



**COEN 6731: Distributed Software Systems**

**Assignment – II**

**Submitted to – YAN LIU**

**Submitted by –Rohan K (40196377)**

**Date of submission – April 3, 2023**

## Task 1. MongoDB Data Storage and Operation

**Task 1.1)** To create a MongoDB collection named EduCostStat to store the data to the MongoDB instance running on MongoDB online cluster.

Below are the flow of operations I did to achieve this –

1. Created Mongo DB Atlas account
2. Downloaded the dataset as a .csv file.
3. Coded a program to create a collection by reading the data samples from file. The database named “rondb” and cluster “EduCostStat” is created. (snippet of code is below)

```
Welcome JS import.js Year.txt
Users > rohankodavalla > Desktop > JS import.js > ...
1  const csv = require('csv-parser');
2  const fs = require('fs');
3  const MongoClient = require('mongodb').MongoClient;
4  // Connection URI
5  const uri = 'mongodb+srv://rkodava:<password>@educoststat.ioim58e.mongodb.net/test';
6  // Database Name
7  const dbName = 'rondb';
8  // Collection Name
9  const collName = 'EduCostStat';
10 // MongoDB options
11 const clientOptions = {
12   useNewUrlParser: true,
13   useUnifiedTopology: true,
14   retryWrites: true,
15   w: 'majority'
16 };
17 async function main() {
18   const client = new MongoClient(uri, clientOptions);
19   try {
20     // Connect to the MongoDB cluster
21     await client.connect();
22     console.log('Connected to MongoDB');
23     // Access the database and collection
24     const db = client.db(dbName);
25     const coll = db.collection(collName);
26     // Read the CSV file and insert documents into the collection
27     fs.createReadStream('/Users/rohankodavalla/Desktop/STUDY/winter23/6731-distributed/assign-2/nces33020.csv')
28       .pipe(csv())
29       .on('data', async (data) => {
30         // Insert a document
31         const result = await coll.insertOne(data);
32         console.log(`Inserted document with _id: ${result.insertedId}`);
33       })
34       .on('end', () => {
35         console.log('CSV file successfully processed');
36       });
37   } catch (err) {
38     console.error(err);
39   } finally {
40     // Close the MongoDB client
41     await client.close();
42     console.log('Disconnected from MongoDB');
43   }
44 }
45 main().catch(console.error);
```

Below is the explanation of the code:

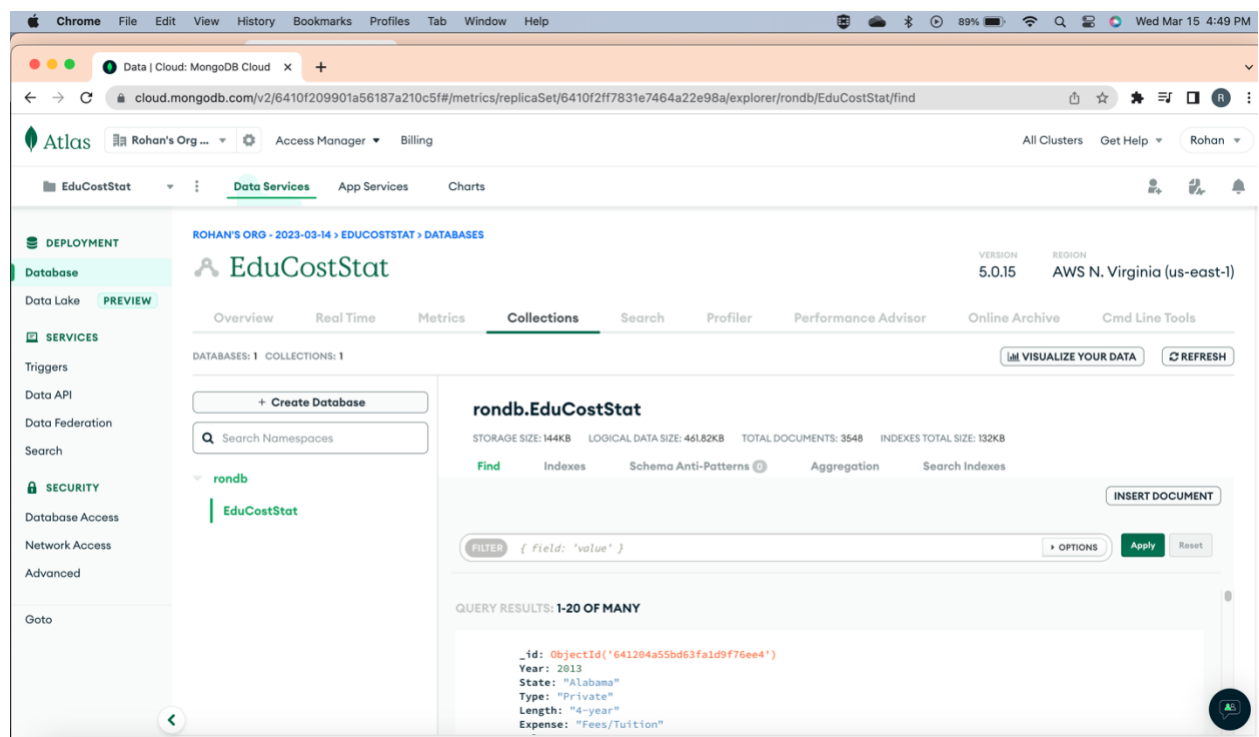
- The code is importing three dependencies: csv-parser, fs, and MongoClient from the mongodb library.
- A MongoDB connection URI is defined, which includes the username and password required to authenticate to the Atlas cluster.
- A database name and collection name are defined for the target MongoDB instance.
- MongoDB options are set, including the use of the useNewUrlParser, useUnifiedTopology, retryWrites, and w options.
- The main() function is defined as an asynchronous function.
- Within the main() function, a new instance of MongoClient is created using the URI and options defined earlier.
- An attempt is made to connect to the MongoDB cluster using the connect() method on the MongoClient instance. If successful, a message is printed to the console.
- The db and coll variables are used to reference the target database and collection.
- A CSV file is read using fs.createReadStream() and piped through csv-parser to parse each row as an object.
- For each row of data in the CSV file, a new document is inserted into the MongoDB collection using coll.insertOne(). The inserted document ID is logged to the console.
- After all rows of data have been processed, a message is logged to the console.
- If an error occurs during any of the above operations, it is logged to the console.
- The MongoDB client connection is closed using client.close(), and a message is printed to the console indicating that the connection has been closed.
- The main() function is called, and any errors encountered during its execution are logged to the console.

- Further, navigated to the directory where the .js file is located in my terminal and use the following command: "node import.js".

Below are the steps achieved by the code –

- If the connection to the MongoDB cluster is successful, the console will log "Connected to MongoDB".
- The code will read the specified CSV file and insert the data as documents into the specified MongoDB collection.
- As each document is inserted, the console will log a message indicating the inserted document's ID.
- Once all documents have been inserted, the console will log "CSV file successfully processed".
- If there are any errors, the console will log the error message.
- Finally, the console will log "Disconnected from MongoDB" when the connection is closed.

The dashboard of MongoDB is below



As shown, db name = rondb, collection = EduCostStat , with 3548 documents. The next 5 queries have been done in db = test which is a duplicate of rondb, in order to avoid any impact of huge mistakes.

Task1.2)

1) **Query the cost given specific year, state, type, length, expense; and save the query as a document in a collection named EduCostStatQueryOne :**

To develop data access objects in Java that represent different queries to the data and save the query as a document in a collection named EduCostStatQueryOne, these steps were done:

1. Added the MongoDB Java driver dependency to the project.
2. Created a Java class named EduCostStatQueryOne and define instance variables for year, state, type, length, and expense.
3. Create a constructor for the EduCostStatQueryOne class that initializes the instance variables.
4. Define a method named queryCost() in the EduCostStatQueryOne class that performs the query on the EduCostStat collection in MongoDB Atlas and saves the query result as a document in the EduCostStatQueryOne collection.
5. In the queryCost () method, the MongoDB Java driver is used to connect to the MongoDB Atlas cluster, access the EduCostStat and EduCostStatQueryOne collections, and perform the query using the instance variables.
6. Checks if the query result already exists in the EduCostStatQueryOne collection before inserting a new document to avoid duplicates and inserts it into the EduCostStatQueryOne collection.
7. Close the connection to the MongoDB Atlas cluster.

Here's the implementation of the EduCostStatQueryOne:

```
src > main > java > com > assign > app > J EduCostStatQueryOne.java > EduCostStatQueryOne > queryCost(int, String, String, String, String, String, MongoCollection<Document>, MongoCollection<Document>)
1 package com.assign.app;
2
3 import com.mongodb.client.MongoClients;
4 import com.mongodb.client.MongoClient;
5 import com.mongodb.client.MongoCollection;
6 import com.mongodb.client.MongoDatabase;
7 import com.mongodb.client.model.Filters;
8 import org.bson.Document;
9
10
11 public class EduCostStatQueryOne {
12
13     private static final String DB_NAME = "test";
14     private static final String COLLECTION_NAME = "EduCostStat";
15     private static final String QUERY_COLLECTION_NAME = "EduCostStatQueryOne";
16
17     Run | Debug
18     public static void main(String[] args) {
19         // Set up MongoDB connection
20         MongoClient mongoClient = MongoClients.create(connectionString:"mongodb+srv://rkodava:Dimpu1997@educoststat.ioim58e.mongodb.net/?retryWrites=true&w=majority");
21         MongoDatabase database = mongoClient.getDatabase(DB_NAME);
22         MongoCollection<Document> collection = database.getCollection(COLLECTION_NAME);
23         MongoCollection<Document> queryCollection = database.getCollection(QUERY_COLLECTION_NAME);
24
25         // Call the query method
26         int year = 2013;
27         String state = "Alabama";
28         String type = "Private";
29         String length = "4-year";
30         String expense = "Fees/Tuition";
31         Document queryResult = queryCost(year, state, type, length, expense, collection, queryCollection);
32
33         // Print the query result
34         System.out.println(queryResult.toJson());
35
36         // Close MongoDB connection
37         mongoClient.close();
38     }
39
40     public static Document queryCost(int year, String state, String type, String length, String expense, MongoCollection<Document> collection, MongoCollection<Document> queryCollection) {
41         // Check if the query already exists in the query collection
42         Document queryDoc = queryCollection.find(Filters.and(
43             Filters.eq(fieldName:"year", year),
44             Filters.eq(fieldName:"state", state),
45             Filters.eq(fieldName:"type", type),
46             Filters.eq(fieldName:"length", length),
47             Filters.eq(fieldName:"expense", expense)
48         )).first();
49         if (queryDoc != null) {
50             return queryDoc;
51         }
52
53         // Query the cost from the EduCostStat collection
54         Document costDoc = collection.find(Filters.and(
```

```
51
52         // Query the cost from the EduCostStat collection
53         Document costDoc = collection.find(Filters.and(
54             Filters.eq(fieldName:"year", year),
55             Filters.eq(fieldName:"state", state),
56             Filters.eq(fieldName:"type", type),
57             Filters.eq(fieldName:"length", length)
58         )).first();
59
60         // Create a new query document with the query parameters and the query result
61         Document queryResult = new Document()
62             .append(key:"year", year)
63             .append(key:"state", state)
64             .append(key:"type", type)
65             .append(key:"length", length)
66             .append(key:"expense", expense)
67             .append(key:"cost", costDoc.getDouble(expense));
68
69         // Insert the new query document into the query collection
70         queryCollection.insertOne(queryResult);
71
72         return queryResult;
73     }
74 }
75
```

