalOrderAmount": 7972306.16, "custkey": 464, "name": "Customer#000000464"}
{"totalOrderAmount": 7972306.16, "custkey": 464, "name": "Customer#000000464"}
{"totalOrderAmount": 7961968.68, "custkey": 98, "name": "Customer#000000098"}
{"totalOrderAmount": 7961968.68, "custkey": 98, "name": "Customer#000000098"}
Number of rows: 10
```

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

```
 /**
     * Performs a MongoDB query that returns top 5 customers based on total
     * order amount using custorders collection.
     */
    public MongoCursor<Document> query4Nest() {
        System.out
                .println("\nExecuting query 4 nested: Find top 5 customers based
on total order amount in custorders");

        try {
            // Accessing 'custorders' collection
            MongoCollection<Document> col = db.getCollection("custorders");

            // Aggregation pipeline
            List<Document> pipeline = List.of(
                    // Unwind orders array to process each order as a separate
document
                    new Document("$unwind", "$orders"),

                    // Group by customer and calculate total order amount for
each customer
                    new Document("$group", new Document("_id", "$custkey")
                            .append("name", new Document("$first", "$name")) //
Include customer name
                            .append("totalOrderAmount", new Document("$sum",
"$orders.totalprice"))),
```

```
                // Sort by total order amount in descending order
                new Document("$sort", new Document("totalOrderAmount", -1)),

                // Limit to top 5 customers
                new Document("$limit", 5),

                // ProjectING required fields
                new Document("$project", new Document("custkey", "$_id")
                        .append("name", 1)
                        .append("totalOrderAmount", 1)
                        .append("_id", 0)));

            // Executing aggregation
            return col.aggregate(pipeline).iterator();
        } catch (Exception ex) {
            // Handling any exceptions that occur during query
            ex.printStackTrace();
            return null; // Return null to indicate an error
        }
    }
```

Output

```
Executing query 4 nested: Find top 5 customers based on total order amount in custorders
Rows:
{"name": "Customer#000000413", "totalOrderAmount": 8904393.46, "custkey": 413}
{"name": "Customer#000000686", "totalOrderAmount": 8261352.96, "custkey": 686}
{"name": "Customer#000001202", "totalOrderAmount": 8037459.56, "custkey": 1202}
{"name": "Customer#000000464", "totalOrderAmount": 7972306.16, "custkey": 464}
{"name": "Customer#000000098", "totalOrderAmount": 7961968.68, "custkey": 98}
Number of rows: 5
```

All Codes:

```
import static com.mongodb.client.model.Filters.*;
import static com.mongodb.client.model.Projections.*;
import java.io.BufferedReader;
import java.io.File;
import java.io.FileReader;
import java.math.BigDecimal;
import java.util.ArrayList;
import java.util.Arrays;
import java.util.HashMap;
import java.util.List;
```

```
import java.util.Map;

import org.bson.Document;
import org.bson.conversions.Bson;
import com.mongodb.BasicDBList;
import com.mongodb.BasicDBObject;
import com.mongodb.client.AggregateIterable;
import com.mongodb.client.MongoClient;
import com.mongodb.client.MongoClients;
import com.mongodb.client.MongoCollection;
import com.mongodb.client.MongoCursor;
import com.mongodb.client.MongoDatabase;

/**
 * Program to create a collection, insert JSON objects, and perform simple
 * queries on MongoDB.
 */
public class MongoDB {

    /**
     * MongoDB database name
     */
    public static final String DATABASE_NAME = "mydb";

    /**
     * MongoDB collection name
     */
    public static final String COLLECTION_NAME = "data";

    /**
     * Mongo client connection to server
     */
    public MongoClient mongoClient;

    /**
     * Mongo database
     */
    public MongoDatabase db;

    /**
     * Main method
     *
     * @param args no arguments required
     */
    public static void main(String[] args) throws Exception {
        MongoDB qmongo = new MongoDB();
        qmongo.connect();
```

```
        // qmongo.load();
        // qmongo.loadNest();
        System.out.println(qmongo.query1(1000));
        System.out.println(qmongo.query2(32));
        System.out.println(qmongo.query2Nest(32));
        System.out.println(qmongo.query3());
        System.out.println(qmongo.query3Nest());
        System.out.println(MongoDB.toString(qmongo.query4()));
        System.out.println(MongoDB.toString(qmongo.query4Nest()));
    }

