



Big data Management  
Assignment 7  
Jeyadev L  
G23AI2071

## Load Data

```
77
78  /**
79   * Ensures the collections exist and loads data into them.
80   *
81   * @param database MongoDB database instance
82   */
83  public static void createAndLoadCollections(MongoDatabase database) {
84      // File paths
85      String customerFilePath = "data/customer.tbl";
86      String ordersFilePath = "data/order.tbl";
87
88      // Ensure collections are created
89      MongoClient customerCollection = database.getCollection(collectionName:"customer");
90      MongoClient ordersCollection = database.getCollection(collectionName:"orders");
91
92      // Load data into collections
93      System.out.println("Loading customer data...");
94      loadPipeDelimitedData(customerFilePath, customerCollection, new String[]{
95          "c_custkey", "c_name", "c_address", "c_nationkey", "c_phone", "c_acctbal", "c_mktsegment", "c_comment"
96      });
97
98      System.out.println("Loading orders data...");
99      loadPipeDelimitedData(ordersFilePath, ordersCollection, new String[]{
100          "o_orderkey", "o_custkey", "o_orderstatus", "o_totalprice", "o_orderdate",
101          "o_orderpriority", "o_clerk", "o_shippriority", "o_comment"
102      });
103      loadNest(database, customerCollection, ordersCollection);
104  }
```

```

05  /**
06   * Loads pipe-delimited data from a file into a MongoDB collection.
07   *
08   * @param filePath Path to the pipe-delimited file
09   * @param collection MongoDB collection to load data into
10   * @param headers Array of field names corresponding to the file structure
11   */
12  private static void loadPipeDelimitedData(String filePath, MongoCollection<Document> collection, String[] headers) {
13      try (BufferedReader br = new BufferedReader(new FileReader(filePath))) {
14          String line;
15          List<Document> documents = new ArrayList<>();
16
17          while ((line = br.readLine()) != null) {
18              String[] values = line.split("\\|"); // Pipe-delimited
19              Document document = new Document();
20              for (int i = 0; i < headers.length; i++) {
21                  document.append(headers[i], parseValue(values[i]));
22              }
23              documents.add(document);
24
25              // Insert in batches of 1000 for performance
26              if (documents.size() >= 1000) {
27                  collection.insertMany(documents);
28                  documents.clear();
29              }
30
31              // Insert remaining documents
32              if (!documents.isEmpty()) {
33                  collection.insertMany(documents);
34              }
35
36              System.out.println("Loaded data into collection: " + collection.getNamespace().getCollectionName());
37          } catch (Exception e) {
38              System.err.println("Error loading data into collection: " + e.getMessage());
39              e.printStackTrace();
40          }
41      }
42  }
43

```

```

/**
 * Loads customer and orders data into a nested collection 'custorders'.
 *
 * @param database MongoDB database instance
 */
public static void loadNest(MongoDatabase database, MongoCollection<Document> customerCollection, MongoCollection<Document> ordersCollection) {

    // Target collection
    MongoCollection<Document> custOrdersCollection = database.getCollection(collectionName="custorders");

    // Clear the target collection if it exists
    custOrdersCollection.drop();

    System.out.println("Creating nested collection 'custorders'...");

    // Iterate through each customer
    try (MongoCursor<Document> customerCursor = customerCollection.find().iterator()) {
        List<Document> nestedDocuments = new ArrayList<>();
        while (customerCursor.hasNext()) {
            Document customer = customerCursor.next();
            int customerId = customer.getInteger(key:"c_custkey");

            // Find all orders for the current customer
            List<Document> orders = ordersCollection.find(new Document(key:"o_custkey", customerId)).into(new ArrayList<>());

            // Embed the orders in the customer document
            customer.append(key:"orders", orders);

            // Add the nested document to the list
            nestedDocuments.add(customer);

            // Insert in batches for performance
            if (nestedDocuments.size() >= 1000) {
                custOrdersCollection.insertMany(nestedDocuments);
                nestedDocuments.clear();
            }
        }

        // Insert remaining nested documents
        if (!nestedDocuments.isEmpty()) {
            custOrdersCollection.insertMany(nestedDocuments);
        }

        System.out.println("Data loaded into 'custorders' collection.");
    } catch (Exception e) {
        System.err.println("Error loading nested data: " + e.getMessage());
        e.printStackTrace();
    }
}

```

```

/**
 * Parses a string value to handle nulls or type conversions.
 *
 * @param value Raw string value
 * @return Parsed object (String, Integer, Double, or null)
 */
private static Object parseValue(String value) {
    if (value == null || value.isEmpty()) {
        return null;
    }
    try {
        if (value.contains(".")) {
            return Double.parseDouble(value);
        } else {
            return Integer.parseInt(value);
        }
    } catch (NumberFormatException e) {
        return value.trim(); // Default to String
    }
}

```

```

Loaded data into collection: customer
Loading orders data...
Loaded data into collection: orders
Creating nested collection 'custorders'...
Data loaded into 'custorders' collection.

```

MongoDB Compass - cluster-jdtrhf.mongodb.net/assignment\_7\_ID/customer

Connections Edit View Collection Help

Compass

My Queries

Search connections

cluster-jdtrhf.mongodb.net

admin

assignment\_7\_ID

customer

custorders

orders

config

local

cluster-jdtrhf.mongodb.net > assignment\_7\_ID > customer

Documents L&K Aggregations Schema Indexes Validation

Type a query: { field: 'value' } or [Generate query](#)