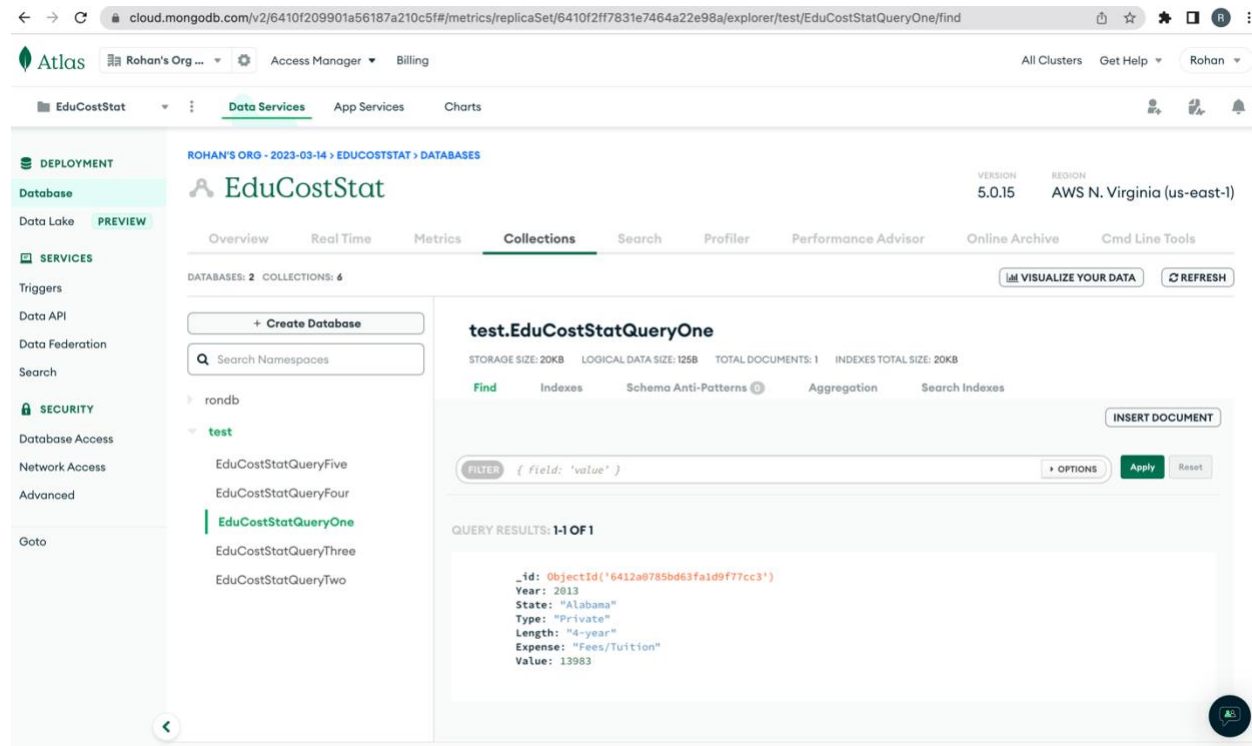
This Java code queries the education cost of a specific expense category in a given year, state, type, and length of education. The code connects to a MongoDB Atlas database and retrieves data from a collection named "EduCostStat".

The program defines a class called "EduCostStatQueryOne" with a "main" method. In the main method, the code establishes a connection to the MongoDB Atlas database, creates references to the "EduCostStat" collection and a new collection called "EduCostStatQueryOne" to store the query results.

Then the code calls a method "queryCost" which accepts the input parameters of year, state, type, length, expense, and references to the "EduCostStat" collection and "EduCostStatQueryOne" collection. The method checks if the query already exists in the "EduCostStatQueryOne" collection, and if so, returns the existing result document. If not, the method queries the cost of the specified expense category using the given input parameters from the "EduCostStat" collection, and creates a new document that contains the input parameters and the cost result. Finally, the method inserts the new document into the "EduCostStatQueryOne" collection.

In the main method, the code calls the "queryCost" method with the input parameters and retrieves the resulting document. The program prints the JSON representation of the resulting document to the console and then closes the MongoDB connection.

The result from the above code in MongoDB is as below :



The screenshot shows the MongoDB Atlas web interface. The top navigation bar includes the Atlas logo, the organization name "Rohan's Org", and various settings like "Access Manager" and "Billing". The main content area is divided into a left sidebar with navigation options (Deployment, Database, Services, Security) and a main panel. The main panel shows the "EduCostStat" database with a list of collections. The "test" collection is selected, and the "test.EduCostStatQueryOne" collection is displayed. The collection details show a storage size of 20KB, logical data size of 125B, total documents of 1, and indexes total size of 20KB. A search bar is present, and the "Find" button is highlighted. Below the search bar, the query results are shown as a single document in JSON format:

```
{
  "_id": ObjectId("6412a0785bd63fa1d9f77cc3"),
  "Year": 2013,
  "State": "Alabama",
  "Type": "Private",
  "Length": "4-year",
  "Expense": "Fees/Tuition",
  "Value": 13983
}
```

2) **Query the top 5 most expensive states (with overall expense) given a year, type, length; and save the query as a document in a collection named EduCostStatQueryTwo :**

Below is a Java program that queries data from a MongoDB database and returns the top 5 most expensive states for a given year, type, and length of education program. The program uses the MongoDB Java driver to connect to the database and execute the query.

Here's a brief overview of the program's structure and function:

The program starts by importing several classes from the MongoDB Java driver and the standard Java library.

The program defines three constants: `DB_NAME`, `COLLECTION_NAME`, and `QUERY_COLLECTION_NAME`, which are used to specify the names of the database, collection, and query collection in the MongoDB database.

The `main()` method sets up a connection to the MongoDB database, retrieves the specified collection and query collection, and calls the `queryTop5ExpensiveStates()` method to execute the query.

The `queryTop5ExpensiveStates()` method first checks if the query has already been executed and saved in the query collection. If so, it returns the saved query result.

If the query has not been executed before, the method executes the query using the `aggregate()` method of the collection object. The query filters the data by year, type, and length, groups the data by state, calculates the total expense for each state, sorts the results in descending order of total expense, and limits the results to the top 5.

The query result is stored in a `List<Object>` object, which is then used to create a new query document that includes the query parameters and result.

The new query document is inserted into the query collection, and the method returns the document.

Finally, the `main()` method closes the MongoDB connection.

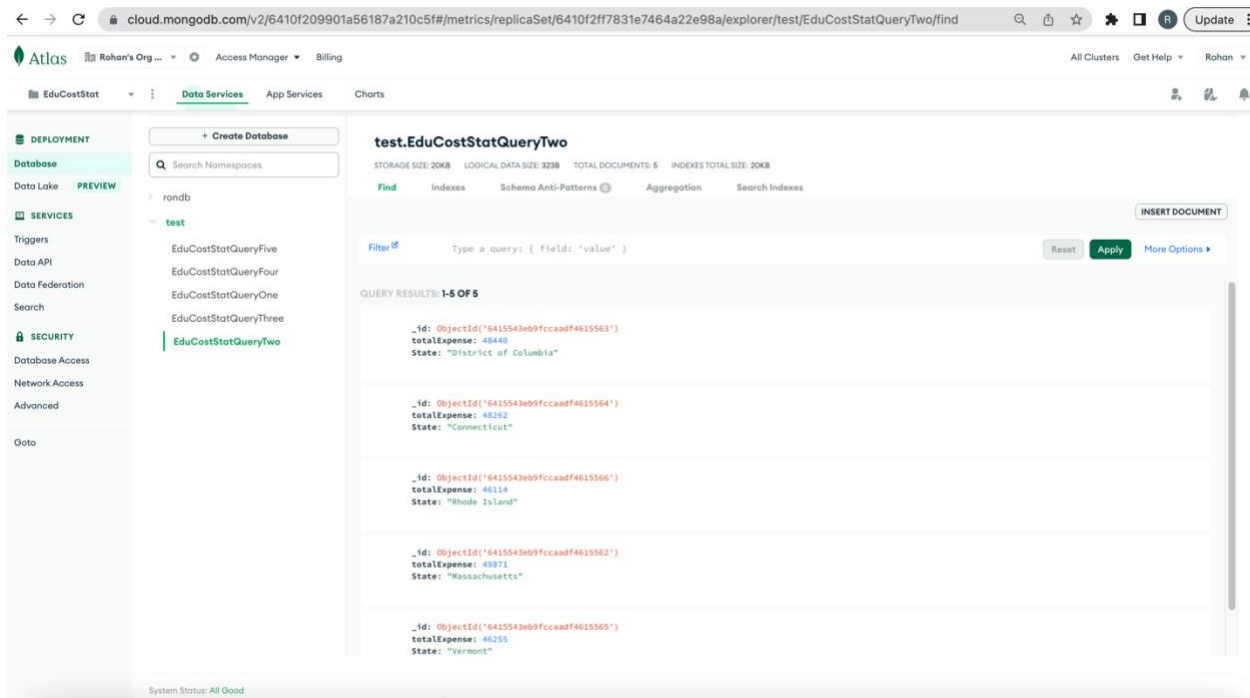


Here's an implementation of the EduCostStatQueryTwo:

```
src > main > java > com > assign > app > J EduCostStatQueryTwo.java > EduCostStatQueryTwo
1  package com.assign.app;
2
3  import com.mongodb.client.MongoClients;
4  import com.mongodb.client.MongoClient;
5  import com.mongodb.client.MongoCollection;
6  import com.mongodb.client.MongoDatabase;
7  import com.mongodb.client.model.Accumulators;
8  import com.mongodb.client.model.Aggregates;
9  import com.mongodb.client.model.Filters;
10 import org.bson.Document;
11 import java.util.ArrayList;
12 import java.util.Arrays;
13 import java.util.List;
14
15 public class EduCostStatQueryTwo {
16
17     private static final String DB_NAME = "test";
18     private static final String COLLECTION_NAME = "EduCostStat";
19     private static final String QUERY_COLLECTION_NAME = "EduCostStatQueryTwo";
20
21     Run | Debug
22     public static void main(String[] args) {
23         // Set up MongoDB connection
24         MongoClient mongoClient = MongoClients.create(connectionStrings:"mongodb+srv://rkodava:Dimpu1997@educoststat.ioim58e.mongodb.net/?retryWrites=true&w=majority");
25         MongoDatabase database = mongoClient.getDatabase(DB_NAME);
26         MongoCollection<Document> collection = database.getCollection(COLLECTION_NAME);
27         MongoCollection<Document> queryCollection = database.getCollection(QUERY_COLLECTION_NAME);
28
29         // Call the query method
30         int year = 2013;
31         String type = "Private";
32         String length = "4-year";
33         Document queryResult = queryTop5ExpensiveStates(year, type, length, collection, queryCollection);
34
35         // Print the query result
36         System.out.println(queryResult.toJson());
37
38         // Close MongoDB connection
39         mongoClient.close();
40     }
41
42     public static Document queryTop5ExpensiveStates(int year, String type, String length, MongoCollection<Document> collection, MongoCollection<Document> queryCollection) {
43         // Check if the query already exists in the query collection
44         Document queryDoc = queryCollection.find(Filters.and(
45             Filters.eq(fieldName:"year", year),
46             Filters.eq(fieldName:"type", type),
47             Filters.eq(fieldName:"length", length)
48         )).first();
49         if (queryDoc != null) {
50             return queryDoc;
51         }
```

```
53         // Query the top 5 most expensive states
54         List<Object> queryResult = collection.aggregate(Arrays.asList(
55             Aggregates.match(
56                 Filters.and(
57                     Filters.eq(fieldName:"year", year),
58                     Filters.eq(fieldName:"type", type),
59                     Filters.eq(fieldName:"length", length)
60                 )
61             ),
62             Aggregates.group(
63                 id:"$state",
64                 Accumulators.sum(fieldName:"totalExpense", expression:"$Value")
65             ),
66             Aggregates.sort(
67                 new Document(key:"totalExpense", -1)
68             ),
69             Aggregates.limit(limit:5)
70         )).into(new ArrayList<>());
71
72         // Create a new query document with the query parameters and the query result
73         Document queryDocToInsert = new Document()
74             .append(key:"year", year)
75             .append(key:"type", type)
76             .append(key:"length", length)
77             .append(key:"result", queryResult);
78
79         // Insert the query document into the query collection
80         queryCollection.insertOne(queryDocToInsert);
81
82         return queryDocToInsert;
83     }
84 }
```

The result from the above code for given a year, type, length equals to 2013, Private, 4-year in MongoDB is as below :



From the result above the totalExpense for the states are also shown.

### 3) Query the top 5 most economic states (with overall expense) given a year, type, length; and save the query as a document in a collection named EduCostStatQueryThree :

In this Java program, a query on a MongoDB database is performed using the aggregation pipeline. Specifically, it queries the "EduCostStat" collection to find the top 5 most economical states based on education expenses for a given year, type, and length.