    /**
     * Connects to Mongo database and returns database object to manipulate for
     * connection.
     *
     * @return Mongo database
     */
    public MongoDatabase connect() {
        try {
            // Provide connection information to MongoDB server
            String url =
"mongodb+srv://g23ai2028:g23ai2028@g23ai2028.o060y.mongodb.net/?retryWrites=true&
w=majority";
            mongoClient = MongoClients.create(url);
            System.out.println("Connection to MongoDB established
successfully.");
        } catch (Exception ex) {
            System.out.println("Error: Unable to establish connection to
MongoDB.");
            System.out.println("Exception: " + ex);
            ex.printStackTrace();
        }
        // Provide database information to connect to
        // Note: If database does not already exist, it will be created
        // automatically.
        db = mongoClient.getDatabase(DATABASE_NAME);
        return db;
    }

    /**
     * Loads TPC-H data into MongoDB.
     *
     * @throws Exception if a file I/O or database error occurs
     */
    public void load() throws Exception {
        // Locaion Paths to my data files
        String customerFilePath = "data/customer.tbl";
```

```
        String orderFilePath = "data/order.tbl";

        // Loading customers data into MongoDB as given
        System.out.println("Loading customers...");
        List<Document> customers = loadFileToDocuments(customerFilePath, "|",
"customer");
        MongoCollection<Document> customerCollection =
db.getCollection("customer");
        customerCollection.insertMany(customers);
        System.out.println("Customers loaded successfully!");

        // Loading orders data into MongoDB as per asked
        System.out.println("Loading orders...");
        List<Document> orders = loadFileToDocuments(orderFilePath, "|",
"orders");
        MongoCollection<Document> orderCollection = db.getCollection("orders");
        orderCollection.insertMany(orders);
        System.out.println("Orders loaded successfully!");
    }

    /**
     * Loads customer and orders TPC-H data into a single collection.
     *
     * @throws Exception if a file I/O or database error occurs
     */
    public void loadNest() throws Exception {
        // TODO: Load customer and orders data into single collection called
custorders
        // TODO: Consider using insertMany() for bulk insert for faster
performance

        // Paths to your data files in my local
        String customerFilePath = "data/customer.tbl";
        String orderFilePath = "data/order.tbl";

        // Loading customers and organize them into a map for nesting
        System.out.println("Loading customers...");
        List<Document> customers = loadFileToDocuments(customerFilePath, "|",
"customer");

        // Loading orders and group them by customer key
        System.out.println("Loading orders...");
        List<Document> orders = loadFileToDocuments(orderFilePath, "|",
"orders");

        // Creating  mapping of custkey to orders
        Map<Integer, List<Document>> ordersByCustomer = new HashMap<>();
```

```
        for (Document order : orders) {
            int custkey = order.getInteger("custkey");
            ordersByCustomer.putIfAbsent(custkey, new ArrayList<>());
            ordersByCustomer.get(custkey).add(order);
        }

        // Combining customers and their orders into a single nested document
        System.out.println("Combining customers and orders into nested
documents...");
        List<Document> custorders = new ArrayList<>();
        for (Document customer : customers) {
            int custkey = customer.getInteger("custkey");
            List<Document> customerOrders =
ordersByCustomer.getOrDefault(custkey, new ArrayList<>());
            customer.put("orders", customerOrders); // Nest orders into customer
document for better qyery
            custorders.add(customer);
        }

        // Inserting into 'custorders' collection
        MongoCollection<Document> custordersCollection =
db.getCollection("custorders");
        custordersCollection.insertMany(custorders);
        System.out.println("Nested customers and orders loaded successfully!");
    }

    /**
     * Helper method to parse a .tbl file into a list of MongoDB documents.
     *
     * @param filePath  Path to .tbl file
     * @param delimiter Delimiter used in file
     * @param type      Type of data (e.g., customer or orders) for field mapping
     * @return List of MongoDB documents
     * @throws Exception If a file I/O error occurs
     */
    private List<Document> loadFileToDocuments(String filePath, String delimiter,
String type) throws Exception {
        List<Document> documents = new ArrayList<>();
        try (BufferedReader reader = new BufferedReader(new
FileReader(filePath))) {
            String line;
            while ((line = reader.readLine()) != null) {
                String[] fields = line.split("\\" + delimiter);