25 1 - 25 of 1800

#	customer	_id ObjectId	c_custkey Int32	c_name String	c_address String	c_nationkey Int32	c_phone String	c_acctbal Double	c_mkt
1	Object	ObjectId("6752afeb32dfcc...	1	"Customer#000000001"	"AbN2A0GR3 g1545v"	15	"25-989-741-2958"	711.56	"BUZLU /
2	Object	ObjectId("6752afeb32dfcc...	2	"Customer#000000002"	"W0L30ZNgY1x2"	13	"23-768-687-3665"	121.65	"AUTON /
3	Object	ObjectId("6752afeb32dfcc...	3	"Customer#000000003"	"PSL745KcWk20N6E1xgw7w...	1	"11-719-748-3364"	7498.12	"AUTON /
4	Object	ObjectId("6752afeb32dfcc...	4	"Customer#000000004"	"ekn150NPWPC1KSLx0B wD"	4	"14-119-199-5944"	2866.83	"MACK /
5	Object	ObjectId("6752afeb32dfcc...	5	"Customer#000000005"	"y0w5z0nPW1S0lQWPCkxLx2...	3	"13-758-942-6364"	794.47	"HDSU /
6	Object	ObjectId("6752afeb32dfcc...	6	"Customer#000000006"	"v570yK4n x511x385w1RzJ...	20	"38-114-968-4951"	7636.57	"AUTON /
7	Object	ObjectId("6752afeb32dfcc...	7	"Customer#000000007"	"Ch13840gA126GxQh17LjJ...	18	"28-159-982-9759"	9561.95	"AUTON /
8	Object	ObjectId("6752afeb32dfcc...	8	"Customer#000000008"	"kC8z8CunWgh4h4P5HQ18nL...	17	"27-147-574-9335"	6819.74	"BUZLU /
9	Object	ObjectId("6752afeb32dfcc...	9	"Customer#000000009"	"L4z63g28NgW6Pw584j0P8T...	8	"18-338-988-3675"	8324.07	"FURK /
10	Object	ObjectId("6752afeb32dfcc...	10	"Customer#000000010"	"L3j3j3xw16A8B183804ymw"	5	"15-741-346-9870"	2753.54	"HDSU /
11	Object	ObjectId("6752afeb32dfcc...	11	"Customer#000000011"	"8CPjy280x4"	23	"33-464-151-3439"	-272.6	"BUZLU /
12	Object	ObjectId("6752afeb32dfcc...	12	"Customer#000000012"	"152B882QhxC1AyL5M6CwL...	13	"23-791-270-1263"	3396.49	"HDSU /
13	Object	ObjectId("6752afeb32dfcc...	13	"Customer#000000013"	"wM1A1CT9T0L6Jx2M6C"	3	"13-761-547-5974"	3857.34	"BUZLU /
14	Object	ObjectId("6752afeb32dfcc...	14	"Customer#000000014"	"L28JL Q3j16LxL1 n2y 1y...	1	"11-845-129-3855"	5266.3	"FURK /
15	Object	ObjectId("6752afeb32dfcc...	15	"Customer#000000015"	"Cg33Bv1xy QwWzHQ"	23	"33-687-542-7885"	2788.52	"HDSU /
16	Object	ObjectId("6752afeb32dfcc...	16	"Customer#000000016"	"058J1LPDCzh"	18	"28-781-689-3187"	4681.83	"FURK /
17	Object	ObjectId("6752afeb32dfcc...	17	"Customer#000000017"	"y85R3J 1yH81w822L CzLzL...	2	"12-978-682-3487"	6.34	"AUTON /
18	Object	ObjectId("6752afeb32dfcc...	18	"Customer#000000018"	"9gh1w7PM65xwC3583A"	6	"16-155-215-1315"	5494.43	"BUZLU /
19	Object	ObjectId("6752afeb32dfcc...	19	"Customer#000000019"	"800h5zw8gy1P51zCP616LW...	18	"28-396-526-5853"	8914.71	"HDSU /
20	Object	ObjectId("6752afeb32dfcc...	20	"Customer#000000020"	"2nCR18j77w6ny12Ql0ggPgD...	22	"32-957-234-8742"	7683.4	"FURK /
21	Object	ObjectId("6752afeb32dfcc...	21	"Customer#000000021"	"2nQHLM8P1"	8	"18-992-614-8344"	1428.25	"MACK /
22	Object	ObjectId("6752afeb32dfcc...	22	"Customer#000000022"	"6w670Zw8Hq2120HL85MA0Q...	3	"13-886-545-9791"	591.88	"MACK /
23	Object	ObjectId("6752afeb32dfcc...	23	"Customer#000000023"	"11CnQ8LLCy3x3Qv0g7N0Zw...	3	"13-312-472-6245"	3332.82	"HDSU /

MongoDB Compass - cluster-jdtsrhf.mongodb.net/assignment\_T\_JD/orders

Connections Edit View Collection Help

Compass

My Queries

CONNECTIONS (1)

Search connections

cluster-jdtsrhf.mongodb.net

admin

assignment\_T\_JD

customer

customers

orders

config

local

cluster-jdtsrhf.mongodb.net > assignment\_T\_JD > orders

Documents 15.6K Aggregations Schema Indexes Validation

Type a query: ( field: 'value' ) or [Generate query](#)

