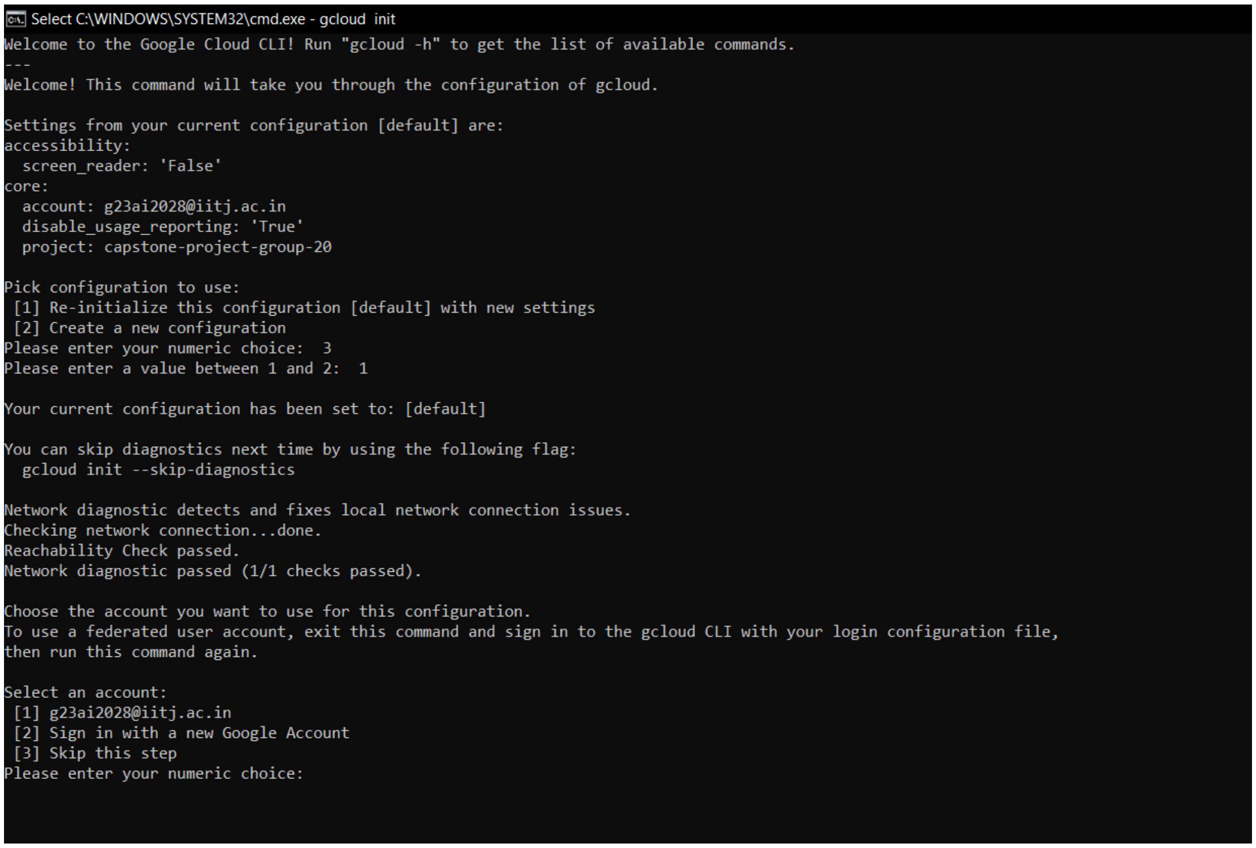
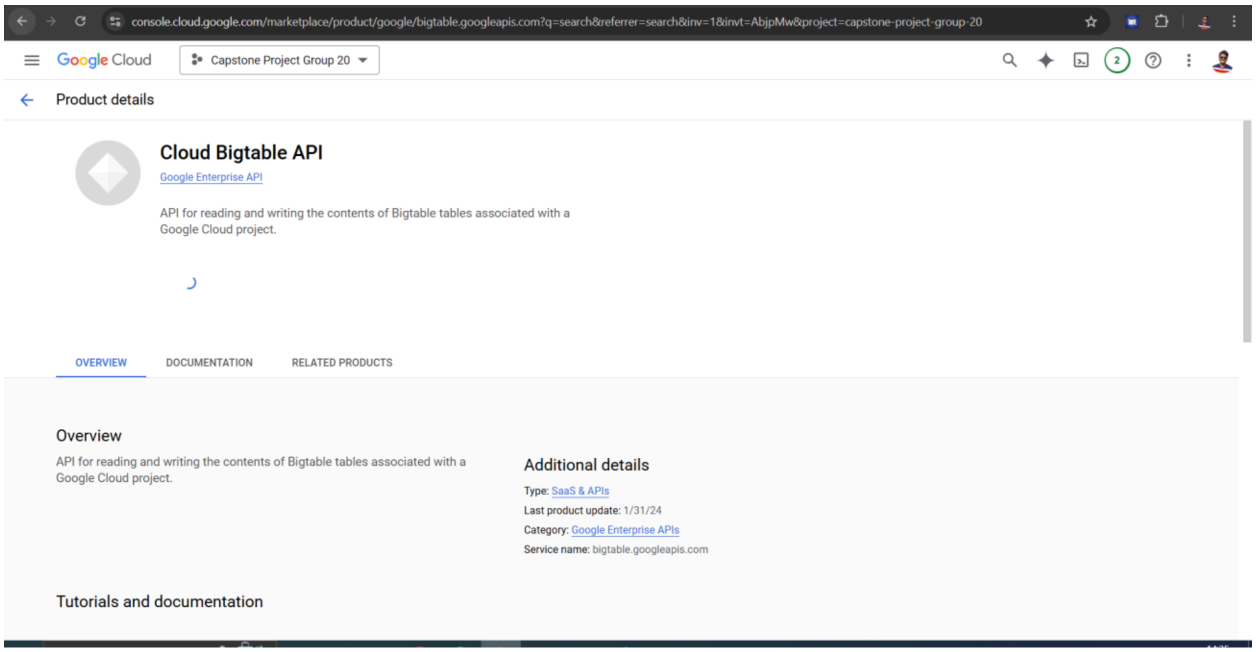
BIG DATA MANAGEMENT
Assignment - 4

Name	:	Shubham Raj
Roll No	:	G23AI2028


Select C:\WINDOWS\SYSTEM32\cmd.exe - gcloud init

```
Welcome to the Google Cloud CLI! Run "gcloud -h" to get the list of available commands.
---
Welcome! This command will take you through the configuration of gcloud.

Settings from your current configuration [default] are:
accessibility:
  screen_reader: 'False'
core:
  account: g23ai2028@iitj.ac.in
  disable_usage_reporting: 'True'
  project: capstone-project-group-20

Pick configuration to use:
 [1] Re-initialize this configuration [default] with new settings
 [2] Create a new configuration
Please enter your numeric choice:  3
Please enter a value between 1 and 2:  1

Your current configuration has been set to: [default]

You can skip diagnostics next time by using the following flag:
  gcloud init --skip-diagnostics

Network diagnostic detects and fixes local network connection issues.
Checking network connection...done.
Reachability Check passed.
Network diagnostic passed (1/1 checks passed).

Choose the account you want to use for this configuration.
To use a federated user account, exit this command and sign in to the gcloud CLI with your login configuration file,
then run this command again.

Select an account:
 [1] g23ai2028@iitj.ac.in
 [2] Sign in with a new Google Account
 [3] Skip this step
Please enter your numeric choice:
```


console.cloud.google.com/marketplace/product/google/bigtable.googleapis.com?q=search&referrer=search&inv=1&invt=AbjpMw&project=capstone-project-group-20

Product details

Cloud Bigtable API

Google Enterprise API

API for reading and writing the contents of Bigtable tables associated with a Google Cloud project.

