Name: Yash Jayeshkumar Pandya
Email : yashp614929@gmail.com
Project : Distributed Systems & Cloud Databases

File : **Driver.java**

- **SQL Connection Variables**

```
// MySql credentials
String sqlUrl =
"jdbc:mysql://pgc-sd-bigdata.cyaielc9bmnf.us-east-1.rds.amazonaws.com:330
6/pgcdata";
String sqlUser = "student";
String sqlPassword = "STUDENT123";
```

The above portion of the code is used to define connection variables so that the program connects with SQL Database.

- **MongoDB Connection Variables**

```
// MongoDB Configurations
String mongodbUrl =
"mongodb://ec2-54-89-205-236.compute-1.amazonaws.com:27017";
String mongodbDB = "Assignment12";
String mongodbCollection = "products12";
```

The above part of the code is used to define the connection variables so that the program connects with MongoDB Database.

- **Default variable declaration**

```
// Connection Default Value Initialization
Connection sqlConnection = null;
MongoClient mongoClient = null;
```

The above part of the code is used to declare class variables with default value so that further it is used to connect with SQL and MongoDB

- **Creation of SQL Connection**

```
// SQL connections
sqlConnection = DriverManager.getConnection(sqlUrl, sqlUser,
sqlPassword);
Statement statement = sqlConnection.createStatement();
```

The above part of code uses jdbc DriverManager class and uses it's getConnection method which takes SQL DB URL as string along with SQL Username and Password strings to establish connection with SQL DB. The second line creates a Statement class object from the SQL Connection which is further used to execute SQL queries.

- **Creation of MongoDB Connection**

```
// mongoDB connections
mongoClient = MongoClients.create(mongodbUrl);
MongoDatabase db = mongoClient.getDatabase(mongodbDB);

// If collection is already exists with data then drop the collection
and program recreates collection and adds data from SQL tables
db.getCollection(mongodbCollection).drop();

// Creation of mongoDB collection instance
MongoCollection collection = db.getCollection(mongodbCollection);
```

The above part of code uses mongodbUrl string to create mongoClient instance which will further use to get the MongoDatabase db instance from the specified database name in mongodbDB. The next part is used to check if collection already exists then it drops the existing connection so that data will not be duplicated while running the program multiple times. And at the end it creates a MongoCollection instance with the specified collection name from the mongodbCollection variable in the declaration part.

- **Data Import**

```
// Import data into MongoDb
readDataFromSQLAndImportDataInMongoDB(statement, collection, "mobiles");
readDataFromSQLAndImportDataInMongoDB(statement, collection, "cameras");
readDataFromSQLAndImportDataInMongoDB(statement, collection,
"headphones");
```

The above part of code used to import the data from the three different tables mobiles, cameras, headphones specified at the last index of the function call.

```
public static void readDataFromSQLAndImportDataInMongoDB(
      Statement statement, MongoCollection<Document> collection,
      String tableName
) throws SQLException {
   // Create SQL string to execute query and get ResultSet and
   ResultSetMetaData
   String sql = "select * from " + tableName;
   ResultSet rs = statement.executeQuery(sql);
   ResultSetMetaData rsmd = rs.getMetaData();

   // Add new attribute Category in mongoDB collection with specific
   value
   String category = null;
   if (tableName.equals("mobiles")) {
       category = "Mobiles";
   } else if (tableName.equals("cameras")) {
       category = "Cameras";
   } else {
       category = "Headphones";
   }

   // Iterate through all rows in ResultSet
   while (rs.next()) {
       // Create new document
       Document doc = new Document();
       // Iterate through all columns in ResultSet and add column name
       and value pair in mongoDB document
       for(int i=1; i<=rsmd.getColumnCount();i++) {
           doc.append(rsmd.getColumnName(i), rs.getString(i));
       }
       // Add Category in mongoDB document
       doc.append("Category", category);
       // Add document in mongoDB collection
       collection.insertOne(doc);
   }
}
```

The above function takes a Statement object for SQL read operations, MongoCollection object as collection to write the data in mongoDB collection and tableName which specifies the SQL table name used to get the data. Based on executing the query and getting the metadata, from the metadata I have used column name as a key and iterating over the ResultSet to get the values for each row as value and putting into mongodb Document class object. I have added “Category” as a key in mongodb collection and based on the tableName the value of this key is mapped. Ie. Mobiles, Cameras, Headphones.

- **Display Result function calls**

```
// List all products in the inventory
CRUDHelper.displayAllProducts(collection);

// Display top 5 Mobiles
CRUDHelper.displayTop5Mobiles(collection);

// Display products ordered by their categories in Descending Order
Without autogenerated Id
CRUDHelper.displayCategoryOrderedProductsDescending(collection);

// Display product count in each category
CRUDHelper.displayProductCountByCategory(collection);

// Display wired headphones
CRUDHelper.displayWiredHeadphones(collection);
```

The above part of the code is used to call the mongoDB collection query execution Methods.