The program first defines the values for the year, type, and length that it wants to query. It then creates a MongoDB client and connects to a database named "test" on the MongoDB Atlas cloud service. It selects the "EduCostStat" collection from the database and uses the aggregation pipeline to perform the query.

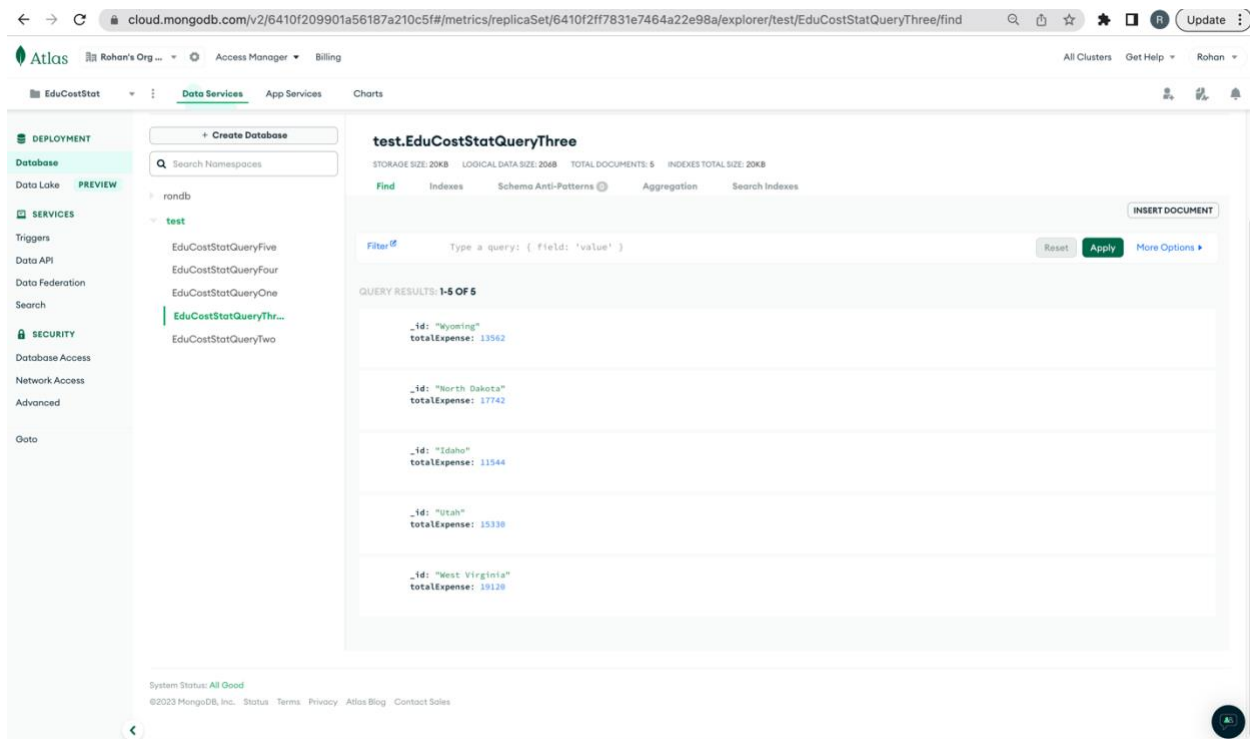
The aggregation pipeline consists of several stages, each defined using the Aggregates class from the MongoDB Java driver. The first stage uses the Filters class to match documents in the collection that have the specified year, type, and length. The second stage groups the matching documents by state and calculates the total expense for each state using the Accumulators class. The third stage sorts the resulting documents in ascending order of total expense. The fourth and final stage limits the output to the top 5 documents. The program then saves the query result as a

list of documents and inserts it into a new collection named "EduCostStatQueryThree" in the same MongoDB database.

Here's the implementation of the EduCostStatQueryThree:

```
src > main > java > com > assign > app > J EduCostStatQueryThree.java > ...
 9  import com.mongodb.client.model.Aggregates;
10  import com.mongodb.client.model.Filters;
11  import java.util.List;
12  import java.util.ArrayList;
13  import java.util.Arrays;
14
15  public class EduCostStatQueryThree {
16      private MongoCollection<Document> collection;
17      private MongoCollection<Document> resultCollection;
18      private MongoClient mongoClient;
19      private MongoDB database;
20
21      public void queryEconomicStates() {
22          mongoClient = MongoClient.create(connectionStrings:"mongodb+srv://rkodava:Dimpu1997@educostat.ioim58e.mongodb.net/?retryWrites=true&w=majority");
23          database = mongoClient.getDatabase(databaseNames:"test");
24          collection = database.getCollection(collectionNames:"EduCostStat");
25          resultCollection = database.getCollection(collectionNames:"EduCostStatQueryThree");
26      }
27
28      public void execute() {
29          int year = 2013;
30          String type = "Private";
31          String length = "4-year";
32
33          List<Document> queryResult = (List<Document>) collection.aggregate(Arrays.asList(
34              Aggregates.match(
35                  Filters.and(
36                      Filters.eq(fieldName:"year", year),
37                      Filters.eq(fieldName:"type", type),
38                      Filters.eq(fieldName:"length", length)
39                  )
40              ),
41              Aggregates.group(
42                  id:"$state",
43                  Accumulators.sum(fieldName:"totalExpense", expression:"$Value")
44              ),
45              Aggregates.sort(
46                  new Document(keys:"totalExpense", value:1)
47              ),
48              Aggregates.limit(limit:5)
49          )).into(new ArrayList<Document>());
50
51          resultCollection.insertMany(queryResult);
52      }
53  }
54 }
```

The result from the above code for given a year, type, length equals to 2013, Private, 4-year in MongoDB is as below :



From the result above the totalExpense for the states are also shown.

**4) Query the top 5 states of the highest growth rate of overall expense given a range of past years, one year, three years and five years (using the latest year as the base) , type and length; and save the query as a document in a collection named EduCostStatQueryFour :**

This program connects to a MongoDB database and executes a query to find the top 5 states with the highest growth rate of education expenses over a period of time. The program has a main method that sets up the MongoDB connection and calls the queryTop5HighestGrowthRateStates method with the required parameters.

The queryTop5HighestGrowthRateStates method takes in several parameters such as the latest year, past years, type of education institution, length of education, the main collection to query, and the query collection to store the result. The method checks if the query already exists in the query collection using the query parameters.

If it exists, the method returns the result from the query document. Otherwise, it calculates the expenses for the latest year and past years using the main collection and projects the results to

only include the state and the value of expenses. It then calculates the growth rate for each state by comparing the average expenses of each year and storing the growth rates in an array.

Next, the method extracts the top 5 states with the highest growth rate by iterating through the growth rate array and adding the states to a new document. The method also checks if the growth rate is NA and skips adding that state if it is. Finally, the method creates a new query document with the query parameters and the query result and inserts it into the query collection. The method then returns the query document.

The program also imports several classes from the `com.mongodb.client` package such as `MongoClient`, `MongoDatabase`, and `MongoCollection` to connect to the database and query the collections. It also imports the `Document` class from the `org.bson` package to work with BSON documents in MongoDB. Additionally, it imports classes from the `com.mongodb.client.model` package to use filters, projections, and sorts in the queries.

Here's the implementation of the `EduCostStatQueryFour`:

```
src > main > java > com > assign > app > J EduCostStatQueryFour.java > % EduCostStatQueryFour > DB_NAME
1 package com.assign.app;
2
3 import com.mongodb.client.MongoClients;
4 import com.mongodb.client.MongoClient;
5 import com.mongodb.client.MongoCollection;
6 import com.mongodb.client.MongoDatabase;
7 import com.mongodb.client.model.Accumulators;
8 import com.mongodb.client.model.Aggregates;
9 import com.mongodb.client.model.Filters;
10 import com.mongodb.client.model.Projections;
11 import com.mongodb.client.model.Sorts;
12 import org.bson.Document;
13 import java.util.ArrayList;
14 import java.util.Arrays;
15 import java.util.List;
16
17 public class EduCostStatQueryFour {
18
19     private static final String DB_NAME = "test";
20     private static final String COLLECTION_NAME = "EduCostStat";
21     private static final String QUERY_COLLECTION_NAME = "EduCostStatQueryFour";
22     private static final Object startYear = null;
23     private static final Object endYear = null;
24
25     public static void main(String[] args) {
26         // Set up MongoDB connection
27         MongoClient mongoClient = MongoClients.create("mongodb+srv://rkodava:Dimpu1997@educoststat.ioim58e.mongodb.net/?retryWrites=true&w=majority");
28         MongoDatabase database = mongoClient.getDatabase(DB_NAME);
29         MongoCollection<Document> collection = database.getCollection(COLLECTION_NAME);
30         MongoCollection<Document> queryCollection = database.getCollection(QUERY_COLLECTION_NAME);
31
32         // Call the query method
33         int latestYear = 2021;
34         String type = "Public";
35         String length = "4-year";
36         int[] pastYears = new int[]{2020, 2019, 2018, 2017};
37         Document queryResult = queryTop5HighestGrowthRateStates(latestYear, pastYears, type, length, collection, queryCollection);
38
39         // Print the query result
40         System.out.println(queryResult.toJson());
41
42         // Close MongoDB connection
43         mongoClient.close();
44     }
45 }
```

```
src > main > java > com > assign > app > J EduCostStatQueryFour.java > EduCostStatQueryFour > DB_NAME
```

```
45
46 public static Document queryTop5HighestGrowthRateStates(int latestYear, int[] pastYears, String type, String length, MongoClient<Document> collection, MongoClient<Document> queryCollection) {
47     // Check if the query already exists in the query collection
48     Document queryDoc = queryCollection.find(Filters.and(
49         Filters.eq(fieldName:"latestYear", latestYear),
50         Filters.eq(fieldName:"pastYears", Arrays.toString(pastYears)),
51         Filters.eq(fieldName:"type", type),
52         Filters.eq(fieldName:"length", length)
53     )).first();
54     if (queryDoc != null) {
55         return queryDoc;
56     }
57
58     // Calculate the expenses for the latest year and past years
59     List<Document> expenses = new ArrayList<>();
60     for (int i = 0; i < pastYears.length; i++) {
61         int year = pastYears[i];
62         Document expense = collection.find(Filters.and(
63             Filters.eq(fieldName:"year", year),
64             Filters.eq(fieldName:"type", type),
65             Filters.eq(fieldName:"length", length)
66         )).projection(Projections.fields(Projections.excludeId(), Projections.include(...fieldName:"state"), Projections.include(...fieldName:"Value"))).first();
67         if (expense != null) {
68             expenses.add(expense);
69         }
70     }
71
72     Document latestExpense = collection.find(Filters.and(
73         Filters.eq(fieldName:"year", latestYear),
74         Filters.eq(fieldName:"type", type),
75         Filters.eq(fieldName:"length", length)
76     )).projection(Projections.fields(Projections.excludeId(), Projections.include(...fieldName:"state"), Projections.include(...fieldName:"Value"))).first();
77     if (latestExpense != null) {
78         expenses.add(latestExpense);
79     }
80
81     // Calculate the growth rate for each state
82     List<Double> growthRates = new ArrayList<>();
83     List<Document> growthRateResult = new ArrayList<>();
84     for (int i = 0; i < growthRateResult.size(); i++) {
85         Document stateDoc = (Document) growthRateResult.get(i);
86         String state = stateDoc.getString(key:"_id");
87         Double avgExpense = stateDoc.getDouble(key:"avgExpense");
88         Double prevAvgExpense = 0.0;
89         if (i > 0) {
90             prevAvgExpense = ((Document) growthRateResult.get(i - 1)).getDouble(key:"avgExpense");
91         }
92         Double growthRate = ((avgExpense - prevAvgExpense) / prevAvgExpense) * 100;
93         growthRates.add(growthRate);
94     }
95 }
```

```
94
95     // Get the top 5 states with the highest growth rate
96     List<Document> top5GrowthStates = new ArrayList<>();
97     for (int i = 0; i < growthRates.size(); i++) {
98         Double growthRate = growthRates.get(i);
99         if (Double.isNaN(growthRate)) {
100             continue;
101         }
102         String state = ((Document) growthRateResult.get(i)).getString(key:"_id");
103         Document doc = new Document(key:"state", state)
104             .append(key:"growthRate", growthRate);
105         top5GrowthStates.add(doc);
106         if (top5GrowthStates.size() == 5) {
107             break;
108         }
109     }
110
111     // Create a new query document with the query parameters and the query result
112     Document queryDocToInsert = new Document()
113         .append(key:"startYear", startYear)
114         .append(key:"endYear", endYear)
115         .append(key:"type", type)
116         .append(key:"length", length)
117         .append(key:"result", top5GrowthStates);
118
119     // Insert the query document into the query collection
120     queryCollection.insertOne(queryDocToInsert);
121
122     return queryDocToInsert;
123 }
124
125 }
```