                Document document = new Document();
                if (type.equals("customer")) {
```

```
                    document.put("custkey",
Integer.parseInt(fields[0].trim()));
                    document.put("name", fields[1].trim());
                    document.put("address", fields[2].trim());
                    document.put("nationkey",
Integer.parseInt(fields[3].trim()));
                    document.put("phone", fields[4].trim());
                    document.put("acctbal",
Double.parseDouble(fields[5].trim()));
                    document.put("mktsegment", fields[6].trim());
                    document.put("comment", fields[7].trim());
                } else if (type.equals("orders")) {
                     document.put("orderkey",
Integer.parseInt(fields[0].trim()));
                    document.put("custkey", Integer.parseInt(fields[1].trim()));
                    document.put("orderstatus", fields[2].trim());
                    document.put("totalprice",
Double.parseDouble(fields[3].trim()));
                    document.put("orderdate", fields[4].trim());
                    document.put("orderpriority", fields[5].trim());
                    document.put("clerk", fields[6].trim());
                    document.put("shippriority",
Integer.parseInt(fields[7].trim()));
                    document.put("comment", fields[8].trim());
                }

                documents.add(document);
            }
        }
        return documents;
    }

    /**
     * Performs a MongoDB query that returns customer name given a customer id
     * using customer collection.
     *
     * @param custkey customer id (custkey) to search for.
     * @return name of customer or a message indicating no customer was
     *         found.
     */
    public String query1(int custkey) {
        System.out.println("\nExecuting query 1: Find customer name by custkey");
        // TODO: Writing query
        try {
            // Accessing 'customer' collection
            MongoCollection<Document> col = db.getCollection("customer");
```

```
            // See: https://docs.mongodb.com/drivers/java/sync/current/usage-examples/find/

            // Query collection for given custkey
            Document customer = col.find(eq("custkey", custkey)).projection(fields(include("name"), exclude("_id")))
                    .first();

            // Checking if a result was found
            if (customer != null) {
                // Returning customer name
                return customer.getString("name");
            } else {
                // No customer found with given custkey
                return "No customer found with custkey: " + custkey;
            }
        } catch (Exception ex) {
            // Handling any exceptions that occur during  query
            ex.printStackTrace();
            return "Error executing query: " + ex.getMessage();
        }
    }

    /**
     * Performs a MongoDB query that returns order date for a given order id using
     * orders collection.
     */
    public String query2(int orderId) {
        // TODO: Write a MongoDB query
        System.out.println("\nExecuting query 2: Find order date by orderId");

        try {
            // Accessing 'orders' collection
            MongoCollection<Document> col = db.getCollection("orders");

            // Query collection for given orderId
            Document order = col.find(eq("orderkey", orderId)).projection(fields(include("orderdate"), exclude("_id")))
                    .first();

            // Checking if a result was found
            if (order != null) {
                // Return order date
                return order.getString("orderdate");
            } else {
                // No order found with given orderId
```

```
                return "No order found with orderId: " + orderId;
            }
        } catch (Exception ex) {
            // Handling any exceptions that occur during query
            ex.printStackTrace();
            return "Error executing query: " + ex.getMessage();
        }
    }

    /**
     * Performs a MongoDB query that returns order date for a given order id
using
     * custorders collection.
     */
    public String query2Nest(int orderId) {
        // TODO: Write a MongoDB query
        System.out.println("\nExecuting query 2 nested: Find order date by
orderId in custorders");

        try {
            // Accessing 'custorders' collection
            MongoCollection<Document> col = db.getCollection("custorders");

            // Query to search for an order within nested orders array
            Document customer = col.find(eq("orders.orderkey", orderId))
                    .projection(fields(include("orders"), exclude("_id")))
                    .first();

            // Checking if a result was found
            if (customer != null) {
                // Get orders array from document
                List<Document> orders = (List<Document>) customer.get("orders");