Explain Reset Find Options

ADD DATA EXPORT DATA UPDATE DELETE

25 1 - 28 of 16000

	_id ObjectId	o_orderkey Int32	o_custkey Int32	o_orderstatus String	o_totalprice Double	o_orderdate String	o_orderpriority String	o_cl
1	ObjectID('6752afe332dfce...	1	781	"O"	172799.49	"1996-01-02"	"5-Low"	"Clari / / / / /
2	ObjectID('6752afe332dfce...	2	1234	"O"	41848.98	"1996-12-01"	"1-URGENT"	"Clari / / / / /
3	ObjectID('6752afe332dfce...	3	445	"P"	250878.73	"1993-10-14"	"5-Low"	"Clari / / / / /
4	ObjectID('6752afe332dfce...	4	557	"O"	6788.3	"1995-10-11"	"5-Low"	"Clari / / / / /
5	ObjectID('6752afe332dfce...	5	392	"P"	128227.38	"1994-07-30"	"5-Low"	"Clari / / / / /
6	ObjectID('6752afe332dfce...	6	1381	"P"	2566.57	"1992-02-21"	"4-NOT SPECIFIED"	"Clari / / / / /
7	ObjectID('6752afe332dfce...	7	678	"O"	235483.33	"1996-01-18"	"3-HIGH"	"Clari / / / / /
8	ObjectID('6752afe332dfce...	32	611	"O"	197403.12	"1995-07-16"	"2-HIGH"	"Clari / / / / /
9	ObjectID('6752afe332dfce...	33	1276	"P"	72247.39	"1993-10-27"	"3-MEDIUM"	"Clari / / / / /
10	ObjectID('6752afe332dfce...	34	1153	"O"	131039.87	"1996-07-21"	"3-MEDIUM"	"Clari / / / / /
11	ObjectID('6752afe332dfce...	35	882	"O"	171884.36	"1995-10-23"	"4-NOT SPECIFIED"	"Clari / / / / /
12	ObjectID('6752afe332dfce...	36	1249	"O"	81984.33	"1995-11-03"	"1-URGENT"	"Clari / / / / /
13	ObjectID('6752afe332dfce...	37	818	"P"	158846.36	"1992-06-03"	"3-MEDIUM"	"Clari / / / / /
14	ObjectID('6752afe332dfce...	38	322	"O"	7219.48	"1996-08-21"	"4-NOT SPECIFIED"	"Clari / / / / /
15	ObjectID('6752afe332dfce...	39	163	"O"	198663.57	"1996-09-28"	"3-MEDIUM"	"Clari / / / / /
16	ObjectID('6752afe332dfce...	64	1292	"P"	27435.96	"1994-07-16"	"3-MEDIUM"	"Clari / / / / /
17	ObjectID('6752afe332dfce...	65	286	"P"	196298.16	"1995-03-18"	"1-URGENT"	"Clari / / / / /
18	ObjectID('6752afe332dfce...	66	845	"P"	58896.21	"1994-01-28"	"5-Low"	"Clari / / / / /
19	ObjectID('6752afe332dfce...	67	644	"O"	288489.85	"1996-12-19"	"4-NOT SPECIFIED"	"Clari / / / / /
20	ObjectID('6752afe332dfce...	68	34	"O"	281976.75	"1998-04-18"	"3-MEDIUM"	"Clari / / / / /
21	ObjectID('6752afe332dfce...	69	1878	"P"	168572.43	"1994-06-04"	"4-NOT SPECIFIED"	"Clari / / / / /
22	ObjectID('6752afe332dfce...	70	211	"P"	327932.49	"1993-12-18"	"5-Low"	"Clari / / / / /
23	ObjectID('6752afe332dfce...	71	1845	"O"	398151.99	"1998-01-24"	"4-NOT SPECIFIED"	"Clari / / / / /

MongoDB Compass - cluster-jdtsrhf.mongodb.net/assignment\_T\_JD/customers

Connections Edit View Collection Help

Compass

My Queries

CONNECTIONS (1)

Search connections

cluster-jdtsrhf.mongodb.net

admin

assignment\_T\_JD

customer

customers

orders

config

local

cluster-jdtsrhf.mongodb.net > assignment\_T\_JD > customers

Documents 1.6K Aggregations Schema Indexes Validation

Type a query: ( field: 'value' ) or [Generate query](#)

Explain Reset Find Options

ADD DATA EXPORT DATA UPDATE DELETE

25 1 - 25 of 1800

	_id ObjectId	c_custkey Int32	c_name String	c_address String	c_nationkey Int32	c_phone String	c_acctbal Double	c_pk
1	ObjectID('6752afe332dfce...	1	"Customer#000000001"	"Abn2A8R3 g1S4Su"	15	"25-989-741-2988"	711.56	"BUZLI / / / / /
2	ObjectID('6752afe332dfce...	2	"Customer#000000002"	"M0L3C2Ngy1x2"	13	"23-768-687-3665"	121.65	"AUTON / / / / /
3	ObjectID('6752afe332dfce...	3	"Customer#000000003"	"PSL74SMCwkh2ON6Lxgw7w...	1	"11-719-748-1364"	7496.12	"AUTON / / / / /
4	ObjectID('6752afe332dfce...	4	"Customer#000000004"	"mkn15H8PHz1kSLw20B wD"	4	"14-128-190-5944"	2866.83	"RACH / / / / /
5	ObjectID('6752afe332dfce...	5	"Customer#000000005"	"yOw1znhPH1501QmPCkLx2...	3	"13-758-942-6364"	794.47	"HODSI / / / / /
6	ObjectID('6752afe332dfce...	6	"Customer#000000006"	"n57dykL4n k519k3Sw1kzj...	20	"38-114-968-4951"	7638.57	"AUTON / / / / /
7	ObjectID('6752afe332dfce...	7	"Customer#000000007"	"Ch1J840gA1n6AqH7LjJ...	18	"28-198-982-9759"	9561.95	"AUTON / / / / /
8	ObjectID('6752afe332dfce...	8	"Customer#000000008"	"kCR2CkNwHn4P58J8nLS...	17	"27-147-574-9335"	6819.74	"BUZLI / / / / /
9	ObjectID('6752afe332dfce...	9	"Customer#000000009"	"L4z63g2Rgh8PwN58Jp8T...	8	"18-338-980-3679"	8324.87	"FURK / / / / /
10	ObjectID('6752afe332dfce...	10	"Customer#000000010"	"L3Jg3xw16A8B18388Ayme"	5	"15-741-346-9678"	2753.34	"HODSI / / / / /
11	ObjectID('6752afe332dfce...	11	"Customer#000000011"	"8CPjy2B8x4"	23	"33-464-151-3439"	-272.6	"BUZLI / / / / /
12	ObjectID('6752afe332dfce...	12	"Customer#000000012"	"152B8B2Qv8vC1Ay1J8kCexT...	13	"23-791-270-1263"	3396.49	"HODSI / / / / /
13	ObjectID('6752afe332dfce...	13	"Customer#000000013"	"w8IA1CT5Q16Ljx2N6C"	3	"13-761-547-5974"	3857.34	"BUZLI / / / / /
14	ObjectID('6752afe332dfce...	14	"Customer#000000014"	"L2AJL Q3J1zL4L1 n2y 1y...	1	"11-845-129-3851"	5266.3	"FURK / / / / /
15	ObjectID('6752afe332dfce...	15	"Customer#000000015"	"CG8338xxy QcWzhQ"	23	"33-687-542-7601"	2788.52	"HODSI / / / / /
16	ObjectID('6752afe332dfce...	16	"Customer#000000016"	"058JL1P0Czhz"	19	"28-781-689-3187"	4681.83	"FURK / / / / /
17	ObjectID('6752afe332dfce...	17	"Customer#000000017"	"y85R3J jYh81w822L CzLzL...	2	"12-978-682-3487"	6.34	"AUTON / / / / /
18	ObjectID('6752afe332dfce...	18	"Customer#000000018"	"8ghjw7PM653wC35R5A"	6	"16-155-215-1315"	5494.43	"BUZLI / / / / /
19	ObjectID('6752afe332dfce...	19	"Customer#000000019"	"R0H5Zw8gy3PS1zCPk16W...	18	"28-396-526-5853"	8914.71	"HODSI / / / / /
20	ObjectID('6752afe332dfce...	20	"Customer#000000020"	"2nCR13T77m8fyJ2Ql0ggp8L...	22	"32-907-234-8742"	7683.4	"FURK / / / / /
21	ObjectID('6752afe332dfce...	21	"Customer#000000021"	"2nRHL8MP1"	8	"18-982-614-8344"	1428.25	"RACH / / / / /
22	ObjectID('6752afe332dfce...	22	"Customer#000000022"	"6wn70Zd8Hq120RL8PMQD...	3	"13-886-545-9702"	591.88	"RACH / / / / /
23	ObjectID('6752afe332dfce...	23	"Customer#000000023"	"*1CnQ8LLCyJkx3Qv8Q7N07...	3	"13-312-472-8245"	3332.82	"HODSI / / / / /