- **Closing statements**

```
// closing statement instance
statement.close();

if (sqlConnection != null){
   sqlConnection.close();
}
if (null != mongoClient) {
   mongoClient.close();
}
```

The above code is used to close connection instances of SQL and mongoDB.

File : **CRUDHelper.java**

Query 01 : `collection.find()`

```
public static void displayAllProducts(MongoCollection<Document> collection) {
   System.out.println("------ Displaying All Products ------");
   // Call printSingleCommonAttributes to display the attributes on the Screen
   MongoCursor<Document> cursor = collection.find().cursor();
   try {
       while (cursor.hasNext()) {
           PrintHelper.printSingleCommonAttributes(cursor.next());
       }
   } finally {
       cursor.close();
   }
   System.out.println();
}
```

The above part of the code takes collection instance and performing query to the collection instance which returns cursor instance which will use in printSingleCommonAttributes method of PrintHelper class to print the common attributes on the console.

Query 02 : `collection.find(new Document("Category","Mobiles")).limit(5)`

```
public static void displayTop5Mobiles(MongoCollection<Document> collection) {
   System.out.println("------ Displaying Top 5 Mobiles ------");
   // Call printAllAttributes to display the attributes on the Screen
      MongoCursor<Document>  cursor  =  collection.find(new  Document("Category",
"Mobiles")).limit(5).cursor();
   try {
       while (cursor.hasNext()) {
           PrintHelper.printAllAttributes(cursor.next());
       }
   } finally {
       cursor.close();
   }
   System.out.println();
}
```

The above part of the code takes a collection instance and performs a query by calling find method of collection instance which takes a Document instance having category as Mobiles which filters data from mongodb whose Category is Mobiles and by using limit only 5 top rows are fetched in cursor. And, cursor follows calling printAllAttributes method of PrintHelper class to display results on the console.

Query 03 :

```
collection.find().sort(new
Document("Category",-1)).projection(fields(excludeId()))
```

```
public static void
displayCategoryOrderedProductsDescending(MongoCollection<Document> collection)
{
   System.out.println("------ Displaying Products ordered by categories
------");
   // Call printAllAttributes to display the attributes on the Screen
   MongoCursor<Document> cursor = collection.find().sort(new
Document("Category", -1)).projection(fields(excludeId())).cursor();
   try {
       while (cursor.hasNext()) {
           PrintHelper.printAllAttributes(cursor.next());
       }
   } finally {
       cursor.close();
   }
   System.out.println();
}
```

The above part of the code takes a collection instance and performs a query by calling find method of collection instance after that uses sort which takes a Document instance having category with value -1 which performs descending ordering on category the result will yet be updated by excluding mongoDB default generated ID called using fields in projection. And, cursor follows calling printAllAttributes method of PrintHelper class to display results on the console.

Query 04 :

```
collection.aggregate(Arrays.asList(Aggregates.group("$Category",
Accumulators.sum("Count", 1))))
```

```
public static void displayProductCountByCategory(MongoCollection<Document>
collection) {
   System.out.println("------ Displaying Product Count by categories ------");
   // Call printProductCountInCategory to display the attributes on the Screen
   MongoCursor<Document> cursor =
collection.aggregate(Arrays.asList(Aggregates.group("$Category",
Accumulators.sum("Count", 1)))).cursor();
   try {
       while (cursor.hasNext()) {
           PrintHelper.printProductCountInCategory(cursor.next());
       }
   } finally {
```

```
        cursor.close();
    }
    System.out.println();
}
```

The above part of the code takes a collection instance and performs a query by aggregating collection instances with group by Category and the temporary result is stored as an arraylist. Then Accumulators.sum can perform the count operation on the array based on results by the group which initiated with value 1, so at the end result contains total number of rows/data by each Category. And, cursor follows calling printProductCountInCategory method of PrintHelper class to display results on the Console.

Query 05 :

```
collection.find(new
Document("Category","Headphones").append("ConnectorType", "Wired"))

public static void displayWiredHeadphones(MongoCollection<Document> collection)
{
    System.out.println("------ Displaying Wired headphones ------");
    // Call printAllAttributes to display the attributes on the Screen
       MongoCursor<Document>  cursor  =  collection.find(new  Document("Category",
"Headphones").append("ConnectorType", "Wired")).cursor();
    try {
        while (cursor.hasNext()) {
            PrintHelper.printAllAttributes(cursor.next());
        }
    } finally {
        cursor.close();
    }
    System.out.println();
}
```

The above part of the code takes a collection instance and performs a query by calling find method of collection instance which takes a Document instance having category as Headphones which filters data from mongodb whose Category is Headphones and ConnetoryType as Wired. The results are fetched in cursor. And, cursor follows calling printAllAttributes method of PrintHelper class to display results on the console.