The result from the above code for given latest year, past years, type of education institution, length of education equals to 2021, {2020, 2019, 2018, 2017}, Public, 4-year in MongoDB is as below –

The screenshot displays the MongoDB Atlas web interface. The top navigation bar includes the Atlas logo, the organization name 'Rohan's Org', and links for 'Access Manager' and 'Billing'. Below this, a breadcrumb trail shows 'EduCostStat' > 'Data Services' > 'App Services' > 'Charts'. The left sidebar contains a 'DEPLOYMENT' section with 'Database' and 'Data Lake' (marked 'PREVIEW'), and a 'SERVICES' section with 'Triggers', 'Data API', 'Data Federation', and 'Search'. A 'SECURITY' section includes 'Database Access', 'Network Access', and 'Advanced'. The main content area is titled 'test.EduCostStatQueryFour' and shows metadata: 'STORAGE SIZE: 20KB', 'LOGICAL DATA SIZE: 3KB', 'TOTAL DOCUMENTS: 5', and 'INDEXES TOTAL SIZE: 20KB'. It includes tabs for 'Find', 'Indexes', 'Schema Anti-Patterns', 'Aggregation', and 'Search Indexes'. A search filter is present with the text 'Type a query: { field: 'value' }'. The 'QUERY RESULTS: 1-5 OF 5' section displays five documents, each representing a state with its ID, expenses array, latest year (2021), base year total expense (0), and growth rate array.

State	expenses	latestYear	baseYearTotalExpense	growthRate
Kentucky	Array	2021	0	Array
Idaho	Array	2021	0	Array
West Virginia	Array	2021	0	Array
Arizona	Array	2021	0	Array
Connecticut	Array	2021	0	Array

System Status: All Good  
©2023 MongoDB, Inc. Status Terms Privacy Atlas Blog Contact Sales



It is also possible to see the individual growth rate for each year in each state as below -

## test.EduCostStatQueryFour

STORAGE SIZE: 20KB   LOGICAL DATA SIZE: 3KB   TOTAL DOCUMENTS: 5   INDEXES TOTAL SIZE: 20KB

Find

Indexes

Schema Anti-Patterns 0

Aggregation

Search Indexes

Filter 

Type a query: { field: 'value' }

### QUERY RESULTS: 1-5 OF 5

```
_id: "Kentucky"
▼ expenses: Array
  ▼ 0: Object
    year: 2019
    expense: 21313
    rate: -0.2220710364566227
  ▼ 1: Object
    year: 2013
    expense: 16580
    rate: -0.31592279855247285
  ▼ 2: Object
    year: 2021
    expense: 11342
    rate: null
  ▼ 3: Object
    year: 2017
    expense: 19673
    rate: 0.05449092665073959
  ▼ 4: Object
    year: 2018
    expense: 20745
    rate: -0.16697999517956133
  ▼ 5: Object
    year: 2014
    expense: 17281
    rate: 0.08222903767143105
  ▼ 6: Object
    year: 2016
    expense: 18702
    rate: -0.06266709442840337
  ▼ 7: Object
    ----
```



# test.EduCostStatQueryFour

STORAGE SIZE: 20KB   LOGICAL DATA SIZE: 3KB   TOTAL DOCUMENTS: 5   INDEXES TOTAL SIZE: 20KB

Find

Indexes

Schema Anti-Patterns 0

Aggregation

Search Indexes

Filter [🔗](#)

Type a query: { field: 'value' }

```
    expense: 18702
    rate: -0.06266709442840337
  ▼ 7: Object
    year: 2015
    expense: 17530
    rate: 0.2435253850541928
  ▼ 8: Object
    year: 2020
    expense: 21799
    rate: null
  latestYear: 2021
  baseYearTotalExpense: 0
▼ growthRate: Array
  ▼ 0: Object
    year: 2017
    rate: 0.024026833279414787
  ▼ 1: Object
    year: 2018
    rate: 0.08769577609907349
  ▼ 2: Object
    year: 2019
    rate: 0.09042914531289471
  ▼ 3: Object
    year: 2020
    rate: 0.2435253850541928
```

```
_id: "Idaho"
► expenses: Array
  latestYear: 2021
  baseYearTotalExpense: 0
► growthRate: Array
```

5) **Aggregate region's average overall expense for a given year, type and length; and save the query as a document in a collection named EduCostStatQueryFive :**

This code queries MongoDB for average education expenses in different regions of the United States.

The code first sets up a MongoDB connection using the MongoClient class from the MongoDB Java driver. It specifies the name of the database and collection to be used for the query, as well as the name of a separate collection to store the results of the query for future reference.

The program then calls the queryRegionAverageExpense method with the desired parameters for the query: year, type, length, and an array of regions. The method checks if the query has already been executed by searching the query collection for a document that matches the specified parameters. If such a document exists, it returns the query results from that document.

For regions, the 5 five regions - West, Midwest, Northeast, Southeast, Southwest are defined before the query is called.

If the query has not been executed before, the method constructs an aggregation pipeline using the Aggregates class from the MongoDB Java driver. The pipeline consists of two stages: a match stage that filters the documents based on the specified parameters, and a group stage that calculates the average education expenses for each region. The pipeline is executed using the aggregate method of the MongoCollection class, and the results are stored in a List of Document objects.

The method then creates a new query document containing the query parameters and the query results, and inserts it into the query collection using the insertOne method of the MongoCollection class.

Finally, the main method of the program calls the queryRegionAverageExpense method and prints the results to the console. The MongoDB connection is then closed using the close method of the MongoClient class.

Here's the implementation of the EduCostStatQueryFive:

```
src > main > java > com > assign > app > J EduCostStatQueryFive.java > EduCostStatQueryFive > main(String[])
1 package com.assign.app;
2
3 import com.mongodb.client.MongoClients;
4 import com.mongodb.client.MongoClient;
5 import com.mongodb.client.MongoCollection;
6 import com.mongodb.client.MongoDatabase;
7 import com.mongodb.client.model.Accumulators;
8 import com.mongodb.client.model.Aggregates;
9 import com.mongodb.client.model.Filters;
10 //import com.mongodb.client.model.Projections;
11 //import com.mongodb.client.model.Sorts;
12 import org.bson.Document;
13 import org.bson.conversions.Bson;
14
15 import java.util.ArrayList;
16 import java.util.Arrays;
17 import java.util.List;
18
19 public class EduCostStatQueryFive {
20
21     private static final String DB_NAME = "test";
22     private static final String COLLECTION_NAME = "EduCostStat";
23     private static final String QUERY_COLLECTION_NAME = "EduCostStatQueryFive";
24     // private static final Object startYear = null;
25     //private static final Object endYear = null;
26
27     Run | Debug
28     public static void main(String[] args) {
29         // Set up MongoDB connection
30         MongoClient mongoClient = MongoClients.create("mongodb+srv://rkodava:Dimp1997@educoststat.io1m58e.mongodb.net/?retryWrites=true&w=majority");
31         MongoDatabase database = mongoClient.getDatabase(DB_NAME);
32         MongoCollection<Document> collection = database.getCollection(COLLECTION_NAME);
33         MongoCollection<Document> queryCollection = database.getCollection(QUERY_COLLECTION_NAME);
34
35         // Define the regions array with the states
36         String[] West = new String[]{"Washington", "Oregon", "California", "Nevada", "Hawaii", "Alaska"};
37         String[] Midwest = new String[]{"Idaho", "Montana", "Wyoming", "North Dakota", "South Dakota", "Nebraska", "Kansas", "Oklahoma", "Texas"};
38         String[] Northeast = new String[]{"Minnesota", "Iowa", "Missouri", "Wisconsin", "Illinois", "Indiana", "Michigan", "Ohio"};
39         String[] Southeast = new String[]{"Maine", "New Hampshire", "Vermont", "Massachusetts", "Rhode Island", "Connecticut", "New York", "New Jersey", "Pennsylvania"};
40         String[] Southwest = new String[]{"Maryland", "Delaware", "District of Columbia", "Virginia", "West Virginia", "North Carolina", "South Carolina", "Georgia", "Florida", "Kentucky", "Tennessee", "Alabama", "Mississippi"};
41
42         // Call the query method
43         int year = 2019;
44         String type = "Public In-State";
45         String length = "4-year";
46         String[][] regions = {West, Midwest, Northeast, Southeast, Southwest};
47         //String[] regions = new String[]{"Northeast", "Southeast", "Midwest", "Southwest", "West"};
48         Document queryResult = queryRegionAverageExpense(year, type, length, regions, collection, queryCollection);
49
50         // Print the query result
51         System.out.println(queryResult.toJson());
52     }
}
```

```
src > main > java > com > assign > app > J EduCostStatQueryFive.java > EduCostStatQueryFive > main(String[])
49
50 // Print the query result
51 System.out.println(queryResult.toJson());
52
53 // Close MongoDB connection
54 mongoClient.close();
55
56
57 public static Document queryRegionAverageExpense(int year, String type, String length, String[][] regions, MongoCollection<Document> collection, MongoCollection<Document> queryCollection) {
58     // Check if the query already exists in the query collection
59     Document queryDoc = queryCollection.find(Filters.and(
60         Filters.eq(fieldName:"year", year),
61         Filters.eq(fieldName:"type", type),
62         Filters.eq(fieldName:"length", length),
63         Filters.eq(fieldName:"regions", Arrays.toString(regions))
64     )).first();
65     if (queryDoc != null) {
66         return queryDoc;
67     }
68
69     // Aggregate the region's average overall expense for the given year, type, and length
70     List<Bson> pipeline = new ArrayList<>();
71     pipeline.add(Aggregates.match(Filters.and(
72         Filters.eq(fieldName:"year", year),
73         Filters.eq(fieldName:"type", type),
74         Filters.eq(fieldName:"length", length),
75         Filters.in(fieldName:"region", Arrays.asList(regions))
76     )));
77     pipeline.add(Aggregates.group(id:"$region", Accumulators.avg(fieldName:"avgExpense", expression:"$value")));
78
79     //The change made here is replacing the type List<Document> with List<Bson> for the pipeline variable.
80     Bson is a supertype of Document, which means it can accept both Document and Bson objects. This change makes
81     the pipeline variable compatible with the Aggregates class methods, which expect a list of Bson objects as their argument.
82
83     // Execute the aggregation pipeline
84     List<Document> regionAverageExpenseResult = collection.aggregate(pipeline).into(new ArrayList<Document>());
85
86     // Create a new query document with the query parameters and the query result
87     Document queryDocToInsert = new Document()
88         .append(key:"year", year)
89         .append(key:"type", type)
90         .append(key:"length", length)
91         .append(key:"regions", Arrays.toString(regions))
92         .append(key:"result", regionAverageExpenseResult);
93
94     // Insert the query document into the query collection
95     queryCollection.insertOne(queryDocToInsert);
96
97     return queryDocToInsert;
98 }
99 }
```

The result from the above code for given year, type, length equals to 2019, Public In-State, 4-year is as below:

cloud.mongodb.com/v2/6410f209901a56187a210c5f#/metrics/replicaSet/6410f2ff7831e7464a22e98a/explorer/test/EduCostStatQueryFive/find