## Query 1 and 2

```
demo > src > main > java > com > example > J ExampleMongo.java > ExampleMongo > query2Nest(MongoDatabase, int)
17 public class ExampleMongo {
18
19     * Returns the customer name given a customer ID using the customer collection.
20     *
21     * @param database MongoDB database instance
22     * @param customerId The ID of the customer to query
23     * @return The customer's name, or null if no customer is found
24     */
25     public static String query1(MongoDatabase database, int customerId) {
26         // Access the customer collection
27         MongoClient<Document> customerCollection = database.getCollection(collectionName:"customer");
28
29         // Query the collection for the given customer ID
30         Document customer = customerCollection.find(eq("c_custkey", customerId)).first();
31
32         // Return the customer name if found, otherwise return null
33         if (customer != null) {
34             return customer.getString(key:"c_name");
35         }
36         return null;
37     }
38
39     /**
40     * Returns the order date for a given order ID using the orders collection.
41     *
42     * @param database MongoDB database instance
43     * @param orderId The ID of the order to query
44     * @return The order date, or null if no order is found
45     */
46     public static String query2(MongoDatabase database, int orderId) {
47         // Access the orders collection
48         MongoClient<Document> ordersCollection = database.getCollection(collectionName:"orders");
49
50         // Query the collection for the given order ID
51         Document order = ordersCollection.find(eq("o_orderkey", orderId)).first();
52
53         // Return the order date if found, otherwise return null
54         if (order != null) {
55             return order.getString(key:"o_orderdate");
56         }
57         return null;
58     }
59 }
60
61 PROBLEMS 1 OUTPUT DEBUG CONSOLE TERMINAL PORTS AZURE
Data loading completed successfully!
Customer Name: Customer#000000001
Order Date: 1996-01-02
```

## Query2Nest

```
File Edit Selection View Go Run Terminal Help
Assign-7_new
J ExampleMongo.java U
demo > src > main > java > com > example > J ExampleMongo.java > ExampleMongo > query2Nest(MongoDatabase, int)
17 public class ExampleMongo {
259
260     * Returns the order date for a given order ID using the custorders collection.
261     *
262     * @param database MongoDB database instance
263     * @param orderId The ID of the order to query
264     * @return The order date, or null if no order is found
265     */
266     public static String query2Nest(MongoDatabase database, int orderId) {
267         // Access the custorders collection
268         MongoClient<Document> custOrdersCollection = database.getCollection(collectionName:"custorders");
269
270         // Iterate through each document in custorders
271         for (Document customer : custOrdersCollection.find()) {
272             // Get the list of orders embedded in the customer document
273             List<Document> orders = customer.getList(key:"orders", clazz:Document.class);
274
275             // Search for the order with the given order ID
276             for (Document order : orders) {
277                 if (order.getInteger(key:"o_orderkey") == orderId) {
278                     return order.getString(key:"o_orderdate"); // Return the order date
279                 }
280             }
281         }
282         return null;
283     }
284 }
285
286 PROBLEMS 1 OUTPUT DEBUG CONSOLE TERMINAL PORTS AZURE
Order Date from custorders: 1996-01-02
```