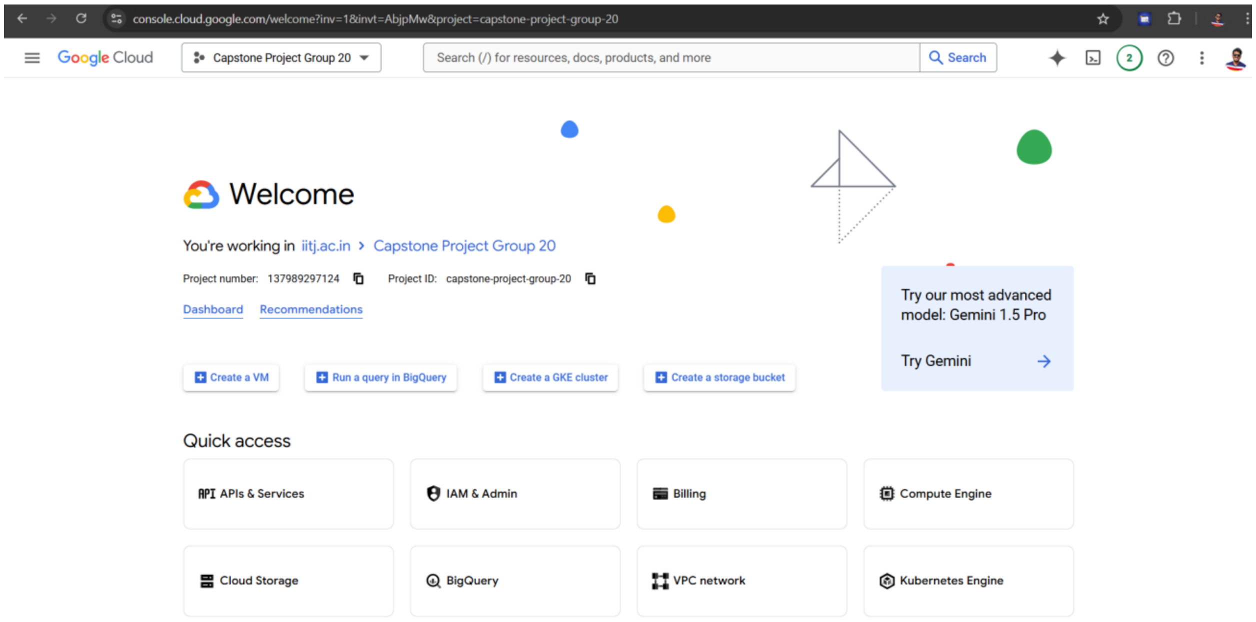
Overview

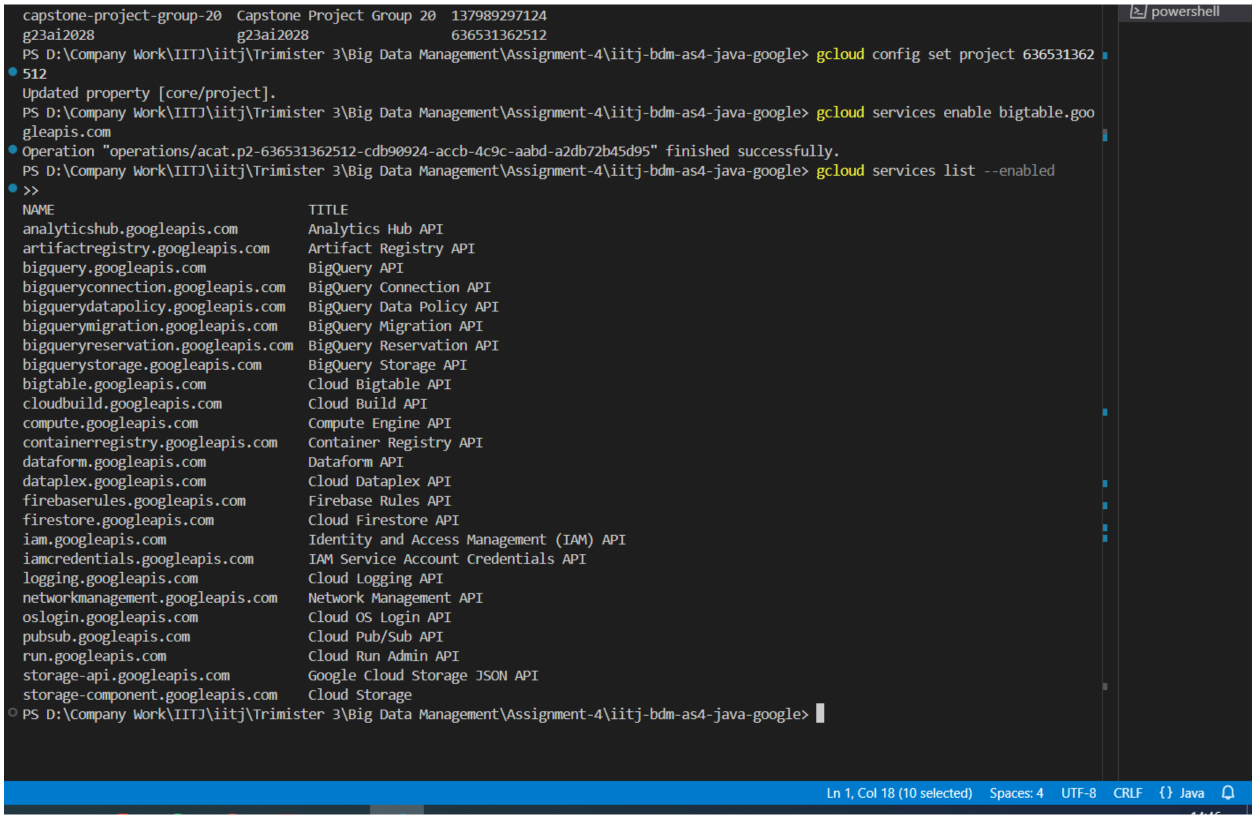
API for reading and writing the contents of Bigtable tables associated with a Google Cloud project.