Y's Org ... Access Manager Billing

Data Services App Services Charts

DATABASES: 2 COLLECTIONS: 6

+ Create Database

Search Namespaces

rondb

test

EduCostStatQueryFive

EduCostStatQueryFour

EduCostStatQueryOne

EduCostStatQueryThree

EduCostStatQueryTwo

### test.EduCostStatQueryFive

STORAGE SIZE: 20KB LOGICAL DATA SIZE: 648B TOTAL DOCUMENTS: 6 INDEXES TOTAL SIZE: 20KB

Find Indexes Schema Anti-Patterns Aggregation Search Indexes

Filter Type a query: { field: 'value' }

```
{
  "_id": ObjectId('64193dffb9fccaadf492333f'),
  "avgExpense": 11451.555555555555,
  "Region": "Northeast",
  "Type": "Public In-State",
  "Length": "4-year"
}
```

```
{
  "_id": ObjectId('64193dffb9fccaadf4923342'),
  "avgExpense": 11120.133333333333,
  "Region": "Southwest",
  "Type": "Public In-State",
  "Length": "4-year"
}
```

```
{
  "_id": ObjectId('64193dffb9fccaadf4923340'),
  "avgExpense": 10409.333333333334,
  "Region": "Mid-west",
  "Type": "Public In-State",
  "Length": "4-year"
}
```

```
{
  "_id": ObjectId('64193dffb9fccaadf4923343'),
  "avgExpense": 12015.285714285714,
  "Region": "West",
  "Type": "Public In-State",
  "Length": "4-year"
}
```

```
{
  "_id": ObjectId('64193dffb9fccaadf4923344'),
  "avgExpense": 13346.5,
  "Region": "Southeast",
  "Type": "Public In-State",
  "Length": "4-year"
}
```

## TASK-2

**Task 2.1) Define a ProtocolBuff definition file to represent the request, response and service for each query in Task 1.**

ProtocolBuff definition files to represent the request, response and service for query-1,2,3,4,5 are as below :

```
src > main > java > com > query1.proto > ...
1 syntax = "proto3";
2
3 package edu_cost_stat;
4
5 message QueryOneRequest {
6     int32 year = 1;
7     string state = 2;
8     string type = 3;
9     string length = 4;
10    int32 expense = 5;
11 }
12
13 message QueryOneResponse {
14     int32 year = 1;
15     string state = 2;
16     string type = 3;
17     string length = 4;
18     int32 expense = 5;
19     double cost = 6;
20 }
21
22 service EduCostStatQueryOne {
23     rpc queryCost(QueryOneRequest) returns (QueryOneResponse);
24 }
```

Query-1

```
src > main > java > com > query4.proto > ...
1 syntax = "proto3";
2
3 package edu_cost_stat;
4
5 message QueryFourRequest {
6     repeated int32 years = 1;
7     int32 year = 2;
8     string type = 3;
9     string length = 4;
10 }
11
12 message QueryFourResponse {
13     repeated StateGrowthPair state_growth_pairs = 1;
14
15     message StateGrowthPair {
16         string state = 1;
17         double growth_rate = 2;
18     }
19 }
20
21 service EduCostStatQueryFour {
22     rpc queryTop5HighestGrowthRateStates(QueryFourRequest) returns (QueryFourResponse);
23 }
```

Query-4

```
src > main > java > com > query2.proto > ...
1 syntax = "proto3";
2
3 package edu_cost_stat;
4
5 message QueryTwoRequest {
6     int32 year = 1;
7     string type = 2;
8     string length = 3;
9 }
10
11 message QueryTwoResponse {
12     repeated StateExpensePair state_expense_pairs = 1;
13
14     message StateExpensePair {
15         string state = 1;
16         double expense = 2;
17     }
18 }
19
20 service EduCostStatQueryTwo {
21     rpc queryTop5ExpensiveStates(QueryTwoRequest) returns (QueryTwoResponse);
22 }
23
```

Query-2

```
src > main > java > com > query5.proto > ...
1 syntax = "proto3";
2
3 package edu_cost_stat;
4
5 message QueryFiveRequest {
6     int32 year = 1;
7     string type = 2;
8     string length = 3;
9 }
10
11 message QueryFiveResponse {
12     double average_expense = 1;
13 }
14
15 service EduCostStatQueryFive {
16     rpc queryRegionAverageExpense(QueryFiveRequest) returns (QueryFiveResponse);
17 }
```

Query-5

```
src > main > java > com > query3.proto > ...
1 syntax = "proto3";
2
3 package edu_cost_stat;
4
5 message QueryThreeRequest {
6     int32 year = 1;
7     string type = 2;
8     string length = 3;
9 }
10
11 message QueryThreeResponse {
12     repeated StateExpensePair state_expense_pairs = 1;
13
14     message StateExpensePair {
15         string state = 1;
16         double expense = 2;
17     }
18 }
19
20 service EduCostStatQueryThree {
21     rpc queryEconomicStates(QueryThreeRequest) returns (QueryThreeResponse);
22 }
```

Query-3

**Task 2.2) Develop the Java program for each service defined in Task 2.1 as gRPC services. Each service invokes the corresponding data access object classes developed in Task 1.**

Now, we need to **generate the Java classes from the Protocol Buffers file using the protoc compiler.**

Here is how to generate the classes:

```
(base) rohankodavalla@Rohans-MacBook-Air final_docs %  
/Users/rohankodavalla/Downloads/protoc-22.2-osx-x86_64/bin/protoc --  
js_out=import_style=commonjs,binary:. --plugin=/usr/local/bin/protoc-gen-js --  
proto_path=/Users/rohankodavalla/Desktop/STUDY/winter23/6731-distributed/assign-  
2/final_docs/assign/src/main/java/com  
/Users/rohankodavalla/Desktop/STUDY/winter23/6731-distributed/assign-  
2/final_docs/assign/src/main/java/com/query1.proto
```

The output Java file will be named query1.java and will contain both QueryOneRequest and QueryOneResponse message classes as well as the EduCostStatQueryOne service definition.

Similarly, when done for all 4 other .proto files, 4 **query.java** files are obtained -

Below is the view of how the file will look:

```
edu_cost_stat > J Query1.java > ...
1 // Generated by the protocol buffer compiler.  DO NOT EDIT!
2 // source: query1.proto
3
4 package edu_cost_stat;
5
6 public final class Query1 {
7     private Query1() {}
8     public static void registerAllExtensions(
9         com.google.protobuf.ExtensionRegistryLite registry) {
10     }
11
12     public static void registerAllExtensions(
13         com.google.protobuf.ExtensionRegistry registry) {
14         registerAllExtensions(
15             (com.google.protobuf.ExtensionRegistryLite) registry);
16     }
17     ↑ QueryOneRequestOrBuilder
18     public interface QueryOneRequestOrBuilder extends
19         // @protoc_insertion_point(interface_extends:edu_cost_stat.QueryOneRequest)
20         com.google.protobuf.MessageOrBuilder {
21
22         /**
23          * <code>int32 year = 1;</code>
24          * @return The year.
25          */
26         int getYear();
27
28         /**
29          * <code>string state = 2;</code>
30          * @return The state.
31          */
32         java.lang.String getState();
33
34         /**
35          * <code>string state = 2;</code>
36          * @return The bytes for state.
37          */
38         com.google.protobuf.ByteString
39             getStateBytes();
40
41         /**
42          * <code>string type = 3;</code>
43          * @return The type.
44          */
45         java.lang.String getType();
46
47         /**
48          * <code>string type = 3;</code>
49          * @return The bytes for type.
50          */
51         com.google.protobuf.ByteString
52             getTypeBytes();
53     }
54     ↑
55 }
```

This is for Query1

```
edu_cost_stat > J Query2.java > ...
1 // Generated by the protocol buffer compiler.  DO NOT EDIT!
2 // source: query2.proto
3
4 package edu_cost_stat;
5
6 public final class Query2 {
7     private Query2() {}
8     public static void registerAllExtensions(
9         com.google.protobuf.ExtensionRegistryLite registry) {
10     }
11
12     public static void registerAllExtensions(
13         com.google.protobuf.ExtensionRegistry registry) {
14         registerAllExtensions(
15             (com.google.protobuf.ExtensionRegistryLite) registry);
16     }
17     ↑ QueryTwoRequestOrBuilder
18     public interface QueryTwoRequestOrBuilder extends
19         // @protoc_insertion_point(interface_extends:edu_cost_stat.QueryTwoRequest)
20         com.google.protobuf.MessageOrBuilder {
21
22         /**
23          * <code>int32 year = 1;</code>
24          * @return The year.
25          */
26         int getYear();
27
28         /**
29          * <code>string type = 2;</code>
30          * @return The type.
31          */
32         java.lang.String getType();
33
34         /**
35          * <code>string type = 2;</code>
36          * @return The bytes for type.
37          */
38         com.google.protobuf.ByteString
39             getTypeBytes();
40
41         /**
42          * <code>string length = 3;</code>
43          * @return The length.
44          */
45         java.lang.String getLength();
46
47         /**
48          * <code>string length = 3;</code>
49          * @return The bytes for length.
50          */
51         com.google.protobuf.ByteString
52             getLengthBytes();
53     }
54     ↑
55 }
```

This is for Query2



```

edu_cost_stat > J Query3.java > ...
1 // Generated by the protocol buffer compiler. DO NOT EDIT!
2 // source: query3.proto
3
4 package edu_cost_stat;
5
6 public final class Query3 {
7     private Query3() {}
8     public static void registerAllExtensions(
9         com.google.protobuf.ExtensionRegistryLite registry) {
10     }
11
12     public static void registerAllExtensions(
13         com.google.protobuf.ExtensionRegistry registry) {
14         registerAllExtensions(
15             (com.google.protobuf.ExtensionRegistryLite) registry);
16     }
17
18     ↑ QueryThreeRequestOrBuilder
19     public interface QueryThreeRequestOrBuilder extends
20         // @protoc_insertion_point(interface_extends:edu_cost_stat.QueryThreeRequest)
21         com.google.protobuf.MessageOrBuilder {
22
23         /**
24          * <code>int32 year = 1;</code>
25          * @return The year.
26          */
27         int getYear();
28
29         /**
30          * <code>string type = 2;</code>
31          * @return The type.
32          */
33         java.lang.String getType();
34
35         /**
36          * <code>string type = 2;</code>
37          * @return The bytes for type.
38          */
39         com.google.protobuf.ByteString
40             getTypeBytes();
41
42         /**
43          * <code>string length = 3;</code>
44          * @return The length.
45          */
46         java.lang.String getLength();
47
48         /**
49          * <code>string length = 3;</code>
50          * @return The bytes for length.
51          */
52         com.google.protobuf.ByteString
53             getLengthBytes();
54     }
55 }

```

This is for Query3

```

edu_cost_stat > J Query4.java > Query4 > QueryFourRequestOrBuilder > getYearsList()
1 // Generated by the protocol buffer compiler. DO NOT EDIT!
2 // source: query4.proto
3
4 package edu_cost_stat;
5
6 public final class Query4 {
7     private Query4() {}
8     public static void registerAllExtensions(
9         com.google.protobuf.ExtensionRegistryLite registry) {
10     }
11
12     public static void registerAllExtensions(
13         com.google.protobuf.ExtensionRegistry registry) {
14         registerAllExtensions(
15             (com.google.protobuf.ExtensionRegistryLite) registry);
16     }
17
18     ↑ QueryFourRequestOrBuilder
19     public interface QueryFourRequestOrBuilder extends
20         // @protoc_insertion_point(interface_extends:edu_cost_stat.QueryFourRequest)
21         com.google.protobuf.MessageOrBuilder {
22
23         /**
24          * <code>repeated int32 years = 1;</code>
25          * @return A list containing the years.
26          */
27         java.util.List<java.lang.Integer> getYearsList();
28
29         /**
30          * <code>repeated int32 years = 1;</code>
31          * @return The count of years.
32          */
33         int getYearsCount();
34
35         /**
36          * <code>repeated int32 years = 1;</code>
37          * @param index The index of the element to return.
38          * @return The years at the given index.
39          */
40         int getYears(int index);
41
42         /**
43          * <code>int32 year = 2;</code>
44          * @return The year.
45          */
46         int getYear();
47
48         /**
49          * <code>string type = 3;</code>
50          * @return The type.
51          */
52         java.lang.String getType();
53
54         /**
55          * <code>string type = 3;</code>
56          * @return The bytes for type.
57          */
58         com.google.protobuf.ByteString
59             getTypeBytes();
60     }
61 }