### Query 3 and Query3Nest

```

1 // ExampleMongoJava X
2 demo > cd new java\com > example > J ExampleMongoJava > ExampleMongo > @ mongo/MongoDatabase
3
4 public class ExampleMongo {
5
6     /**
7      * Returns the total number of orders using the orders collection.
8      *
9      * @param database MongoClient database instance
10     * @return The total number of orders
11     */
12     public static long queryOrders(MongoDatabase database) {
13         // Access the orders collection
14         MongoClientDocument ordersCollection = database.getCollection(collectionName="orders");
15
16         // Use countDocuments() to get the total number of orders
17         return ordersCollection.countDocuments();
18     }
19
20     /**
21      * Returns the total number of orders using the customers collection.
22      *
23      * @param database MongoClient database instance
24      * @return The total number of orders
25      */
26     public static long queryCustomers(MongoDatabase database) {
27         // Access the customers collection
28         MongoClientDocument customerCollection = database.getCollection(collectionName="customers");
29
30         long totalOrders = 0;
31
32         // Iterate through each document in customers
33         for (Document customer : customerCollection.find()) {
34             // Get the list of orders associated in the customer document
35             List<Document> orders = customer.getList("orders", clazz=Document.class);
36             if (orders != null) {
37                 totalOrders += orders.size(); // Add the number of orders for the customer
38             }
39         }
40         return totalOrders;
41     }
42 }
43
44 PROBLEMS OUTPUT DEBUG CONSOLE TERMINAL PORTS Azure
45
46 [INFO] -----
47 [INFO] Building demo 1.0-SNAPSHOT
48 [INFO] from pom.xml
49 [INFO] -----
50 [INFO] [INFO]
51 [INFO] [INFO]
52 [INFO] [INFO]
53 [INFO] [INFO]
54 [INFO] [INFO]
55 [INFO] [INFO]
56 [INFO] [INFO]
57 [INFO] [INFO]
58 [INFO] [INFO]
59 [INFO] [INFO]
60 [INFO] [INFO]
61 [INFO] [INFO]
62 [INFO] [INFO]
63 [INFO] [INFO]
64 [INFO] [INFO]
65 [INFO] [INFO]
66 [INFO] [INFO]
67 [INFO] [INFO]
68 [INFO] [INFO]
69 [INFO] [INFO]
70 [INFO] [INFO]
71 [INFO] [INFO]
72 [INFO] [INFO]
73 [INFO] [INFO]
74 [INFO] [INFO]
75 [INFO] [INFO]
76 [INFO] [INFO]
77 [INFO] [INFO]
78 [INFO] [INFO]
79 [INFO] [INFO]
80 [INFO] [INFO]
81 [INFO] [INFO]
82 [INFO] [INFO]
83 [INFO] [INFO]
84 [INFO] [INFO]
85 [INFO] [INFO]
86 [INFO] [INFO]
87 [INFO] [INFO]
88 [INFO] [INFO]
89 [INFO] [INFO]
90 [INFO] [INFO]
91 [INFO] [INFO]
92 [INFO] [INFO]
93 [INFO] [INFO]
94 [INFO] [INFO]
95 [INFO] [INFO]
96 [INFO] [INFO]
97 [INFO] [INFO]
98 [INFO] [INFO]
99 [INFO] [INFO]
100 [INFO] [INFO]
101 [INFO] [INFO]
102 [INFO] [INFO]
103 [INFO] [INFO]
104 [INFO] [INFO]
105 [INFO] [INFO]
106 [INFO] [INFO]
107 [INFO] [INFO]
108 [INFO] [INFO]
109 [INFO] [INFO]
110 [INFO] [INFO]
111 [INFO] [INFO]
112 [INFO] [INFO]
113 [INFO] [INFO]
114 [INFO] [INFO]
115 [INFO] [INFO]
116 [INFO] [INFO]
117 [INFO] [INFO]
118 [INFO] [INFO]
119 [INFO] [INFO]
120 [INFO] [INFO]
121 [INFO] [INFO]
122 [INFO] [INFO]
123 [INFO] [INFO]
124 [INFO] [INFO]
125 [INFO] [INFO]
126 [INFO] [INFO]
127 [INFO] [INFO]
128 [INFO] [INFO]
129 [INFO] [INFO]
130 [INFO] [INFO]
131 [INFO] [INFO]
132 [INFO] [INFO]
133 [INFO] [INFO]
134 [INFO] [INFO]
135 [INFO] [INFO]
136 [INFO] [INFO]
137 [INFO] [INFO]
138 [INFO] [INFO]
139 [INFO] [INFO]
140 [INFO] [INFO]
141 [INFO] [INFO]
142 [INFO] [INFO]
143 [INFO] [INFO]
144 [INFO] [INFO]
145 [INFO] [INFO]
146 [INFO] [INFO]
147 [INFO] [INFO]
148 [INFO] [INFO]
149 [INFO] [INFO]
150 [INFO] [INFO]
151 [INFO] [INFO]
152 [INFO] [INFO]
153 [INFO] [INFO]
154 [INFO] [INFO]
155 [INFO] [INFO]
156 [INFO] [INFO]
157 [INFO] [INFO]
158 [INFO] [INFO]
159 [INFO] [INFO]
160 [INFO] [INFO]
161 [INFO] [INFO]
162 [INFO] [INFO]
163 [INFO] [INFO]
164 [INFO] [INFO]
165 [INFO] [INFO]
166 [INFO] [INFO]
167 [INFO] [INFO]
168 [INFO] [INFO]
169 [INFO] [INFO]
170 [INFO] [INFO]
171 [INFO] [INFO]
172 [INFO] [INFO]
173 [INFO] [INFO]
174 [INFO] [INFO]
175 [INFO] [INFO]
176 [INFO] [INFO]
177 [INFO] [INFO]
178 [INFO] [INFO]