Additional details

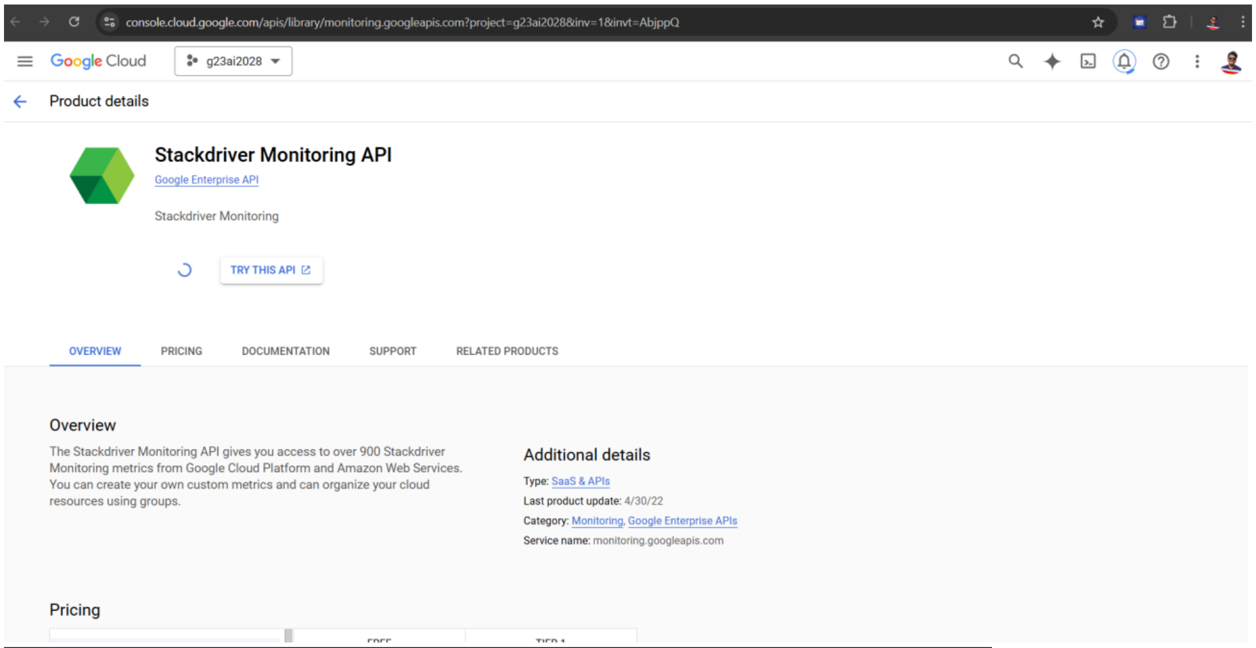
Type: SaaS & APIs
Last product update: 1/31/24
Category: Google Enterprise APIs
Service name: bigtable.googleapis.com

Tutorials and documentation


console.cloud.google.com/welcome?inv=1&invt=AbjpMw&project=capstone-project-group-20
Google Cloud
Capstone Project Group 20
Search (/) for resources, docs, products, and more
Search
Welcome
You're working in iitj.ac.in > Capstone Project Group 20
Project number: 137989297124
Project ID: capstone-project-group-20
Dashboard
Recommendations
Create a VM
Run a query in BigQuery
Create a GKE cluster
Create a storage bucket
Try our most advanced model: Gemini 1.5 Pro
Try Gemini
Quick access
APIs & Services
IAM & Admin
Billing
Compute Engine
Cloud Storage
BigQuery
VPC network
Kubernetes Engine



```
capstone-project-group-20  Capstone Project Group 20  137989297124
g23ai2028                  g23ai2028                  636531362512
PS D:\Company Work\IITJ\iitj\Trimister 3\Big Data Management\Assignment-4\iitj-bdm-as4-java-google> gcloud config set project 636531362
512
Updated property [core/project].
PS D:\Company Work\IITJ\iitj\Trimister 3\Big Data Management\Assignment-4\iitj-bdm-as4-java-google> gcloud services enable bigtable.goo
gleapis.com
Operation "operations/acat.p2-636531362512-cdb90924-accb-4c9c-aabd-a2db72b45d95" finished successfully.
PS D:\Company Work\IITJ\iitj\Trimister 3\Big Data Management\Assignment-4\iitj-bdm-as4-java-google> gcloud services list --enabled
>>
NAME                                  TITLE
analyticshub.googleapis.com           Analytics Hub API
artifactregistry.googleapis.com       Artifact Registry API
bigquery.googleapis.com               BigQuery API
bigqueryconnection.googleapis.com     BigQuery Connection API
bigquerydatapolicy.googleapis.com     BigQuery Data Policy API
bigquerymigration.googleapis.com      BigQuery Migration API
bigqueryreservation.googleapis.com    BigQuery Reservation API
bigquerystorage.googleapis.com        BigQuery Storage API
bigtable.googleapis.com               Cloud Bigtable API
cloudbuild.googleapis.com             Cloud Build API
compute.googleapis.com                Compute Engine API
containerregistry.googleapis.com      Container Registry API
dataform.googleapis.com               Dataform API
dataplex.googleapis.com               Cloud Dataplex API
firebaserules.googleapis.com          Firebase Rules API
firestore.googleapis.com              Cloud Firestore API
iam.googleapis.com                    Identity and Access Management (IAM) API
iamcredentials.googleapis.com         IAM Service Account Credentials API
logging.googleapis.com                Cloud Logging API
networkmanagement.googleapis.com      Network Management API
oslogin.googleapis.com                Cloud OS Login API
pubsub.googleapis.com                 Cloud Pub/Sub API
run.googleapis.com                    Cloud Run Admin API
storage-api.googleapis.com            Google Cloud Storage JSON API
storage-component.googleapis.com      Cloud Storage
PS D:\Company Work\IITJ\iitj\Trimister 3\Big Data Management\Assignment-4\iitj-bdm-as4-java-google>
```



console.cloud.google.com/apis/library/monitoring.googleapis.com?project=g23ai2028&inv=1&invt=AbjppQ

Google Cloud g23ai2028

Product details

Stackdriver Monitoring API

Google Enterprise API

Stackdriver Monitoring

TRY THIS API

OVERVIEW PRICING DOCUMENTATION SUPPORT RELATED PRODUCTS

Overview

The Stackdriver Monitoring API gives you access to over 900 Stackdriver Monitoring metrics from Google Cloud Platform and Amazon Web Services. You can create your own custom metrics and can organize your cloud resources using groups.