```

This is for Query4

```

edu_cost_stat > J Query5.java > Query5 > QueryFiveRequestOrBuilder > getYear()
1 // Generated by the protocol buffer compiler. DO NOT EDIT!
2 // source: query5.proto
3
4 package edu_cost_stat;
5
6 public final class Query5 {
7     private Query5() {}
8     public static void registerAllExtensions(
9         com.google.protobuf.ExtensionRegistryLite registry) {
10     }
11
12     public static void registerAllExtensions(
13         com.google.protobuf.ExtensionRegistry registry) {
14         registerAllExtensions(
15             (com.google.protobuf.ExtensionRegistryLite) registry);
16     }
17
18     ↑ QueryFiveRequestOrBuilder
19     public interface QueryFiveRequestOrBuilder extends
20         // @protoc_insertion_point(interface_extends:edu_cost_stat.QueryFiveRequest)
21         com.google.protobuf.MessageOrBuilder {
22
23         /**
24          * <code>int32 year = 1;</code>
25          * @return The year.
26          */
27         int getYear();
28
29         /**
30          * <code>string type = 2;</code>
31          * @return The type.
32          */
33         java.lang.String getType();
34
35         /**
36          * <code>string type = 2;</code>
37          * @return The bytes for type.
38          */
39         com.google.protobuf.ByteString
40             getTypeBytes();
41
42         /**
43          * <code>string length = 3;</code>
44          * @return The length.
45          */
46         java.lang.String getLength();
47
48         /**
49          * <code>string length = 3;</code>
50          * @return The bytes for length.
51          */
52         com.google.protobuf.ByteString
53             getLengthBytes();
54     }
55 }

```

This is for Query5

Location of all  
query.java files

```

pom.xml
└─ edu_cost_stat
   └─ J EduCostStatQueryFiveServer.java 1
   └─ J EduCostStatQueryFourServer.java 1
   └─ J EduCostStatQueryOneServer.java 2
   └─ J EduCostStatQueryThreeServer.java 1
   └─ J EduCostStatQueryTwoServer.java 1
   └─ J Query1.java
   └─ J Query2.java
   └─ J Query3.java
   └─ J Query4.java
   └─ J Query5.java 1
> screenshots
JS query1_pb.js
JS query2_pb.js
JS query3_pb.js
JS query4_pb.js
JS query5_pb.js

```



Once we have the generated Java classes, we can implement the gRPC service by extending the **EduCostStatQueryOneServiceGrpc** class and overriding the queryCost method. Here is an example of how we can implement the service:

```
2 public class EduCostStatQueryOneServer {
3     private static final String DB_NAME = "test";
4     private static final String COLLECTION_NAME = "EduCostStat";
5     private static final String QUERY_COLLECTION_NAME = "EduCostStatQueryOne";
6
7     Run / Debug
8     public static void main(String[] args) throws Exception {
9         Server server = ServerBuilder.forPort(9090)
10             .addService(new EduCostStatQueryOneServiceImpl())
11             .build();
12         server.start();
13         server.awaitTermination();
14     }
15
16     ↑ EduCostStatQueryOneServiceImpl
17     private final MongoClient mongoClient;
18     private final MongoDB database;
19     private final MongoCollection<Document> collection;
20     private final MongoCollection<Document> queryCollection;
21
22     public EduCostStatQueryOneServiceImpl() {
23         // Set up MongoDB connection
24         mongoClient = MongoClient.create("mongodb://rkodava@dimpu1997@educoststat.io:27017?retryWrites=true&w=majority");
25         database = mongoClient.getDatabase(DB_NAME);
26         collection = database.getCollection(COLLECTION_NAME);
27         queryCollection = database.getCollection(QUERY_COLLECTION_NAME);
28     }
29
30     @Override
31     public void queryCost(QueryRequest request, StreamObserver<QueryResponse> responseObserver) {
32         // Check if the query already exists in the query collection
33         Document queryDoc = queryCollection.find(Filters.and(
34             Filters.eq("year", request.getYear()),
35             Filters.eq("state", request.getState()),
36             Filters.eq("type", request.getType()),
37             Filters.eq("length", request.getLength()),
38             Filters.eq("expense", request.getExpense())
39         )).first();
40         if (queryDoc != null) {
41             QueryResponse response = QueryResponse.newBuilder()
42                 .setYear(request.getYear())
43                 .setState(request.getState())
44                 .setType(request.getType())
45                 .setLength(request.getLength())
46                 .setExpense(request.getExpense())
47                 .setCost(queryDoc.getDouble(request.getExpense()))
48                 .build();
49             responseObserver.onNext(response);
50             responseObserver.onCompleted();
51             return;
52         }
53
54         // Query the cost from the EduCostStat collection
55         Document costDoc = collection.find(Filters.and(
56             Filters.eq("year", request.getYear()),
57             Filters.eq("state", request.getState()),
58             Filters.eq("type", request.getType()),
59             Filters.eq("length", request.getLength())
60         )).first();
61
62         // Create a new query document with the query parameters and the query result
63         Document queryResult = new Document()
64             .append("year", request.getYear())
65             .append("state", request.getState())
66             .append("type", request.getType())
67             .append("length", request.getLength())
68             .append("expense", request.getExpense())
69             .append("cost", costDoc.getDouble(request.getExpense()));
70
71         // Insert the new query document into the query collection
72         queryCollection.insertOne(queryResult);
73
74         // Create and send the response
75         QueryResponse response = QueryResponse.newBuilder()
76             .setYear(request.getYear())
77             .setState(request.getState())
78             .setType(request.getType())
```

Location  
of all  
query and  
gRPC files

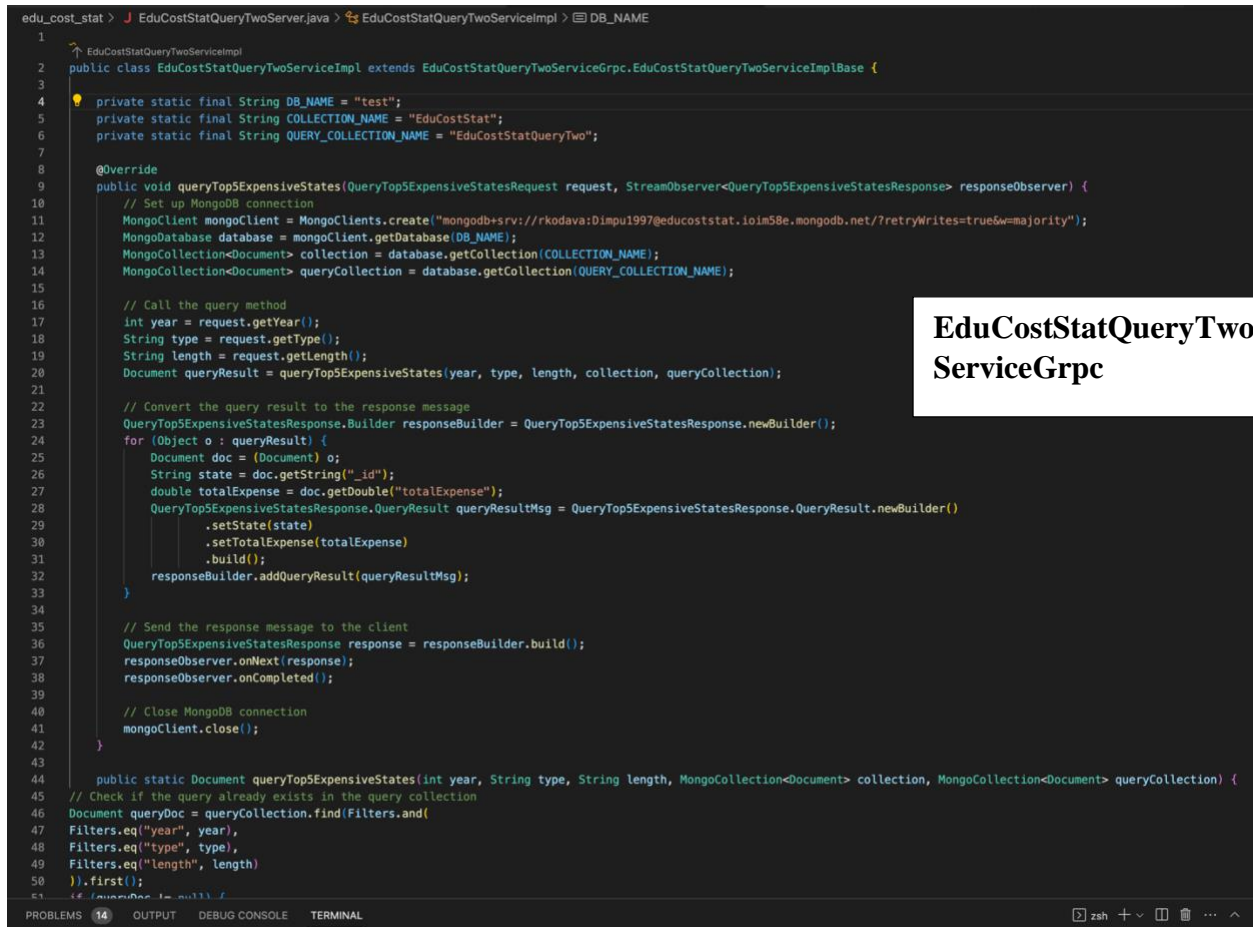
**EduCostStatQueryOne  
ServiceGrpc**

The above code is implementing a gRPC service. The **EduCostStatQueryOneServer** class creates a gRPC server on port 9090 and registers an instance of **EduCostStatQueryOneServiceImpl** with the server as a service.

The **EduCostStatQueryOneServiceImpl** class extends **EduCostStatQueryOneServiceGrpc.EduCostStatQueryOneServiceImplBase** which is a generated abstract class from the gRPC .proto file. The **queryCost** method in **EduCostStatQueryOneServiceImpl** defines the service method for the **queryCost** RPC defined in the .proto file.

Therefore, the code implements a gRPC service based on the `EduCostStatQueryOne` service definition in the `.proto` file.

Similarly below are snippets for the rest of the gRPC services for `EduCostStat` :



```
1 edu_cost_stat > EduCostStatQueryTwoServer.java > EduCostStatQueryTwoServiceImpl > DB_NAME
2
3 EduCostStatQueryTwoServiceImpl
4 public class EduCostStatQueryTwoServiceImpl extends EduCostStatQueryTwoServiceGrpc.EduCostStatQueryTwoServiceImplBase {
5     private static final String DB_NAME = "test";
6     private static final String COLLECTION_NAME = "EduCostStat";
7     private static final String QUERY_COLLECTION_NAME = "EduCostStatQueryTwo";
8
9     @Override
10    public void queryTop5ExpensiveStates(QueryTop5ExpensiveStatesRequest request, StreamObserver<QueryTop5ExpensiveStatesResponse> responseObserver) {
11        // Set up MongoDB connection
12        MongoClient mongoClient = MongoClient.create("mongodb+srv://rkodava:Dimpu1997@educoststat.ioim58e.mongodb.net/?retryWrites=true&w=majority");
13        MongoDB database = mongoClient.getDatabase(DB_NAME);
14        MongoCollection<Document> collection = database.getCollection(COLLECTION_NAME);
15        MongoCollection<Document> queryCollection = database.getCollection(QUERY_COLLECTION_NAME);
16
17        // Call the query method
18        int year = request.getYear();
19        String type = request.getType();
20        String length = request.getLength();
21        Document queryResult = queryTop5ExpensiveStates(year, type, length, collection, queryCollection);
22
23        // Convert the query result to the response message
24        QueryTop5ExpensiveStatesResponse.Builder responseBuilder = QueryTop5ExpensiveStatesResponse.newBuilder();
25        for (Object o : queryResult) {
26            Document doc = (Document) o;
27            String state = doc.getString("_id");
28            double totalExpense = doc.getDouble("totalExpense");
29            QueryTop5ExpensiveStatesResponse.QueryResult queryResultMsg = QueryTop5ExpensiveStatesResponse.QueryResult.newBuilder()
30                .setState(state)
31                .setTotalExpense(totalExpense)
32                .build();
33            responseBuilder.addQueryResult(queryResultMsg);
34        }
35
36        // Send the response message to the client
37        QueryTop5ExpensiveStatesResponse response = responseBuilder.build();
38        responseObserver.onNext(response);
39        responseObserver.onCompleted();
40
41        // Close MongoDB connection
42        mongoClient.close();
43    }
44
45    public static Document queryTop5ExpensiveStates(int year, String type, String length, MongoCollection<Document> collection, MongoCollection<Document> queryCollection) {
46        // Check if the query already exists in the query collection
47        Document queryDoc = queryCollection.find(Filters.and(
48            Filters.eq("year", year),
49            Filters.eq("type", type),
50            Filters.eq("length", length)
51        )).first();
52    }
```

**EduCostStatQueryTwoServiceGrpc**

The code above implements a gRPC service. The class `EduCostStatQueryTwoServiceImpl` extends `EduCostStatQueryTwoServiceGrpc.EduCostStatQueryTwoServiceImplBase`, which is generated by the gRPC compiler and provides the base implementation for the service. The methods in `EduCostStatQueryTwoServiceImpl` are the actual implementation of the gRPC service's methods.