179 [INFO] [INFO]
180 [INFO] [INFO]
181 [INFO] [INFO]
182 [INFO] [INFO]
183 [INFO] [INFO]
184 [INFO] [INFO]
185 [INFO] [INFO]
186 [INFO] [INFO]
187 [INFO] [INFO]
188 [INFO] [INFO]
189 [INFO] [INFO]
190 [INFO] [INFO]
191 [INFO] [INFO]
192 [INFO] [INFO]
193 [INFO] [INFO]
194 [INFO] [INFO]
195 [INFO] [INFO]
196 [INFO] [INFO]
197 [INFO] [INFO]
198 [INFO] [INFO]
199 [INFO] [INFO]
200 [INFO] [INFO]
201 [INFO] [INFO]
202 [INFO] [INFO]
203 [INFO] [INFO]
204 [INFO] [INFO]
205 [INFO] [INFO]
206 [INFO] [INFO]
207 [INFO] [INFO]
208 [INFO] [INFO]
209 [INFO] [INFO]
210 [INFO] [INFO]
211 [INFO] [INFO]
212 [INFO] [INFO]
213 [INFO] [INFO]
214 [INFO] [INFO]
215 [INFO] [INFO]
216 [INFO] [INFO]
217 [INFO] [INFO]
218 [INFO] [INFO]
219 [INFO] [INFO]
220 [INFO] [INFO]
221 [INFO] [INFO]
222 [INFO] [INFO]
223 [INFO] [INFO]
224 [INFO] [INFO]
225 [INFO] [INFO]
226 [INFO] [INFO]
227 [INFO] [INFO]
228 [INFO] [INFO]
229 [INFO] [INFO]
230 [INFO] [INFO]
231 [INFO] [INFO]
232 [INFO] [INFO]
233 [INFO] [INFO]
234 [INFO] [INFO]
235 [INFO] [INFO]
236 [INFO] [INFO]
237 [INFO] [INFO]
238 [INFO] [INFO]
239 [INFO] [INFO]
240 [INFO] [INFO]
241 [INFO] [INFO]
242 [INFO] [INFO]
243 [INFO] [INFO]
244 [INFO] [INFO]
245 [INFO] [INFO]
246 [INFO] [INFO]
247 [INFO] [INFO]
248 [INFO] [INFO]
249 [INFO] [INFO]
250 [INFO] [INFO]
251 [INFO] [INFO]
252 [INFO] [INFO]
253 [INFO] [INFO]
254 [INFO] [INFO]
255 [INFO] [INFO]
256 [INFO] [INFO]
257 [INFO] [INFO]
258 [INFO] [INFO]
259 [INFO] [INFO]
260 [INFO] [INFO]
261 [INFO] [INFO]
262 [INFO] [INFO]
263 [INFO] [INFO]
264 [INFO] [INFO]
265 [INFO] [INFO]
266 [INFO] [INFO]
267 [INFO] [INFO]
268 [INFO] [INFO]
269 [INFO] [INFO]
270 [INFO] [INFO]
271 [INFO] [INFO]
272 [INFO] [INFO]
273 [INFO] [INFO]
274 [INFO] [INFO]
275 [INFO] [INFO]
276 [INFO] [INFO]
277 [INFO] [INFO]
278 [INFO] [INFO]
279 [INFO] [INFO]
280 [INFO] [INFO]
281 [INFO] [INFO]
282 [INFO] [INFO]
283 [INFO] [INFO]
284 [INFO] [INFO]
285 [INFO] [INFO]
286 [INFO] [INFO]
287 [INFO] [INFO]
288 [INFO] [INFO]
289 [INFO] [INFO]
290 [INFO] [INFO]
291 [INFO] [INFO]
292 [INFO] [INFO]
293 [INFO] [INFO]
294 [INFO] [INFO]
295 [INFO] [INFO]
296 [INFO] [INFO]
297 [INFO] [INFO]
298 [INFO] [INFO]
299 [INFO] [INFO]
300 [INFO] [INFO]
301 [INFO] [INFO]
302 [INFO] [INFO]
303 [INFO] [INFO]
304 [INFO] [INFO]
305 [INFO] [INFO]
306 [INFO] [INFO]
307 [INFO] [INFO]
308 [INFO] [INFO]
309 [INFO] [INFO]
310 [INFO] [INFO]
311 [INFO] [INFO]
312 [INFO] [INFO]
313 [INFO] [INFO]
314 [INFO] [INFO]
315 [INFO] [INFO]
316 [INFO] [INFO]
317 [INFO] [INFO]
318 [INFO] [INFO]
319 [INFO] [INFO]
320 [INFO] [INFO]
321 [INFO] [INFO]
322 [INFO] [INFO]
323 [INFO] [INFO]
324 [INFO] [INFO]
325 [INFO] [INFO]
326 [INFO] [INFO]
327 [INFO] [INFO]
328 [INFO] [INFO]
329 [INFO] [INFO]
330 [INFO] [INFO]
331 [INFO] [INFO]
332 [INFO] [INFO]
333 [INFO] [INFO]
334 [INFO] [INFO]
335 [INFO] [INFO]
336 [INFO] [INFO]
337 [INFO] [INFO]
338 [INFO] [INFO]
339 [INFO] [INFO]
340 [INFO] [INFO]
341 [INFO] [INFO]
342 [INFO] [INFO]
343 [INFO] [INFO]
344 [INFO] [INFO]
345 [INFO] [INFO]
346 [INFO] [INFO]
347 [INFO] [INFO]
348 [INFO] [INFO]
349 [INFO] [INFO]
350 [INFO] [INFO]
351 [INFO] [INFO]
352 [INFO] [INFO]
353 [INFO] [INFO]
354 [INFO] [INFO]
355 [INFO] [INFO]
356 [INFO] [INFO]
357 [INFO] [INFO]
358 [INFO] [INFO]
359 [INFO] [INFO]
360 [INFO] [INFO]
361 [INFO] [INFO]
362 [INFO] [INFO]
363 [INFO] [INFO]
364 [INFO] [INFO]
365 [INFO] [INFO]
366 [INFO] [INFO]
367 [INFO] [INFO]
368 [INFO] [INFO]
369 [INFO] [INFO]
370 [INFO] [INFO]
371 [INFO] [INFO]
372 [INFO] [INFO]
373 [INFO] [INFO]
374 [INFO] [INFO]
375 [INFO] [INFO]
376 [INFO] [INFO]
377 [INFO] [INFO]
378 [INFO] [INFO]
379 [INFO] [INFO]
380 [INFO] [INFO]
381 [INFO] [INFO]
382 [INFO] [INFO]
383 [INFO] [INFO]
384 [INFO] [INFO]
385 [INFO] [INFO]
386 [INFO] [INFO]
387 [INFO] [INFO]
388 [INFO] [INFO]
389 [INFO] [INFO]
390 [INFO] [INFO]
391 [INFO] [INFO]
392 [INFO] [INFO]
393 [INFO] [INFO]
394 [INFO] [INFO]
395 [INFO] [INFO]
396 [INFO] [INFO]
397 [INFO] [INFO]
398 [INFO] [INFO]
399 [INFO] [INFO]
400 [INFO] [INFO]
401 [INFO] [INFO]
402 [INFO] [INFO]
403 [INFO] [INFO]
404 [INFO] [INFO]
405 [INFO] [INFO]
406 [INFO] [INFO]