Additional details

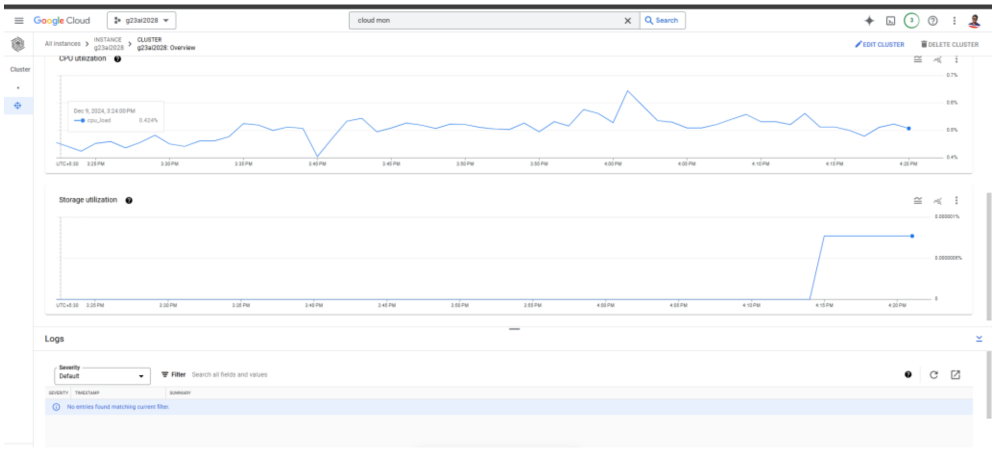
Type: SaaS & APIs

Last product update: 4/30/22

Category: Monitoring, Google Enterprise APIs

Service name: monitoring.googleapis.com

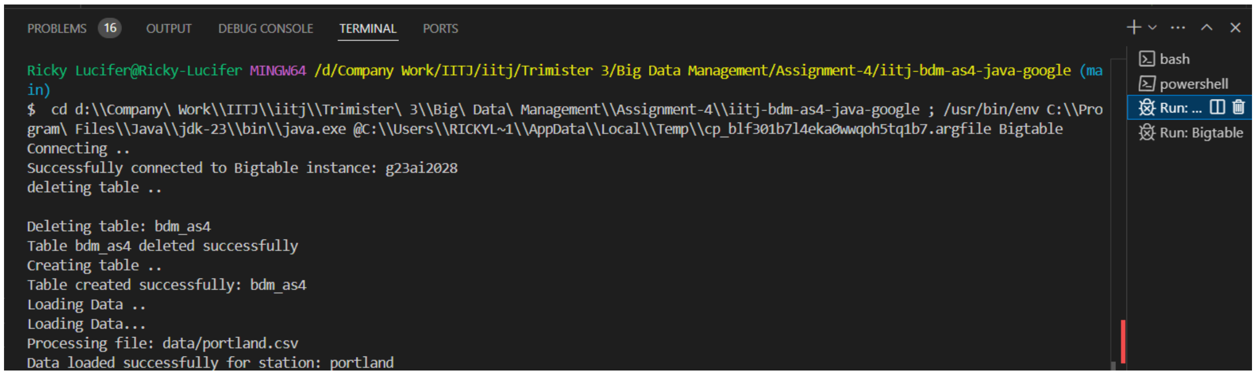
Pricing



1. 10 mark - Write the method connect() to create a connection. Create a Bigtable data client and admin client. See HelloWorld.java for starter code.

```java
public void connect() {
        try {
            // Settinging up data client to interact with Bigtable
            BigtableDataSettings dataSettings = BigtableDataSettings.newBuilder()
                    .setProjectId("g23ai2028") // Project ID
                    .setInstanceId("g23ai2028") // Instance ID
                    .build();
            dataClient = BigtableDataClient.create(dataSettings);

            // Settinging up admin client to manage Bigtable resources
            BigtableTableAdminSettings adminSettings =
BigtableTableAdminSettings.newBuilder()
                    .setProjectId("g23ai2028") // Project ID
                    .setInstanceId("g23ai2028") // Instance ID
                    .build();
            adminClient = BigtableTableAdminClient.create(adminSettings);

            System.out.println("Successfully connected to Bigtable instance: 
g23ai2028");
        } catch (Exception e) {
            System.out.println("Error: Unable to connect to Bigtable instance.");
            e.printStackTrace();
        }
    }
```

Output

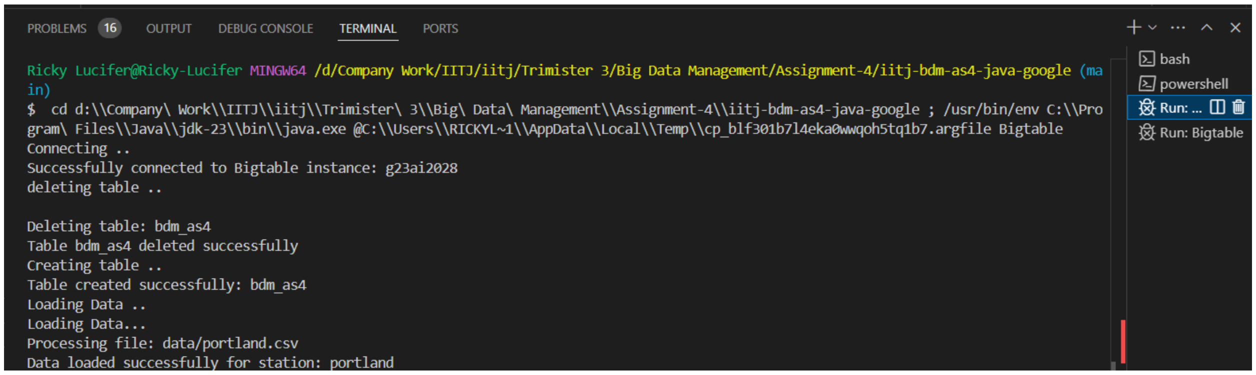


```
Ricky Lucifer@Ricky-Lucifer MINGW64 /d/Company Work/IITJ/iitj/Trimister 3/Big Data Management/Assignment-4/iitj-bdm-as4-java-google (ma
in)
$  cd d:\\Company\ Work\\IITJ\\iitj\\Trimister\ 3\\Big\ Data\ Management\\Assignment-4\\iitj-bdm-as4-java-google ; /usr/bin/env C:\\Pro
gram\ Files\\Java\\jdk-23\\bin\\java.exe @C:\\Users\\RICKYL~1\\AppData\\Local\\Temp\\cp_blf301b7l4eka0wwqoh5tq1b7.argfile Bigtable
Connecting ..
Successfully connected to Bigtable instance: g23ai2028
deleting table ..

Deleting table: bdm_as4
Table bdm_as4 deleted successfully
Creating table ..
Table created successfully: bdm_as4
Loading Data ..
Loading Data...
Processing file: data/portland.csv
Data loaded successfully for station: portland
```

2. 10 mark - Write the method createTable() to create a table to store the sensor data.

```
public void createTable() {
        try {
            if (!adminClient.exists(tableId)) {
                CreateTableRequest request = CreateTableRequest.of(tableId)
                        .addFamily(COLUMN_FAMILY); // Adding single column 
family: "sensor"
                adminClient.createTable(request);
                System.out.println("Table created successfully: " + tableId);
            } else {
                System.out.println("Table already exists: " + tableId);
            }
        } catch (Exception e) {
            System.err.println("Error creating table: " + e.getMessage());
            e.printStackTrace();
        }
    }
```