                // Search for specific order within nested orders array
                for (Document order : orders) {
                    if (order.getInteger("orderkey") == orderId) {
                        // Return order date if found
                        return order.getString("orderdate");
                    }
                }
            }

            // No order found with given orderId
            return "No order found with orderId: " + orderId;
        } catch (Exception ex) {
            // Handling any exceptions that occur during query
            ex.printStackTrace();
```

```
            return "Error executing query: " + ex.getMessage();
        }
    }

    /**
     * Performs a MongoDB query that returns total number of orders using the
     * orders collection.
     */
    public long query3() {
        // TODO: Write a MongoDB query
        System.out.println("\nExecuting query 3: Count total number of orders");

        try {
            // Accessing 'orders' collection
            MongoCollection<Document> col = db.getCollection("orders");

            // Use countDocuments() method to count all documents in collection
            long totalOrders = col.countDocuments();

            // Return total count
            return totalOrders;
        } catch (Exception ex) {
            // Handling any exceptions that occur during query
            ex.printStackTrace();
            return -1; // Return -1 to indicate an error
        }
    }

    /**
     * Performs a MongoDB query that returns total number of orders using the
     * custorders collection.
     */
    public long query3Nest() {
        // TODO: Write a MongoDB query
        System.out.println("\nExecuting query 3 nested: Count total number of
orders in custorders");

        try {
            // Accessing 'custorders' collection
            MongoCollection<Document> col = db.getCollection("custorders");

            // Using an aggregation pipeline to sum lengths of all 'orders'
arrays
            List<Document> pipeline = List.of(
                    new Document("$unwind", "$orders"), // Unwind 'orders' array
to process each order as a separate
                                                        // document
```

```
                    new Document("$count", "totalOrders") // Count all unwound
documents
            );

            // Executing aggregation query
            Document result = col.aggregate(pipeline).first();

            // Checking if result contains total count
            if (result != null) {
                // Handling result as either Integer or Long
                Object totalOrders = result.get("totalOrders");
                if (totalOrders instanceof Integer) {
                    return ((Integer) totalOrders).longValue();
                } else if (totalOrders instanceof Long) {
                    return (Long) totalOrders;
                } else {
                    return 0; // Fallback if type is unexpected
                }
            } else {
                return 0; // Return 0 if no orders were found
            }
        } catch (Exception ex) {
            // Handling any exceptions that occur during query
            ex.printStackTrace();
            return -1; // Return -1 to indicate an error
        }
    }

    /**
     * Performs a MongoDB query that returns top 5 customers based on total
     * order amount using customer and orders collections.
     */
    public MongoCursor<Document> query4() {
        System.out.println("\nExecuting query 4: Find top 5 customers based on
total order amount");

        try {
            // Accessing 'orders' and 'customer' collections
            MongoCollection<Document> ordersCol = db.getCollection("orders");
            MongoCollection<Document> customerCol = db.getCollection("customer");

            // Aggregation pipeline for 'orders' collection
            List<Document> pipeline = List.of(
                    // Group orders by customer key and calculate total order
amount for each
                    // customer
                    new Document("$group", new Document("_id", "$custkey")
```

```
                            .append("totalOrderAmount", new Document("$sum",
"$totalprice"))),

                    // Sort customers by total order amount in descending order
                    new Document("$sort", new Document("totalOrderAmount", -1)),

                    // Limit to top 5 customers
                    new Document("$limit", 5),

                    // Lookup customer details from 'customer' collection
                    new Document("$lookup", new Document("from", "customer")
                            .append("localField", "_id")
                            .append("foreignField", "custkey")
                            .append("as", "customerDetails")),

                    // Unwind customer details array to flatten results
                    new Document("$unwind", "$customerDetails"),

                    // ProjectING required fields
                    new Document("$project", new Document("custkey", "$_id")
                            .append("name", "$customerDetails.name")
                            .append("totalOrderAmount", 1)
                            .append("_id", 0)));

            // Executing aggregation
            return ordersCol.aggregate(pipeline).iterator();
        } catch (Exception ex) {
            // Handling any exceptions that occur during query
            ex.printStackTrace();
            return null; // Return null to indicate an error
        }
    }