```

### Query 4

```

17 public class ExampleKungu {
18     public static long queryWest(KunguDatabase database) {
19     }
20 }
21
22 // Returns the top 5 customers based on total order amount using the customer and orders collections.
23 -
24 * Return database MongoDB database instance
25 * Return a list of top 5 customers as documents, including their total order amount
26 -
27 public static List<Document> query4(KunguDatabase database) {
28     // Access the customer and orders collections
29     MongoCollection<Document> customerCollection = database.getCollection(collectionName<"customer">);
30     MongoCollection<Document> ordersCollection = database.getCollection(collectionName<"orders">);
31     // Map to store total order amount by customer ID
32     Map<Integer, Double> customerOrderTotals = new HashMap<>();
33     // Iterate through all orders to calculate total order amount for each customer
34     try (MongoCursor<Document> orderCursor = ordersCollection.find().iterator()) {
35         while (orderCursor.hasNext()) {
36             Document order = orderCursor.next();
37             int customerId = order.getInteger(key<"customer_id">);
38             double orderAmount = order.getDouble(key<"total_order_amount">);
39
40             customerOrderTotals.put(customerId, customerOrderTotals.getOrDefault(customerId, 0.0) + orderAmount);
41         }
42     }
43
44     // Retrieve customer details and include total order amount
45     List<Document> customersWithTotals = customerCollection.find(
46         new<Document> {
47             $eq<Integer> {
48                 int customerId = customer.getIdInteger(key<"customer_id">);
49                 double totalOrderAmount = customerOrderTotals.getOrDefault(customerId, 0.0);
50                 customer.append(key<"total_order_amount">, totalOrderAmount);
51             }
52         }
53     );
54     // Sort customers by total order amount in descending order and return the top 5
55     return customersWithTotals.stream()
56         .sorted((c1, c2) -> Double.compare(c2.getDouble(key<"total_order_amount">), c1.getDouble(key<"total_order_amount">)))
57         .limit(5)
58         .collect(Collectors.toList());
59 }
60
61 // Returns the top 5 customers based on total order amount using the customer and orders collections.
62 -
63 * Return database MongoDB database instance
64 * Return a list of top 5 customers as documents, including their total order amount
65 -
66 public static List<Document> query5(KunguDatabase database) {
67     // Access the customer and orders collections
68     MongoCollection<Document> customerCollection = database.getCollection(collectionName<"customer">);
69     MongoCollection<Document> ordersCollection = database.getCollection(collectionName<"orders">);
70     // Map to store total order amount by customer ID
71     Map<Integer, Double> customerOrderTotals = new HashMap<>();
72     // Iterate through all orders to calculate total order amount for each customer
73     try (MongoCursor<Document> orderCursor = ordersCollection.find().iterator()) {
74         while (orderCursor.hasNext()) {
75             Document order = orderCursor.next();
76             int customerId = order.getInteger(key<"customer_id">);
77             double orderAmount = order.getDouble(key<"total_order_amount">);
78
79             customerOrderTotals.put(customerId, customerOrderTotals.getOrDefault(customerId, 0.0) + orderAmount);
80         }
81     }
82
83     // Retrieve customer details and include total order amount
84     List<Document> customersWithTotals = customerCollection.find(
85         new<Document> {
86             $eq<Integer> {
87                 int customerId = customer.getIdInteger(key<"customer_id">);
88                 double totalOrderAmount = customerOrderTotals.getOrDefault(customerId, 0.0);
89                 customer.append(key<"total_order_amount">, totalOrderAmount);
90             }
91         }
92     );
93     // Sort customers by total order amount in descending order and return the top 5
94     return customersWithTotals.stream()
95         .sorted((c1, c2) -> Double.compare(c2.getDouble(key<"total_order_amount">), c1.getDouble(key<"total_order_amount">)))
96         .limit(5)
97         .collect(Collectors.toList());
98 }
99
100 // Returns the top 5 customers based on total order amount using the customer and orders collections.
101 -
102 * Return database MongoDB database instance
103 * Return a list of top 5 customers as documents, including their total order amount
104 -
105 public static List<Document> query6(KunguDatabase database) {
106     // Access the customer and orders collections
107     MongoCollection<Document> customerCollection = database.getCollection(collectionName<"customer">);
108     MongoCollection<Document> ordersCollection = database.getCollection(collectionName<"orders">);
109     // Map to store total order amount by customer ID
110     Map<Integer, Double> customerOrderTotals = new HashMap<>();
111     // Iterate through all orders to calculate total order amount for each customer
112     try (MongoCursor<Document> orderCursor = ordersCollection.find().iterator()) {
113         while (orderCursor.hasNext()) {
114             Document order = orderCursor.next();
115             int customerId = order.getInteger(key<"customer_id">);
116             double orderAmount = order.getDouble(key<"total_order_amount">);
117
118             customerOrderTotals.put(customerId, customerOrderTotals.getOrDefault(customerId, 0.0) + orderAmount);
119         }
120     }
121
122     // Retrieve customer details and include total order amount
123     List<Document> customersWithTotals = customerCollection.find(
124         new<Document> {
125             $eq<Integer> {
126                 int customerId = customer.getIdInteger(key<"customer_id">);
127                 double totalOrderAmount = customerOrderTotals.getOrDefault(customerId, 0.0);
128                 customer.append(key<"total_order_amount">, totalOrderAmount);
129             }
130         }
131     );
132     // Sort customers by total order amount in descending order and return the top 5
133     return customersWithTotals.stream()
134         .sorted((c1, c2) -> Double.compare(c2.getDouble(key<"total_order_amount">), c1.getDouble(key<"total_order_amount">)))
135         .limit(5)
136         .collect(Collectors.toList());
137 }
138
139 // Returns the top 5 customers based on total order amount using the customer and orders collections.
140 -
141 * Return database MongoDB database instance
142 * Return a list of top 5 customers as documents, including their total order amount
143 -
144 public static List<Document> query7(KunguDatabase database) {
145     // Access the customer and orders collections
146     MongoCollection<Document> customerCollection = database.getCollection(collectionName<"customer">);
147     MongoCollection<Document> ordersCollection = database.getCollection(collectionName<"orders">);
148     // Map to store total order amount by customer ID
149     Map<Integer, Double> customerOrderTotals = new HashMap<>();
150     // Iterate through all orders to calculate total order amount for each customer
151     try (MongoCursor<Document> orderCursor = ordersCollection.find().iterator()) {
152         while (orderCursor.hasNext()) {
153             Document order = orderCursor.next();