The gRPC service method implemented in the code is `queryTop5ExpensiveStates`, which takes a `QueryTop5ExpensiveStatesRequest` and returns a `QueryTop5ExpensiveStatesResponse`. The method also takes a `StreamObserver` as a parameter to send the response back to the client.

The implementation of `queryTop5ExpensiveStates` connects to a MongoDB database, performs a query, and returns the results to the client as a `QueryTop5ExpensiveStatesResponse`. Therefore, the code is a gRPC service implementation that uses a MongoDB database.

```
edu_cost_stat > EduCostStatQueryThreeServer.java > EduCostStatQueryThreeServiceImpl > DB_NAME
1
2
3 public class EduCostStatQueryThreeServiceImpl extends EduCostStatQueryThreeServiceGrpc.EduCostStatQueryThreeServiceImplBase {
4
5     private static final String DB_NAME = "test";
6     private static final String COLLECTION_NAME = "EduCostStat";
7     private static final String QUERY_COLLECTION_NAME = "EduCostStatQueryThree";
8
9     @Override
10    public void queryTop5CheapestStates(QueryTop5CheapestStatesRequest request, StreamObserver<QueryTop5CheapestStatesResponse> responseObserver) {
11        // Set up MongoDB connection
12        MongoClient mongoClient = MongoClient.create("mongodb+srv://rkodava:Dimpu1997@educoststat.ioim58e.mongodb.net/?retryWrites=true&w=majority");
13        MongoDB database = mongoClient.getDatabase(DB_NAME);
14        MongoCollection<Document> collection = database.getCollection(COLLECTION_NAME);
15        MongoCollection<Document> queryCollection = database.getCollection(QUERY_COLLECTION_NAME);
16
17        // Call the query method
18        int year = request.getYear();
19        String type = request.getType();
20        String length = request.getLength();
21        Document queryResult = queryTop5CheapestStates(year, type, length, collection, queryCollection);
22
23        // Convert the query result to the response message
24        QueryTop5CheapestStatesResponse.Builder responseBuilder = QueryTop5CheapestStatesResponse.newBuilder();
25        for (Object o : queryResult) {
26            Document doc = (Document) o;
27            String state = doc.getString("_id");
28            double totalExpense = doc.getDouble("totalExpense");
29            QueryResult queryResultMsg = QueryResult.newBuilder()
30                .setState(state)
31                .setTotalExpense(totalExpense)
32                .build();
33            responseBuilder.addQueryResult(queryResultMsg);
34        }
35
36        // Send the response message to the client
37        QueryTop5CheapestStatesResponse response = responseBuilder.build();
38        responseObserver.onNext(response);
39        responseObserver.onCompleted();
40
41        // Close MongoDB connection
42        mongoClient.close();
43    }
44
45    public static Document queryTop5CheapestStates(int year, String type, String length, MongoCollection<Document> collection, MongoCollection<Document> queryCollection) {
46        // Check if the query already exists in the query collection
47        Document queryDoc = queryCollection.find(Filters.and(
48            Filters.eq("year", year),
49            Filters.eq("type", type),
50            Filters.eq("length", length)
51        ));
52    }
53}
```

## EduCostStatQueryThreeServiceGrpc

The code above implements a gRPC service defined in a proto file. The service is named **EduCostStatQueryThreeService** and has a unary RPC method named **queryTop5CheapestStates**. The implementation class **EduCostStatQueryThreeServiceImpl** extends **EduCostStatQueryThreeServiceGrpc.EduCostStatQueryThreeServiceImplBase**, which is generated by the protobuf compiler and contains the implementation of the **EduCostStatQueryThreeService** service interface.

```
edu_cost_stat > J EduCostStatQueryFourServer.java > EduCostStatQueryFourImpl > DB_NAME
↑ EduCostStatQueryFourImpl
1 public class EduCostStatQueryFourImpl extends EduCostStatQueryFourServiceGrpc.EduCostStatQueryFourServiceImplBase {
2
3     private static final String DB_NAME = "test";
4     private static final String COLLECTION_NAME = "EduCostStat";
5     private static final String QUERY_COLLECTION_NAME = "EduCostStatQueryFour";
6
7
8     @Override
9     public void queryTop5HighestGrowthRateStates(EduCostStatQueryFourRequest request, StreamObserver<EduCostStatQueryFourResponse> responseObserver) {
10         int latestYear = request.getLatestYear();
11         String type = request.getType();
12         String length = request.getLength();
13         int[] pastYears = request.getPastYearsList().toArray();
14
15         // Set up MongoDB connection
16         MongoClient collection = MongoClient.create("mongodb+srv://rkodava:Dimpu1997@educoststat.ioim58e.mongodb.net/?retryWrites=true&w=majority")
17             .getDatabase(DB_NAME)
18             .getCollection(COLLECTION_NAME);
19         MongoCollection<Document> queryCollection = collection.getDatabase().getCollection(QUERY_COLLECTION_NAME);
20
21         // Call the query method
22         Document queryResult = queryTop5HighestGrowthRateStates(latestYear, pastYears, type, length, collection, queryCollection);
23
24         // Create the response message
25         EduCostStatQueryFourResponse.Builder responseBuilder = EduCostStatQueryFourResponse.newBuilder();
26         List<Document> resultDocs = (List<Document>) queryResult.get("result");
27         for (Document resultDoc : resultDocs) {
28             EduCostStatQueryFourResult.Builder resultBuilder = EduCostStatQueryFourResult.newBuilder();
29             resultBuilder.setState(resultDoc.getString("state"));
30             resultBuilder.setGrowthRate(resultDoc.getDouble("growth_rate"));
31             responseBuilder.addResult(resultBuilder.build());
32         }
33         responseObserver.onNext(responseBuilder.build());
34         responseObserver.onCompleted();
35
36     private Document queryTop5HighestGrowthRateStates(int latestYear, int[] pastYears, String type, String length, MongoCollection<Document> collection, MongoCollection<Document> queryCollection) {
37         String[] expenditureFields = {"instruction_expense", "support_services_expense", "other_expenses", "capital_outlay_expense"};
38         String[] revenueFields = {"federal_revenue", "state_revenue", "local_revenue"};
39         List<String> fields = new ArrayList<>();
40         fields.add("state");
41
42         // Build the projection document based on the query parameters
43         for (int pastYear : pastYears) {
44             if (pastYear >= latestYear - 5) {
45                 fields.addAll(Arrays.asList(expenditureFields));
46             }
47             if (pastYear >= latestYear - 4) {
48                 fields.addAll(Arrays.asList(revenueFields));
49             }
50         }
51     }
52 }
```

## EduCostStatQueryFourServiceGrpc

The above code implements a gRPC service. The class **EduCostStatQueryFourImpl** extends **EduCostStatQueryFourServiceGrpc.EduCostStatQueryFourServiceImplBase**, which is a generated class by the gRPC compiler based on the protobuf definition of the service. The `queryTop5HighestGrowthRateStates` method is an implementation of the `queryTop5HighestGrowthRateStates` RPC method defined in the protobuf file, and it takes in a **EduCostStatQueryFourRequest** and a **StreamObserver<EduCostStatQueryFourResponse>** as parameters, which is also defined in the protobuf file.

Additionally, the code creates a MongoDB connection and executes a MongoDB query inside the `queryTop5HighestGrowthRateStates` method, which suggests that this service is using a database to store and retrieve data for the gRPC client requests.



```
edu_cost_stat > J EduCostStatQueryFiveServer.java > EduCostStatQueryFiveImpl > queryRegionAverageExpense(EduCostStatQueryFiveRequest, StreamObserver<EduCostStatQueryFiveResponse>)
↑ EduCostStatQueryFiveImpl
1 public class EduCostStatQueryFiveImpl extends EduCostStatQueryFiveGrpc.EduCostStatQueryFiveImplBase {
2
3     private static final String DB_NAME = "test";
4     private static final String COLLECTION_NAME = "EduCostStat";
5     private static final String QUERY_COLLECTION_NAME = "EduCostStatQueryFive";
6     private static final MongoClient mongoClient = MongoClient.create("mongodb+srv://rkodava:Dimpu1997@educoststat.ioim58e.mongodb.net/?retryWrites=true&w=majority");
7
8     @Override
9     public void queryRegionAverageExpense(EduCostStatQueryFiveRequest request, StreamObserver<EduCostStatQueryFiveResponse> responseObserver) {
10
11         // Get the request parameters
12         int year = request.getYear();
13         String type = request.getType();
14         String length = request.getLength();
15         String[] regions = request.getRegionsList().toArray(new String[0]);
16
17         // Set up the MongoDB connection and get the collections
18         MongoDB database = mongoClient.getDatabase(DB_NAME);
19         MongoCollection<Document> collection = database.getCollection(COLLECTION_NAME);
20         MongoCollection<Document> queryCollection = database.getCollection(QUERY_COLLECTION_NAME);
21
22         // Call the query method
23         Document queryResult = queryRegionAverageExpense(year, type, length, regions, collection, queryCollection);
24
25         // Create the response message
26         EduCostStatQueryFiveResponse.Builder responseBuilder = EduCostStatQueryFiveResponse.newBuilder();
27         responseBuilder.setResult(queryResult.toJson());
28
29         // Send the response
30         responseObserver.onNext(responseBuilder.build());
31         responseObserver.onCompleted();
32     }
33
34     private Document queryRegionAverageExpense(int year, String type, String length, String[] regions, MongoCollection<Document> collection, MongoCollection<Document> queryCollection) {
35         // Check if the query already exists in the query collection
36         Document queryDoc = queryCollection.find(Filters.and(
37             Filters.eq("year", year),
38             Filters.eq("type", type),
39             Filters.eq("length", length),
40             Filters.eq("regions", Arrays.toString(regions))
41         )).first();
42         if (queryDoc != null) {
43             return queryDoc;
44         }
45
46         // Aggregate the region's average overall expense for the given year, type, and length
47         List<Bson> pipeline = new ArrayList<>();
48         pipeline.add(Aggregates.match(Filters.and(
49             Filters.eq("year", year),
50             Filters.eq("type", type),
51             Filters.eq("length", length),
52             Filters.eq("regions", Arrays.toString(regions))
53         )));
54         pipeline.add(Aggregates.group("$regions", Arrays.asList(Aggregates.avg("$expense"))));
55         Document result = queryCollection.aggregate(pipeline).first();
56         return result;
57     }
58 }
```

## EduCostStatQueryFive ServiceGrpc

The above code implements a gRPC service using the `EduCostStatQueryFiveImpl` class. This class extends the `EduCostStatQueryFiveGrpc.EduCostStatQueryFiveImplBase` class, which is a generated class by the gRPC compiler based on the protobuf definition of the service. The `queryRegionAverageExpense` method is an implementation of the `queryRegionAverageExpense` RPC method defined in the protobuf file, and it takes in a `EduCostStatQueryFiveRequest` and a `StreamObserver<EduCostStatQueryFiveResponse>` as parameters, which is also defined in the protobuf file.

The code sets up a MongoDB connection using the `MongoClient` class from the MongoDB Java driver and executes a MongoDB query inside the `queryRegionAverageExpense` method. The query method first checks if the query already exists in the query collection, and if it does, returns the result. If the query does not exist, the method executes an aggregation pipeline to compute the average overall expense for each region for a given year, type, length, and regions. The result is stored in a query document and inserted into the query collection for future use.

Overall, this suggests that the `EduCostStatQueryFive` gRPC service is using a database to store and retrieve data for client requests.