    /**
     * Performs a MongoDB query that returns top 5 customers based on total
     * order amount using custorders collection.
     */
    public MongoCursor<Document> query4Nest() {
        System.out
                .println("\nExecuting query 4 nested: Find top 5 customers based
on total order amount in custorders");

        try {
            // Accessing 'custorders' collection
            MongoCollection<Document> col = db.getCollection("custorders");

            // Aggregation pipeline
```

```
            List<Document> pipeline = List.of(
                    // Unwind orders array to process each order as a separate
document
                    new Document("$unwind", "$orders"),

                    // Group by customer and calculate total order amount for
each customer
                    new Document("$group", new Document("_id", "$custkey")
                            .append("name", new Document("$first", "$name")) //
Include customer name
                            .append("totalOrderAmount", new Document("$sum",
"$orders.totalprice"))),

                    // Sort by total order amount in descending order
                    new Document("$sort", new Document("totalOrderAmount", -1)),

                    // Limit to top 5 customers
                    new Document("$limit", 5),

                    // ProjectING required fields
                    new Document("$project", new Document("custkey", "$_id")
                            .append("name", 1)
                            .append("totalOrderAmount", 1)
                            .append("_id", 0)));

            // Executing aggregation
            return col.aggregate(pipeline).iterator();
        } catch (Exception ex) {
            // Handling any exceptions that occur during query
            ex.printStackTrace();
            return null; // Return null to indicate an error
        }
    }

    /**
     * Returns Mongo database being used.
     *
     * @return Mongo database
     */
    public MongoDatabase getDb() {
        return db;
    }

    /**
     * Outputs a cursor of MongoDB results in string form.
     *
     * @param cursor Mongo cursor
```