Output



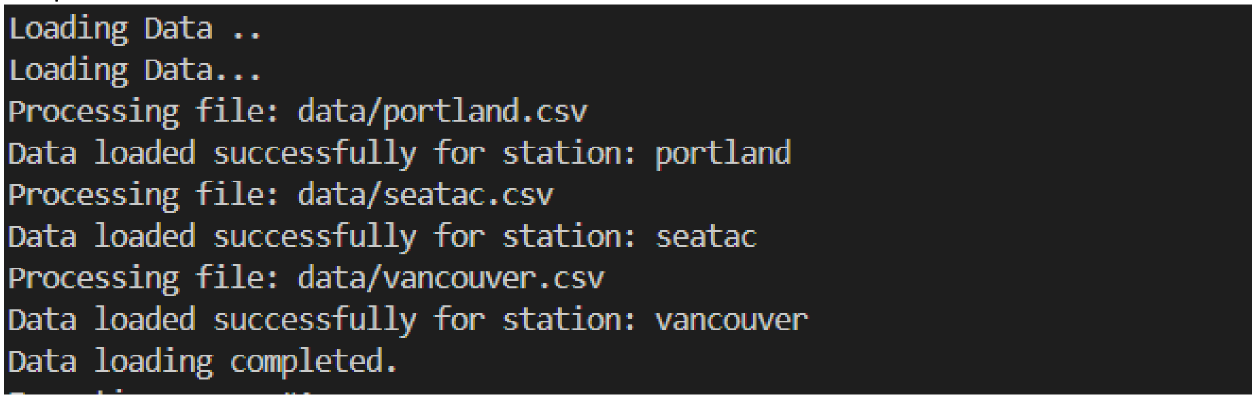
```
Ricky Lucifer@Ricky-Lucifer MINGW64 /d/Company Work/IITJ/iitj/Trimister 3/Big Data Management/Assignment-4/iitj-bdm-as4-java-google (ma
in)
$  cd d:\\Company\ Work\\IITJ\\iitj\\Trimister\ 3\\Big\ Data\ Management\\Assignment-4\\iitj-bdm-as4-java-google ; /usr/bin/env C:\\Pro
gram\ Files\\Java\\jdk-23\\bin\\java.exe @C:\\Users\\RICKYL~1\\AppData\\Local\\Temp\\cp_blf301b7l4eka0wwqoh5tq1b7.argfile Bigtable
Connecting ..
Successfully connected to Bigtable instance: g23ai2028
deleting table ..

Deleting table: bdm_as4
Table bdm_as4 deleted successfully
Creating table ..
Table created successfully: bdm_as4
Loading Data ..
Loading Data...
Processing file: data/portland.csv
Data loaded successfully for station: portland
```

3. 5 marks - Write the method load() to load the sensor data into the database. The data files are in the data folder.

```
  /**
     * Loads data into database.
     * Data is in CSV files. Note that it must be converted to hourly data.
     * Takes first reading in an hour and ignores any others.
     */
    public void loadData() throws Exception {
        // Hardcoding  files paths for specific data files so no mismathc happns
        String[] files = {
                "data/portland.csv",
                "data/seatac.csv",
                "data/vancouver.csv"
        };

        System.out.println("Loading Data...");

        for (String filePath : files) {
            // Extracting station ID from file name
            String stationId = filePath.substring(filePath.lastIndexOf("/") + 1,
filePath.lastIndexOf("."));
            System.out.println("Processing file: " + filePath);

            try (BufferedReader reader = new BufferedReader(new
FileReader(filePath))) {
                String line = reader.readLine(); // Skiping header row so we can
save data
                String lastHour = ""; // To track hourly data
                BulkMutation bulkMutation = BulkMutation.create(tableId); // Use
bulk mutation for better detail view

                while ((line = reader.readLine()) != null) {
                    String[] fields = line.split(",");

                    if (fields.length < 9)
```

```
                        continue; // Skiping invalid rows

                    String date = fields[1]; // Date
                    String time = fields[2]; // Time
                    String hour = time.split(":")[0]; // Extracting hour

                    // Skiping non-hourly data
                    if (hour.equals(lastHour))
                        continue;
                    lastHour = hour;

                    // Constructing row key
                    String rowKey = stationId + "#" + date + "#" + hour;

                    // Adding mutation entry for current row
                    bulkMutation.add(RowMutationEntry.create(rowKey)
                            .setCell(COLUMN_FAMILY, "temperature", fields[3]) //
Temperature
                            .setCell(COLUMN_FAMILY, "dewpoint", fields[4]) //
Dewpoint
                            .setCell(COLUMN_FAMILY, "relhum", fields[5]) //
Relative Humidity
                            .setCell(COLUMN_FAMILY, "speed", fields[6]) // Wind
Speed
                            .setCell(COLUMN_FAMILY, "gust", fields[7]) // Wind
Gust
                            .setCell(COLUMN_FAMILY, "pressure", fields[8])); //
Atmospheric Pressure
                }

                // Executing all mutations for current file
                dataClient.bulkMutateRows(bulkMutation);
                System.out.println("Data loaded successfully for station: " +
stationId);
            } catch (IOException e) {
                System.err.println("Error reading file: " + filePath + " - " +
e.getMessage());
            }
        }

        System.out.println("Data loading completed.");
    }
```

Output

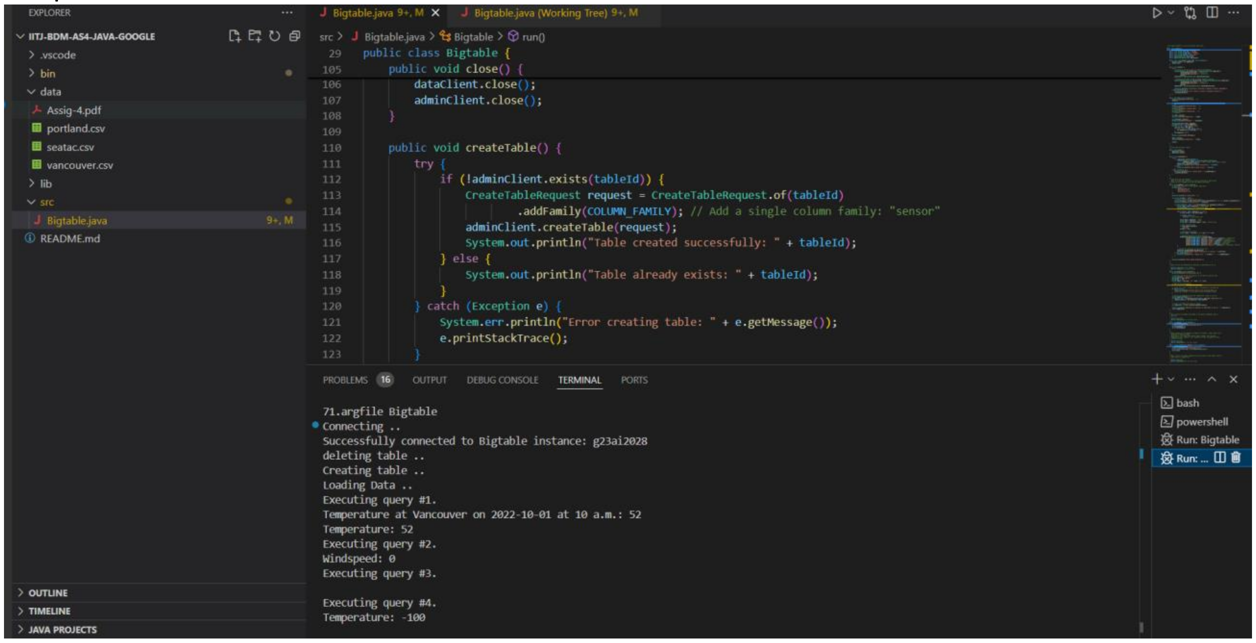


```
Loading Data ..
Loading Data...
Processing file: data/portland.csv
Data loaded successfully for station: portland
Processing file: data/seatac.csv
Data loaded successfully for station: seatac
Processing file: data/vancouver.csv
Data loaded successfully for station: vancouver
Data loading completed.
```