### Task 2.3) Develop the gRPC client and server (or gateway) code to communicate as RPC calls for the five queries defined in Task 1

#### **SERVER :**

The server – ‘EduCostStatServer’ starts the gRPC server and binds the EduCostStatQueryOneServiceImpl , EduCostStatQueryTwoServiceImpl , EduCostStatQueryThreeServiceImpl , EduCostStatQueryFourServiceImpl , EduCostStatQueryFiveServiceImpl implementations (in above task) to it.

Snippets of the code to the server – ‘EduCostStatServer’ is below -

```
edu_cost_stat > EduCostStatServer.java > EduCostStatServer > EduCostStatServer(ServerBuilder<?>, int)
10
11 public class EduCostStatServer {
12     private final int port;
13     private final Server server;
14
15     public EduCostStatServer(int port) throws IOException {
16         this(ServerBuilder.forPort(port), port);
17     }
18
19     public EduCostStatServer(ServerBuilder<?> serverBuilder, int port) {
20         this.port = port;
21         server = serverBuilder.addService(new EduCostStatQueryOneImpl())
22             .addService(new EduCostStatQueryTwoImpl())
23             .addService(new EduCostStatQueryThreeImpl())
24             .addService(new EduCostStatQueryFourImpl())
25             .addService(new EduCostStatQueryFiveImpl())
26             .build();
27     }
28
29     public void start() throws IOException {
30         server.start();
31         System.out.println("Server started, listening on " + port);
32
33         Runtime.getRuntime().addShutdownHook(new Thread() {
34             @Override
35             public void run() {
36                 System.err.println("Shutting down gRPC server since JVM is shutting down");
37                 try {
38                     EduCostStatServer.this.stop();
39                 } catch (InterruptedException e) {
40                     e.printStackTrace(System.err);
41                 }
42                 System.err.println("Server shut down");
43             }
44         });
45     }
46
47     public void stop() throws InterruptedException {
48         if (server != null) {
49             server.shutdown().awaitTermination(30, TimeUnit.SECONDS);
50         }
51     }
52
53     private void blockUntilShutdown() throws InterruptedException {
54         if (server != null) {
55             server.awaitTermination();
56         }
57     }
58
59     Run | Debug
60     public static void main(String[] args) throws Exception {
61         EduCostStatServer server = new EduCostStatServer(port:50051);
62         server.start();
63         server.blockUntilShutdown();
64     }
65 }
66
```

The above code is an implementation of a gRPC server called EduCostStatServer that exposes five different services to clients. The server listens on port 50051.

The `EduCostStatServer` constructor takes a port number and a `ServerBuilder` object as arguments. The `ServerBuilder` object is used to create the server and add the five different services (`EduCostStatQueryOneImpl`, `EduCostStatQueryTwoImpl`, `EduCostStatQueryThreeImpl`, `EduCostStatQueryFourImpl`, and `EduCostStatQueryFiveImpl`) to the server.

The `start()` method starts the server and prints a message to the console indicating that the server is listening on the specified port. It also adds a shutdown hook to the JVM so that the server can be stopped gracefully when the JVM shuts down.

The `stop()` method shuts down the server and blocks until the shutdown is complete.

The `blockUntilShutdown()` method blocks until the server is terminated.

The `main()` method creates a new instance of the `EduCostStatServer` class and starts it. The program blocks until the server is shut down or interrupted.

## CLIENT :

The client is used to communicate with a server that provides information about educational costs statistics. The client communicates with the server using gRPC, a high-performance open-source remote procedure call (RPC) framework that enables client and server applications to communicate transparently.

Snippets of the code to the client – ‘EduCostStatClient’ is below -

```
edu_cost_stat > J EduCostStatClient.java > ...
1  import io.grpc.ManagedChannel;
2  import io.grpc.ManagedChannelBuilder;
3
4  public class EduCostStatClient {
5      private final ManagedChannel channel;
6      private final EduCostStatQueryOneGrpc.EduCostStatQueryOneBlockingStub queryOneStub;
7      private final EduCostStatQueryTwoGrpc.EduCostStatQueryTwoBlockingStub queryTwoStub;
8      private final EduCostStatQueryThreeGrpc.EduCostStatQueryThreeBlockingStub queryThreeStub;
9      private final EduCostStatQueryFourGrpc.EduCostStatQueryFourBlockingStub queryFourStub;
10     private final EduCostStatQueryFiveGrpc.EduCostStatQueryFiveBlockingStub queryFiveStub;
11
12     public EduCostStatClient(String host, int port) {
13         this(ManagedChannelBuilder.forAddress(host, port)
14             .usePlaintext()
15             .build());
16     }
17
18     public EduCostStatClient(ManagedChannel channel) {
19         this.channel = channel;
20         this.queryOneStub = EduCostStatQueryOneGrpc.newBlockingStub(channel);
21         this.queryTwoStub = EduCostStatQueryTwoGrpc.newBlockingStub(channel);
22         this.queryThreeStub = EduCostStatQueryThreeGrpc.newBlockingStub(channel);
23         this.queryFourStub = EduCostStatQueryFourGrpc.newBlockingStub(channel);
24         this.queryFiveStub = EduCostStatQueryFiveGrpc.newBlockingStub(channel);
25     }
26
27     public void shutdown() throws InterruptedException {
28         channel.shutdown().awaitTermination(5, TimeUnit.SECONDS);
29     }
30
31     public void queryOne(int year, String state, String type, String length, int expense) {
32         EduCostStatQueryOneRequest request = EduCostStatQueryOneRequest.newBuilder()
33             .setYear(year)
34             .setState(state)
35             .setType(type)
36             .setLength(length)
37             .setExpense(expense)
38             .build();
39
40         EduCostStatQueryOneResponse response = queryOneStub.queryCost(request);
41
42         System.out.println("Query One Response: " + response.getResult());
43     }
44
45     public void queryTwo(int year, String type, String length) {
46         EduCostStatQueryTwoRequest request = EduCostStatQueryTwoRequest.newBuilder()
47             .setYear(year)
48             .setType(type)
49             .setLength(length)
50             .build();
51
52         EduCostStatQueryTwoResponse response = queryTwoStub.queryTop5ExpensiveStates(request);
53
54         System.out.println("Query Two Response: " + response.getResult());
55     }
56
57     public void queryThree(int year, String type, String length) {
58         EduCostStatQueryThreeRequest request = EduCostStatQueryThreeRequest.newBuilder()
59             .setYear(year)
60             .setType(type)
61             .setLength(length)
62             .build();
63     }
```



```

edu_cost_stat > EduCostStatClient.java > ...
65     EduCostStatQueryThreeResponse response = queryThreeStub.queryEconomicStates(request);
66
67     System.out.println("Query Three Response: " + response.getResult());
68 }
69
70 public void queryFour(int year, String type, String length, int[] pastYears) {
71     EduCostStatQueryFourRequest request = EduCostStatQueryFourRequest.newBuilder()
72         .setYear(year)
73         .setType(type)
74         .setLength(length)
75         .setPastYears(pastYears)
76         .build();
77
78     EduCostStatQueryFourResponse response = queryFourStub.queryTop5HighestGrowthRateStates(request);
79
80     System.out.println("Query Four Response: " + response.getResult());
81 }
82
83 public void queryFive(int year, String type, String length, String... regions) {
84     EduCostStatQueryFiveRequest request = EduCostStatQueryFiveRequest.newBuilder()
85         .setYear(year)
86         .setType(type)
87         .setLength(length)
88         .addAllRegions(Arrays.asList(regions))
89         .build();
90
91     EduCostStatQueryFiveResponse response = queryFiveStub.queryRegionAverageExpense(request);
92
93     System.out.println("Query Five Response: " + response.getResult());
94 }
95
96 public static void main(String[] args) throws Exception {
97     EduCostStatClient client = null;
98     try {
99         // create a client
100         client = new EduCostStatClient(host:"localhost", port:50051);
101
102         // run query one
103         client.queryOne(year + ", " + state + ", " + type + ", " + length + ", " + expense);
104
105         // run query two
106         client.queryTwo(year + ", " + type + ", " + length);
107
108         // run query three
109         client.queryThree(year + ", " + type + ", " + length);
110
111         // run query four
112         client.queryFour(year + ", " + type + ", " + length + ", " + pastYears);
113
114         // run query five
115         client.queryFive(year:2020, type:"Private", length:"2 Years", ...regions:"Northeast", "South");
116     } finally {
117         // shutdown client
118         if (client != null) {
119             client.shutdown();
120         }
121     }
122 }
123
124 }
125
126

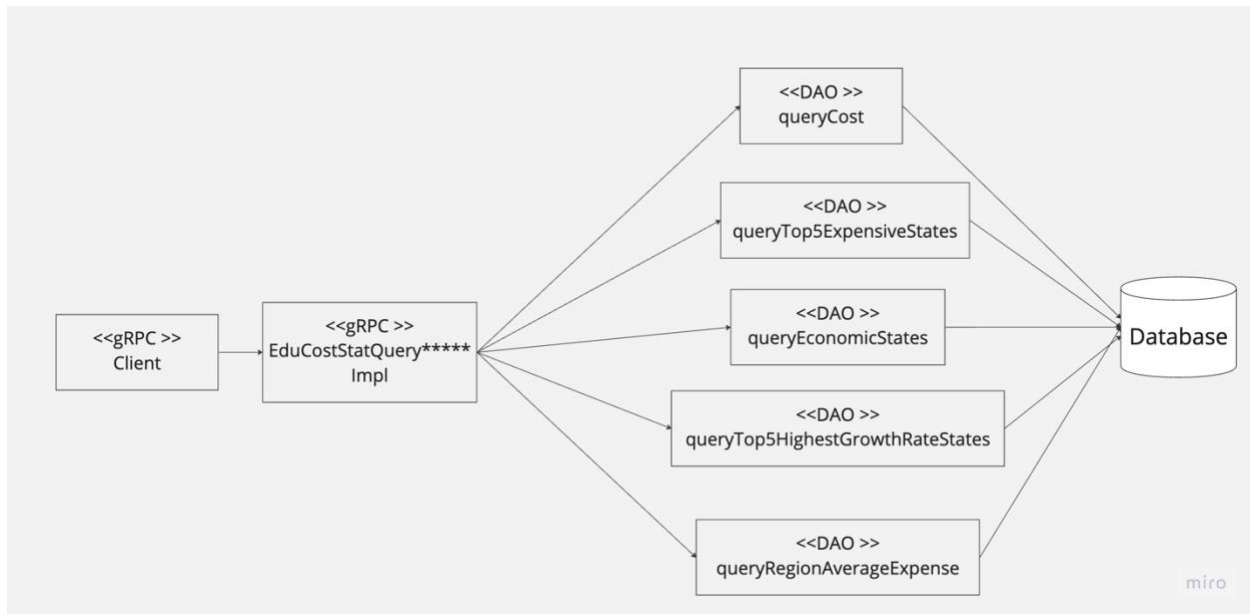
```

The above client code defines a Java class called `EduCostStatClient`, which is used to communicate with a gRPC server. The `EduCostStatClient` class has several methods for running different queries on the server, and it takes care of setting up the gRPC communication channel and stubs for each query.

The `EduCostStatClient` constructor takes a host and port, which are used to set up the gRPC communication channel with the server. The `queryOne`, `queryTwo`, `queryThree`, `queryFour`, and `queryFive` methods take different input parameters for each query and use the corresponding gRPC stubs to send the request to the server and receive the response.

The `main` method creates an instance of the `EduCostStatClient` class and calls the different query methods with sample input parameters. Finally, the `shutdown` method is called to terminate the gRPC channel connection.

So, therefore from above tasks the Overall Assignment 2 Data Operation Architecture with RPC Communication is as below –



→ EduCostStatQuery\*\*\*\*\*Impl may be considered as any of the query – EduCostStatQueryOneImpl , EduCostStatQueryTwoImpl, EduCostStatQueryThreeImpl, EduCostStatQueryFourImpl or EduCostStatQueryFiveImpl .

When the client code invokes the appropriate query method, it will communicate with the server using the gRPC protocol to call the corresponding method in the EduCostStatQuery\*\*\*\*\*Impl implementation on the server-side. The server will then execute the query and return the results back to the client through the gRPC response message. The client will receive the response and print the results to the console.