```
     * @return results as a string
     */
    public static String toString(MongoCursor<Document> cursor) {
        StringBuilder buf = new StringBuilder();
        int count = 0;
        buf.append("Rows:\n");
        if (cursor != null) {
            while (cursor.hasNext()) {
                Document obj = cursor.next();
                buf.append(obj.toJson());
                buf.append("\n");
                count++;
            }
            cursor.close();
        }
        buf.append("Number of rows: " + count);
        return buf.toString();
    }
}
```

All Output

Ricky Lucifer@Ricky-Lucifer MINGW64 /d/Company Work/IITJ/iitj/Trimister 3/Big Data Management/Assignment-7/iitj-bdm-as7-java-aws-mongodb (main)

b (main)

$ cd d:\\Company\ Work\\IITJ\\iitj\\Trimister\ 3\\Big\ Data\ Management\\Assignment-7\\iitj-bdm-as7-java-aws-mongodb ; /usr/bin/env C:\\Program\ Files\\Java\\jdk-23\\bin\\java.exe @C:\\Users\\RICKYL~1\\AppData\\Local\\Temp\\cp_bv2qe2xi6t29cckvpcvi8ttny.argfile MongoDB

\Local\\\\Temp\\\\cp_bv2qe2xi6t29cckvpcvi8ttny.argfile MongoDB ;5113ffdf-0fa0-48d9-abc3-baec8faad67d[main] INFO org.mongodb.driver.cluster - Cluster created with settings {hosts=[127.0.0.1:27017], srvHost=g23ai2028.o060y.mongodb.net, m

ode=MULTIPLE, requiredClusterType=REPLICA_SET, serverSelectionTimeout='30000 ms', requiredReplicaSetName='atlas-ovdpb7-shard-0'}

Connection to MongoDB established successfully.

Executing query 1: Find customer name by custkey

[cluster-ClusterId{value='6756a4062779027a7b38352c', description='null'}-srv-g23ai2028.o060y.mongodb.net] INFO org.mongodb.driver.cluster - Adding discovered server g23ai2028-shard-00-01.o060y.mongodb.net:27017 to client view of cluster

[main] INFO org.mongodb.driver.cluster - Cluster description not yet available. Waiting for 30000 ms before timing out

[cluster-ClusterId{value='6756a4062779027a7b38352c', description='null'}-srv-g23ai2028.o060y.mongodb.net] INFO org.mongodb.driver.cluster - Adding discovered server g23ai2028-shard-00-02.o060y.mongodb.net:27017 to client view of cluster
[cluster-ClusterId{value='6756a4062779027a7b38352c', description='null'}-srv-g23ai2028.o060y.mongodb.net] INFO org.mongodb.driver.cluster - Adding discovered server g23ai2028-shard-00-00.o060y.mongodb.net:27017 to client view of cluster
[main] INFO org.mongodb.driver.cluster - No server chosen by com.mongodb.client.internal.MongoClientDelegate$1@1b68b9a4 from cluster description ClusterDescription{type=REPLICA_SET, connectionMode=MULTIPLE, serverDescriptions=[ServerDescription{address=g23ai2028-shard-00-01.o060y.mongodb.net:27017, type=UNKNOWN, state=CONNECTING}, ServerDescription{address=g23ai2028-shard-00-02.o060y.mongodb.net:27017, type=UNKNOWN, state=CONNECTING}, ServerDescription{address=g23ai2028-shard-00-00.o060y.mongodb.net:27017, type=UNKNOWN, state=CONNECTING}]}. Waiting for 30000 ms before timing out
[cluster-rtt-ClusterId{value='6756a4062779027a7b38352c', description='null'}-g23ai2028-shard-00-02.o060y.mongodb.net:27017] INFO org.mongodb.driver.connection - Opened connection [connectionId{localValue:3, serverValue:89175}] to g23ai2028-shard-00-02.o060y.mongodb.net:27017
[cluster-rtt-ClusterId{value='6756a4062779027a7b38352c', description='null'}-g23ai2028-shard-00-01.o060y.mongodb.net:27017] INFO org.mongodb.driver.connection - Opened connection [connectionId{localValue:5, serverValue:84500}] to g23ai2028-shard-00-01.o060y.mongodb.net:27017
[cluster-rtt-ClusterId{value='6756a4062779027a7b38352c', description='null'}-g23ai2028-shard-00-00.o060y.mongodb.net:27017] INFO org.mongodb.driver.connection - Opened connection [connectionId{localValue:2, serverValue:84971}] to g23ai2028-shard-00-00.o060y.mongodb.net:27017
[cluster-ClusterId{value='6756a4062779027a7b38352c', description='null'}-g23ai2028-shard-00-02.o060y.mongodb.net:27017] INFO org.mongodb.driver.connection - Opened connection [connectionId{localValue:6, serverValue:89092}] to g23ai2028-shard-00-02.o060y.mongodb.net:27017