4. 10 marks - Write the method query1() that returns the temperature at Vancouver on 2022-10-01 at 10 a.m.

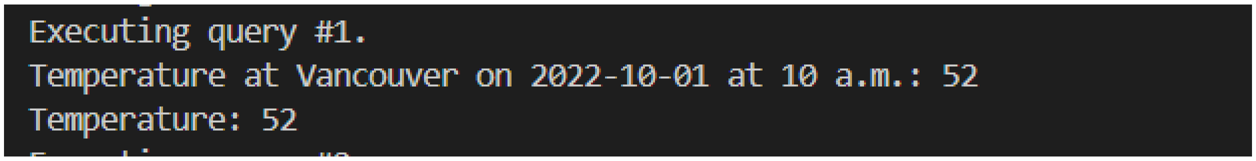
```java
/**
     * Query returns temperature at Vancouver on 2022-10-01 at 10 a.m.
     *
     * @return Temperature as an integer
     * @throws Exception if an error occurs
     */
    public int query1() throws Exception {
        System.out.println("Executing query #1.");

        // Constructing row key for Vancouver at specific date and time
        String stationId = "vancouver";
        String date = "2022-10-01";
        String hour = "10";
        String rowKey = stationId + "#" + date + "#" + hour;

        // Read row from Bigtable
        Row row = dataClient.readRow(tableId, rowKey);

        // If row is null, it means no data is available for given key
        if (row == null) {
            System.out.println("No data found for specified query.");
            return -1; // Return -1 or any other value to indicate no data
        }

        // Extracting temperature from "temperature" column in row
        String temperatureValue = "";
        for (RowCell cell : row.getCells(COLUMN_FAMILY, "temperature")) {
            temperatureValue = cell.getValue().toStringUtf8();
        }

        // Convert temperature value to integer
        int temperature = Integer.parseInt(temperatureValue);
        System.out.println("Temperature at Vancouver on 2022-10-01 at 10 a.m.: "
+ temperature);
        return temperature;
    }
```

Output



```java
public class Bigtable {
    public void close() {
        dataClient.close();
        adminClient.close();
    }

    public void createTable() {
        try {
            if (!adminClient.exists(tableId)) {
                CreateTableRequest request = CreateTableRequest.of(tableId)
                        .addFamily(COLUMN_FAMILY); // Add a single column family: "sensor"
                adminClient.createTable(request);
                System.out.println("Table created successfully: " + tableId);
            } else {
                System.out.println("Table already exists: " + tableId);
            }
        } catch (Exception e) {
            System.err.println("Error creating table: " + e.getMessage());
            e.printStackTrace();
        }
```

```
71.argfile Bigtable
Connecting ..
Successfully connected to Bigtable instance: g23ai2028
deleting table ..
Creating table ..
Loading Data ..
Executing query #1.
Temperature at Vancouver on 2022-10-01 at 10 a.m.: 52
Temperature: 52
Executing query #2.
Windspeed: 0
Executing query #3.

Executing query #4.
Temperature: -100
```

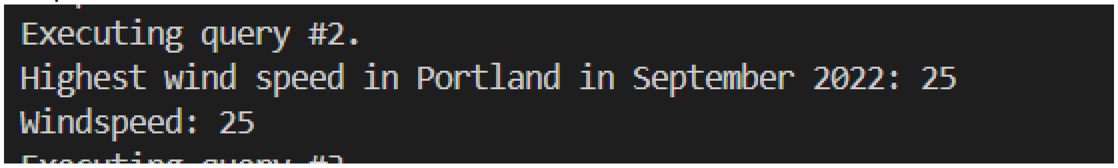


```
Executing query #1.
Temperature at Vancouver on 2022-10-01 at 10 a.m.: 52
Temperature: 52
```

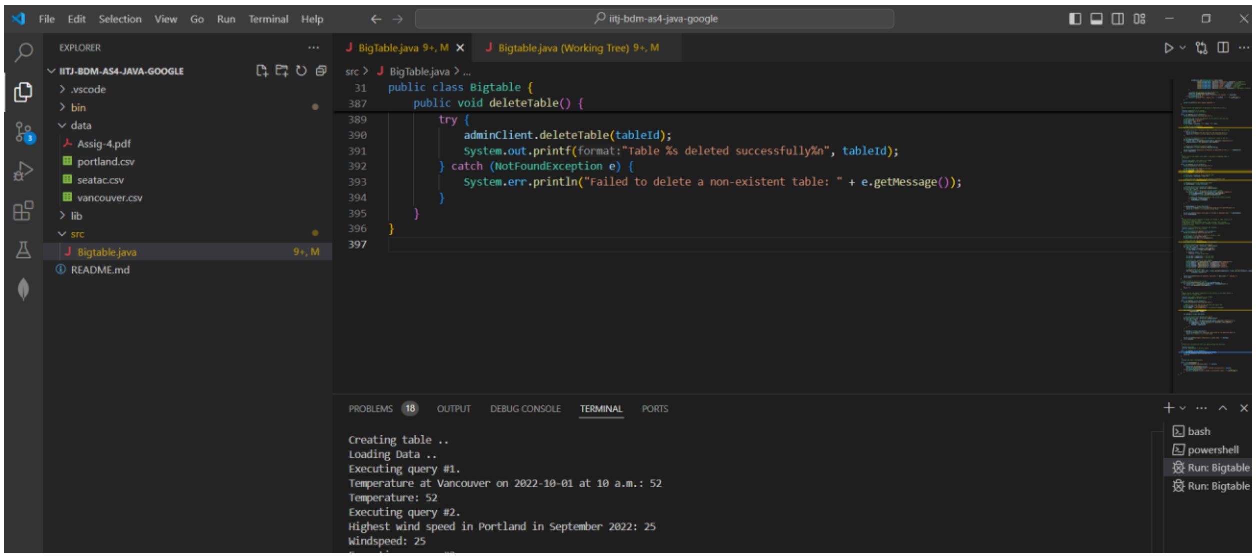
5. 5 marks - Write the method query2() that returns the highest wind speed in the month of September 2022 in Portland.

```java
/**
     * Query returns highest wind speed in month of September 2022 in
     * Portland.
     *
     * @return highest wind speed as an integer
     * @throws Exception if an error occurs
     */
    public int query2() throws Exception {
        System.out.println("Executing query #2.");

        // Station ID and date range for query
        String stationId = "portland";
        String startDate = "2022-09-01";
        String endDate = "2022-09-30";
        // Prefix for row keys to filter relevant data
        String prefix = stationId + "#2022-09";

        // Query rows with prefix for station in September 2022
        Query query = Query.create(tableId).prefix(prefix);

        // Variable to store highest wind speed
        int maxWindSpeed = Integer.MIN_VALUE;

        // Execute query and process results
        ServerStream<Row> rows = dataClient.readRows(query);
        for (Row row : rows) {
            // Extracting wind speed value from "speed" column
            for (RowCell cell : row.getCells(COLUMN_FAMILY, "speed")) {
                String windSpeedValue = cell.getValue().toStringUtf8();
                int windSpeed = Integer.parseInt(windSpeedValue);

                // Update maximum wind speed if current value is greater
                if (windSpeed > maxWindSpeed) {
                    maxWindSpeed = windSpeed;
                }
            }
        }

        if (maxWindSpeed == Integer.MIN_VALUE) {
            System.out.println("No wind speed data found for specified query.");
            return -1; // Return -1 to indicate no data found
        }

        System.out.println("Highest wind speed in Portland in September 2022: " +
maxWindSpeed);
        return maxWindSpeed;
    }
```