154             int customerId = order.getInteger(key<"customer_id">);
155             double orderAmount = order.getDouble(key<"total_order_amount">);
156
157             customerOrderTotals.put(customerId, customerOrderTotals.getOrDefault(customerId, 0.0) + orderAmount);
158         }
159     }
160
161     // Retrieve customer details and include total order amount
162     List<Document> customersWithTotals = customerCollection.find(
163         new<Document> {
164             $eq<Integer> {
165                 int customerId = customer.getIdInteger(key<"customer_id">);
166                 double totalOrderAmount = customerOrderTotals.getOrDefault(customerId, 0.0);
167                 customer.append(key<"total_order_amount">, totalOrderAmount);
168             }
169         }
170     );
171     // Sort customers by total order amount in descending order and return the top 5
172     return customersWithTotals.stream()
173         .sorted((c1, c2) -> Double.compare(c2.getDouble(key<"total_order_amount">), c1.getDouble(key<"total_order_amount">)))
174         .limit(5)
175         .collect(Collectors.toList());
176 }
177
178 // Returns the top 5 customers based on total order amount using the customer and orders collections.
179 -
180 * Return database MongoDB database instance
181 * Return a list of top 5 customers as documents, including their total order amount
182 -
183 public static List<Document> query8(KunguDatabase database) {
184     // Access the customer and orders collections
185     MongoCollection<Document> customerCollection = database.getCollection(collectionName<"customer">);
186     MongoCollection<Document> ordersCollection = database.getCollection(collectionName<"orders">);
187     // Map to store total order amount by customer ID
188     Map<Integer, Double> customerOrderTotals = new HashMap<>();
189     // Iterate through all orders to calculate total order amount for each customer
190     try (MongoCursor<Document> orderCursor = ordersCollection.find().iterator()) {
191         while (orderCursor.hasNext()) {
192             Document order = orderCursor.next();
193             int customerId = order.getInteger(key<"customer_id">);
194             double orderAmount = order.getDouble(key<"total_order_amount">);
195
196             customerOrderTotals.put(customerId, customerOrderTotals.getOrDefault(customerId, 0.0) + orderAmount);
197         }
198     }
199
200     // Retrieve customer details and include total order amount
201     List<Document> customersWithTotals = customerCollection.find(
202         new<Document> {
203             $eq<Integer> {
204                 int customerId = customer.getIdInteger(key<"customer_id">);
205                 double totalOrderAmount = customerOrderTotals.getOrDefault(customerId, 0.0);
206                 customer.append(key<"total_order_amount">, totalOrderAmount);
207             }
208         }
209     );
210     // Sort customers by total order amount in descending order and return the top 5
211     return customersWithTotals.stream()
212         .sorted((c1, c2) -> Double.compare(c2.getDouble(key<"total_order_amount">), c1.getDouble(key<"total_order_amount">)))
213         .limit(5)
214         .collect(Collectors.toList());
215 }
216
217 // Returns the top 5 customers based on total order amount using the customer and orders collections.
218 -
219 * Return database MongoDB database instance
220 * Return a list of top 5 customers as documents, including their total order amount
221 -
222 public static List<Document> query9(KunguDatabase database) {
223     // Access the customer and orders collections
224     MongoCollection<Document> customerCollection = database.getCollection(collectionName<"customer">);
225     MongoCollection<Document> ordersCollection = database.getCollection(collectionName<"orders">);
226     // Map to store total order amount by customer ID
227     Map<Integer, Double> customerOrderTotals = new HashMap<>();
228     // Iterate through all orders to calculate total order amount for each customer
229     try (MongoCursor<Document> orderCursor = ordersCollection.find().iterator()) {
230         while (orderCursor.hasNext()) {
231             Document order = orderCursor.next();
232             int customerId = order.getInteger(key<"customer_id">);
233             double orderAmount = order.getDouble(key<"total_order_amount">);
234
235             customerOrderTotals.put(customerId, customerOrderTotals.getOrDefault(customerId, 0.0) + orderAmount);
236         }
237     }
238
239     // Retrieve customer details and include total order amount
240     List<Document> customersWithTotals = customerCollection.find(
241         new<Document> {
242             $eq<Integer> {
243                 int customerId = customer.getIdInteger(key<"customer_id">);
244                 double totalOrderAmount = customerOrderTotals.getOrDefault(customerId, 0.0);
245                 customer.append(key<"total_order_amount">, totalOrderAmount);
246             }
247         }
248     );
249     // Sort customers by total order amount in descending order and return the top 5
250     return customersWithTotals.stream()
251         .sorted((c1, c2) -> Double.compare(c2.getDouble(key<"total_order_amount">), c1.getDouble(key<"total_order_amount">)))
252         .limit(5)
253         .collect(Collectors.toList());
254 }
255
256 // Returns the top 5 customers based on total order amount using the customer and orders collections.
257 -
258 * Return database MongoDB database instance
259 * Return a list of top 5 customers as documents, including their total order amount
260 -
261 public static List<Document> query10(KunguDatabase database) {
262     // Access the customer and orders collections
263     MongoCollection<Document> customerCollection = database.getCollection(collectionName<"customer">);
264     MongoCollection<Document> ordersCollection = database.getCollection(collectionName<"orders">);
265     // Map to store total order amount by customer ID
266     Map<Integer, Double> customerOrderTotals = new HashMap<>();
267     // Iterate through all orders to calculate total order amount for each customer
268     try (MongoCursor<Document> orderCursor = ordersCollection.find().iterator()) {
269         while (orderCursor.hasNext()) {
270             Document order = orderCursor.next();
271             int customerId = order.getInteger(key<"customer_id">);
272             double orderAmount = order.getDouble(key<"total_order_amount">);
273
274             customerOrderTotals.put(customerId, customerOrderTotals.getOrDefault(customerId, 0.0) + orderAmount);
275         }
276     }
277
278     // Retrieve customer details and include total order amount
279     List<Document> customersWithTotals = customerCollection.find(
280         new<Document> {
281             $eq<Integer> {
282                 int customerId = customer.getIdInteger(key<"customer_id">);
283                 double totalOrderAmount = customerOrderTotals.getOrDefault(customerId, 0.0);
284                 customer.append(key<"total_order_amount">, totalOrderAmount);
285             }
286         }
287     );
288     // Sort customers by total order amount in descending order and return the top 5
289     return customersWithTotals.stream()
290         .sorted((c1, c2) -> Double.compare(c2.getDouble(key<"total_order_amount
```



## Query4Nest

[illegible]