[cluster-ClusterId{value='6756a4062779027a7b38352c', description='null'}-g23ai2028-shard-00-01.o060y.mongodb.net:27017] INFO org.mongodb.driver.connection - Opened connection [connectionId{localValue:1, serverValue:84500}] to g23ai2028-shard-00-01.o060y.mongodb.net:27017
[cluster-ClusterId{value='6756a4062779027a7b38352c', description='null'}-g23ai2028-shard-00-00.o060y.mongodb.net:27017] INFO org.mongodb.driver.connection - Opened connection [connectionId{localValue:4, serverValue:84971}] to g23ai2028-shard-00-00.o060y.mongodb.net:27017
[cluster-ClusterId{value='6756a4062779027a7b38352c', description='null'}-g23ai2028-shard-00-02.o060y.mongodb.net:27017] INFO org.mongodb.driver.cluster - Monitor thread successfully connected to server with description ServerDescription{address=g23ai2028-shard-

00-02.o060y.mongodb.net:27017, type=REPLICA_SET_PRIMARY, state=CONNECTED, ok=true, minWireVersion=0, maxWireVersion=25, maxDocumentSize=16777216, logicalSessionTimeoutMinutes=30, roundTripTimeNanos=1421718500, setName='atlas-ovdpb7-shard-0', canonicalAddress=g23ai2028-shard-00-02.o060y.mongodb.net:27017, hosts=[g23ai2028-shard-00-01.o060y.mongodb.net:27017, g23ai2028-shard-00-02.o060y.mongodb.net:27017, g23ai2028-shard-00-00.o060y.mongodb.net:27017], passives=[], arbiters=[], primary='g23ai2028-shard-00-02.o060y.mongodb.net:27017', tagSet=TagSet{[Tag{name='availabilityZone', value='aps1-az2'}, Tag{name='diskState', value='READY'}, Tag{name='nodeType', value='ELECTABLE'}, Tag{name='provider', value='AWS'}, Tag{name='region', value='AP_SOUTH_1'}, Tag{name='workloadType', value='OPERATIONAL'}]}, electionId=7fffffff00000000000002de, setVersion=290, topologyVersion=TopologyVersion{processId=67533933c2f4c7026f225521, counter=6}, lastWriteDate=Mon Dec 09 13:32:15 IST 2024, lastUpdateTimeNanos=1376285668614600}
[cluster-ClusterId{value='6756a4062779027a7b38352c', description='null'}-g23ai2028-shard-00-01.o060y.mongodb.net:27017] INFO org.mongodb.driver.cluster - Monitor thread successfully connected to server with description ServerDescription{address=g23ai2028-shard-00-01.o060y.mongodb.net:27017, type=REPLICA_SET_SECONDARY, state=CONNECTED, ok=true, minWireVersion=0, maxWireVersion=25, maxDocumentSize=16777216, logicalSessionTimeoutMinutes=30, roundTripTimeNanos=1414067300, setName='atlas-ovdpb7-shard-0', canonicalAddress=g23ai2028-shard-00-01.o060y.mongodb.net:27017, hosts=[g23ai2028-shard-00-01.o060y.mongodb.net:27017, g23ai2028-shard-00-02.o060y.mongodb.net:27017, g23ai2028-shard-00-00.o060y.mongodb.net:27017], passives=[], arbiters=[], primary='g23ai2028-shard-00-02.o060y.mongodb.net:27017', tagSet=TagSet{[Tag{name='availabilityZone', value='aps1-az3'}, Tag{name='diskState', value='READY'}, Tag{name='nodeType', value='ELECTABLE'}, Tag{name='provider', value='AWS'}, Tag{name='region', value='AP_SOUTH_1'}, Tag{name='workloadType', value='OPERATIONAL'}]}, electionId=null, setVersion=290, topologyVersion=TopologyVersion{processId=67533b2fb3e842d3bf161dac, counter=3}, lastWriteDate=Mon Dec 09 13:32:15 IST 2024, lastUpdateTimeNanos=1376285660970300}
[cluster-ClusterId{value='6756a4062779027a7b38352c', description='null'}-g23ai2028-shard-00-00.o060y.mongodb.net:27017] INFO org.mongodb.driver.cluster - Monitor thread successfully connected to server with description ServerDescription{address=g23ai2028-shard-00-00.o060y.mongodb.net:27017, type=REPLICA_SET_SECONDARY, state=CONNECTED, ok=true, minWireVersion=0, maxWireVersion=25, maxDocumentSize=16777216, logicalSessionTimeoutMinutes=30, roundTripTimeNanos=1413539600, setName='atlas-ovdpb7-shard-0', canonicalAddress=g23ai2028-shard-00-00.o060y.mongodb.net:27017, hosts=[g23ai2028-shard-00-01.o060y.mongodb.net:27017, g23ai2028-shard-00-02.o060y.mongodb.net:27017, g23ai2028-shard-00-00.o060y.mongodb.net:27017], passives=[], arbiters=[], primary='g23ai2028-shard-00-02.o060y.mongodb.net:27017', tagSet=TagSet{[Tag{name='availabilityZone', value='aps1-az1'}, Tag{name='diskState', value='READY'}, Tag{name='nodeType', value='ELECTABLE'}, Tag{name='provider', value='AWS'}, Tag{name='region', value='AP_SOUTH_1'}, Tag{name='workloadType',