Output



```
Executing query #2.
Highest wind speed in Portland in September 2022: 25
Windspeed: 25
```



```java
public class Bigtable {
    public void deleteTable() {
        try {
            adminClient.deleteTable(tableId);
            System.out.printf(format:"Table %s deleted successfully%n", tableId);
        } catch (NotFoundException e) {
            System.err.println("Failed to delete a non-existent table: " + e.getMessage());
        }
    }
}
```

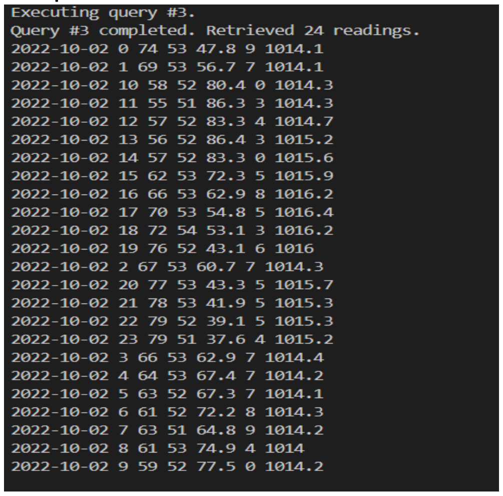
```
Creating table ..
Loading Data ..
Executing query #1.
Temperature at Vancouver on 2022-10-01 at 10 a.m.: 52
Temperature: 52
Executing query #2.
Highest wind speed in Portland in September 2022: 25
Windspeed: 25
```

6. 5 marks - Write the method query3() that returns all the readings for SeaTac for October 2, 2022.

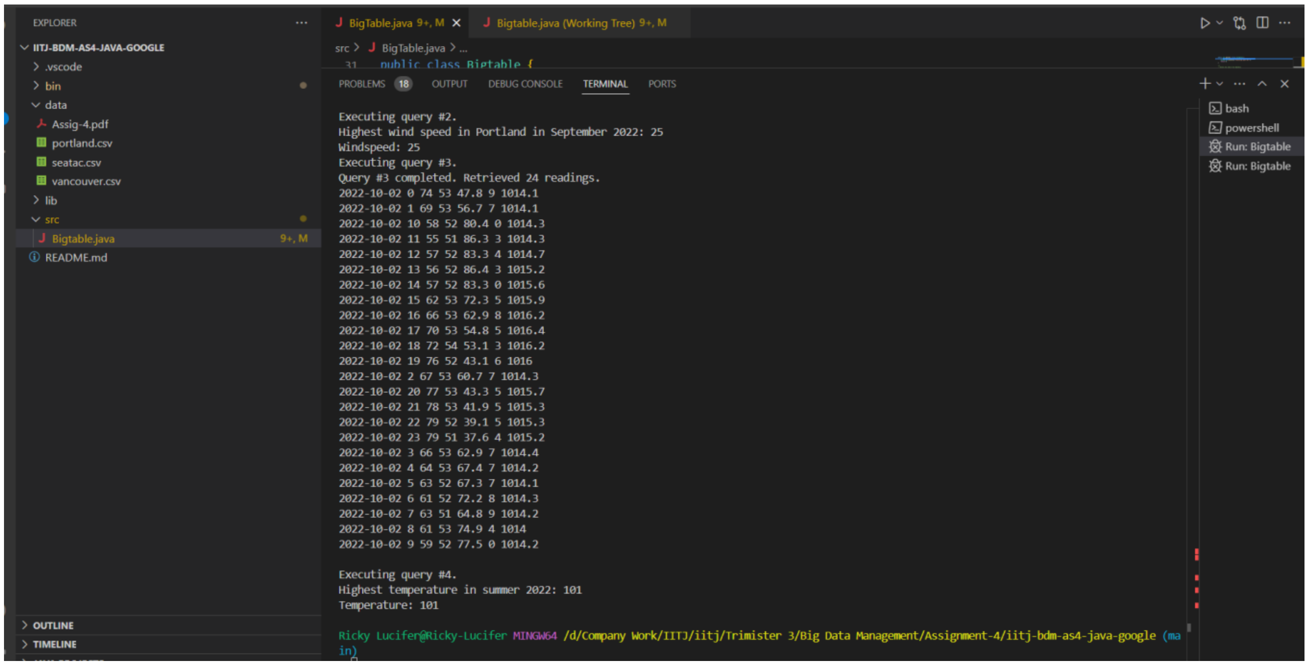
```java
/**
     * Query returns all readings for SeaTac for October 2, 2022. Return as an
     * ArrayList of objects arrays.
     * Each object array should have fields: date (string), hour (string),
     * temperature (int), dewpoint (int), humidity (string), windspeed (string),
     * pressure (string).
     *
     * @return ArrayList<Object[]> containing readings.
     * @throws Exception if an error occurs.
     */
    public ArrayList<Object[]> query3() throws Exception {
        System.out.println("Executing query #3.");

        // Prefix for row keys for SeaTac on October 2, 2022
        String prefix = "seatac#2022-10-02";
        ArrayList<Object[]> data = new ArrayList<>();

        // Quering rows with prefix
        Query query = Query.create(tableId).prefix(prefix);

        // Execute query and process results
        ServerStream<Row> rows = dataClient.readRows(query);
        for (Row row : rows) {
            String rowKey = row.getKey().toStringUtf8();
            String[] keyParts = rowKey.split("#");

            if (keyParts.length < 3)
```

```java
                continue; // Skip invalid keys

            String date = keyParts[1]; // Extracting date
            String hour = keyParts[2]; // Extracting hour

            // Extracting cell values for each reading
            String temperature = getCellValue(row, "temperature");
            String dewpoint = getCellValue(row, "dewpoint");
            String humidity = getCellValue(row, "relhum");
            String windspeed = getCellValue(row, "speed");
            String pressure = getCellValue(row, "pressure");

            // data to list
            data.add(new Object[] { date, hour, Integer.parseInt(temperature),
Integer.parseInt(dewpoint), humidity,
                    windspeed, pressure });
        }

        System.out.println("Query #3 completed. Retrieved " + data.size() + "
readings.");
        return data;
    }

    // method to extracting cell values
    private String getCellValue(Row row, String columnQualifier) {
        for (RowCell cell : row.getCells(COLUMN_FAMILY, columnQualifier)) {
            return cell.getValue().toStringUtf8();
        }
        return "";
    }
```