value='OPERATIONAL'}]}, electionId=null, setVersion=290,
topologyVersion=TopologyVersion{processId=67533a3da6d393519e23f24b, counter=4},
lastWriteDate=Mon Dec 09 13:32:15 IST 2024, lastUpdateTimeNanos=1376285660981900}
[cluster-ClusterId{value='6756a4062779027a7b38352c', description='null'}-g23ai2028-shard-
00-02.o060y.mongodb.net:27017] INFO org.mongodb.driver.cluster - Setting max election id to
7fffffff00000000000002de from replica set primary g23ai2028-shard-00-
02.o060y.mongodb.net:27017
[cluster-ClusterId{value='6756a4062779027a7b38352c', description='null'}-g23ai2028-shard-
00-02.o060y.mongodb.net:27017] INFO org.mongodb.driver.cluster - Setting max set version to
290 from replica set primary g23ai2028-shard-00-02.o060y.mongodb.net:27017
[cluster-ClusterId{value='6756a4062779027a7b38352c', description='null'}-g23ai2028-shard-
00-02.o060y.mongodb.net:27017] INFO org.mongodb.driver.cluster - Discovered replica set
primary g23ai2028-shard-00-02.o060y.mongodb.net:27017
[main] INFO org.mongodb.driver.connection - Opened connection [connectionId{localValue:7,
serverValue:88924}] to g23ai2028-shard-00-02.o060y.mongodb.net:27017
Customer#000001000

Executing query 2: Find order date by orderId
1995-07-16

Executing query 2 nested: Find order date by orderId in custorders
1995-07-16

Executing query 3: Count total number of orders
30000

Executing query 3 nested: Count total number of orders in custorders
30000

Executing query 4: Find top 5 customers based on total order amount
Rows:
{"totalOrderAmount": 8904393.46, "custkey": 413, "name": "Customer#000000413"}
{"totalOrderAmount": 8904393.46, "custkey": 413, "name": "Customer#000000413"}
{"totalOrderAmount": 8261352.96, "custkey": 686, "name": "Customer#000000686"}
{"totalOrderAmount": 8261352.96, "custkey": 686, "name": "Customer#000000686"}
{"totalOrderAmount": 8037459.56, "custkey": 1202, "name": "Customer#000001202"}
{"totalOrderAmount": 8037459.56, "custkey": 1202, "name": "Customer#000001202"}
{"totalOrderAmount": 7972306.16, "custkey": 464, "name": "Customer#000000464"}
{"totalOrderAmount": 7972306.16, "custkey": 464, "name": "Customer#000000464"}
{"totalOrderAmount": 7961968.68, "custkey": 98, "name": "Customer#000000098"}
{"totalOrderAmount": 7961968.68, "custkey": 98, "name": "Customer#000000098"}
Number of rows: 10

Executing query 4 nested: Find top 5 customers based on total order amount in custorders
Rows:
{"name": "Customer#000000413", "totalOrderAmount": 8904393.46, "custkey": 413}
{"name": "Customer#000000686", "totalOrderAmount": 8261352.96, "custkey": 686}
{"name": "Customer#000001202", "totalOrderAmount": 8037459.56, "custkey": 1202}
{"name": "Customer#000000464", "totalOrderAmount": 7972306.16, "custkey": 464}
{"name": "Customer#000000098", "totalOrderAmount": 7961968.68, "custkey": 98}
{"totalOrderAmount": 7961968.68, "custkey": 98, "name": "Customer#000000098"}
{"totalOrderAmount": 7961968.68, "custkey": 98, "name": "Customer#000000098"}
Number of rows: 10

Executing query 4 nested: Find top 5 customers based on total order amount in custorders
Rows:
{"name": "Customer#000000413", "totalOrderAmount": 8904393.46, "custkey": 413}
{"name": "Customer#000000686", "totalOrderAmount": 8261352.96, "custkey": 686}
{"name": "Customer#000001202", "totalOrderAmount": 8037459.56, "custkey": 1202}
{"name": "Customer#000000464", "totalOrderAmount": 7972306.16, "custkey": 464}
{"name": "Customer#000000098", "totalOrderAmount": 7961968.68, "custkey": 98}
{"name": "Customer#000001202", "totalOrderAmount": 8037459.56, "custkey": 1202}
{"name": "Customer#000000464", "totalOrderAmount": 7972306.16, "custkey": 464}
{"name": "Customer#000000098", "totalOrderAmount": 7961968.68, "custkey": 98}
Number of rows: 5

Ricky Lucifer@Ricky-Lucifer MINGW64 /d/Company Work/IITJ/iitj/Trimister 3/Big Data Management/Assignment-7/iitj-bdm-as7-java-aws-mongodb (main)
$

Thank you sir for such a good hands on Assignment 7.
Regards

Shubham Raj
Roll No : G23AI2028