Output



```
Executing query #3.
Query #3 completed. Retrieved 24 readings.
2022-10-02 0 74 53 47.8 9 1014.1
2022-10-02 1 69 53 56.7 7 1014.1
2022-10-02 10 58 52 80.4 0 1014.3
2022-10-02 11 55 51 86.3 3 1014.3
2022-10-02 12 57 52 83.3 4 1014.7
2022-10-02 13 56 52 86.4 3 1015.2
2022-10-02 14 57 52 83.3 0 1015.6
2022-10-02 15 62 53 72.3 5 1015.9
2022-10-02 16 66 53 62.9 8 1016.2
2022-10-02 17 70 53 54.8 5 1016.4
2022-10-02 18 72 54 53.1 3 1016.2
2022-10-02 19 76 52 43.1 6 1016
2022-10-02 2 67 53 60.7 7 1014.3
2022-10-02 20 77 53 43.3 5 1015.7
2022-10-02 21 78 53 41.9 5 1015.3
2022-10-02 22 79 52 39.1 5 1015.3
2022-10-02 23 79 51 37.6 4 1015.2
2022-10-02 3 66 53 62.9 7 1014.4
2022-10-02 4 64 53 67.4 7 1014.2
2022-10-02 5 63 52 67.3 7 1014.1
2022-10-02 6 61 52 72.2 8 1014.3
2022-10-02 7 63 51 64.8 9 1014.2
2022-10-02 8 61 53 74.9 4 1014
2022-10-02 9 59 52 77.5 0 1014.2
```

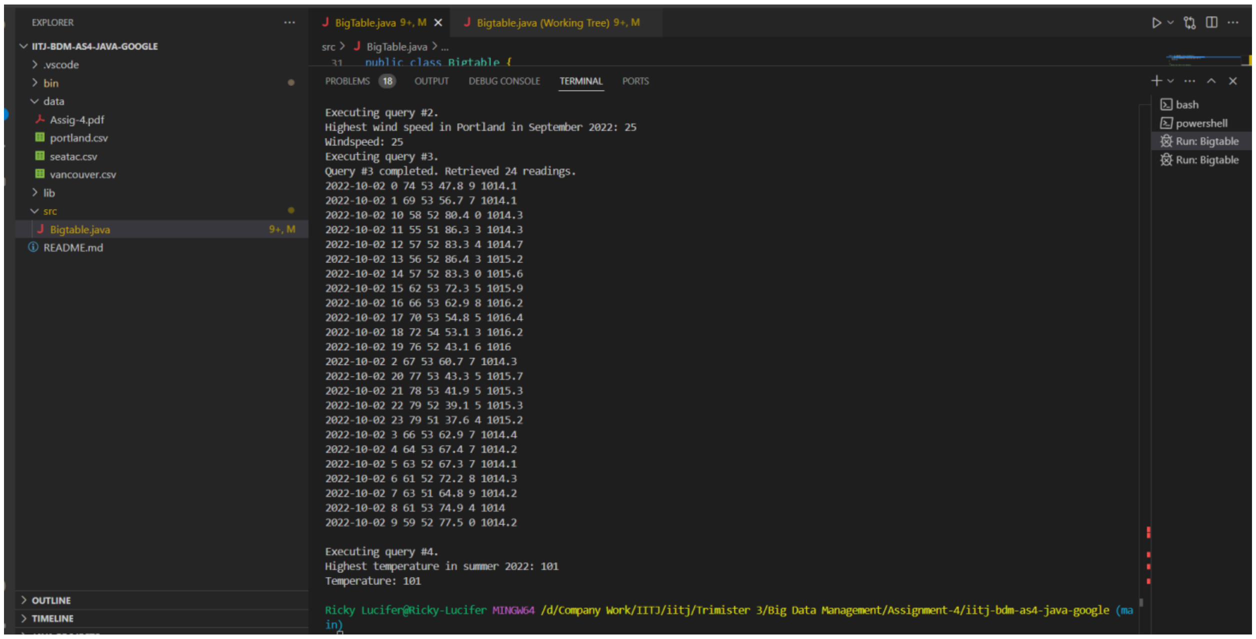


```
Executing query #2.
Highest wind speed in Portland in September 2022: 25
Windspeed: 25
Executing query #3.
Query #3 completed. Retrieved 24 readings.
2022-10-02 0 74 53 47.8 9 1014.1
2022-10-02 1 69 53 56.7 7 1014.1
2022-10-02 10 58 52 80.4 0 1014.3
2022-10-02 11 55 51 86.3 3 1014.3
2022-10-02 12 57 52 83.3 4 1014.7
2022-10-02 13 56 52 86.4 3 1015.2
2022-10-02 14 57 52 83.3 0 1015.6
2022-10-02 15 62 53 72.3 5 1015.9
2022-10-02 16 66 53 62.9 8 1016.2
2022-10-02 17 70 53 54.8 5 1016.4
2022-10-02 18 72 54 53.1 3 1016.2
2022-10-02 19 76 52 43.1 6 1016
2022-10-02 2 67 53 60.7 7 1014.3
2022-10-02 20 77 53 43.3 5 1015.7
2022-10-02 21 78 53 41.9 5 1015.3
2022-10-02 22 79 52 39.1 5 1015.3
2022-10-02 23 79 51 37.6 4 1015.2
2022-10-02 3 66 53 62.9 7 1014.4
2022-10-02 4 64 53 67.4 7 1014.2
2022-10-02 5 63 52 67.3 7 1014.1
2022-10-02 6 61 52 72.2 8 1014.3
2022-10-02 7 63 51 64.8 9 1014.2
2022-10-02 8 61 53 74.9 4 1014
2022-10-02 9 59 52 77.5 0 1014.2

Executing query #4.
Highest temperature in summer 2022: 101
Temperature: 101

Ricky Lucifer@Ricky-Lucifer MINGW64 /d/Company Work/IITJ/iitj/Trimister 3/Big Data Management/Assignment-4/iitj-bdm-as4-java-google (ma
in)
```

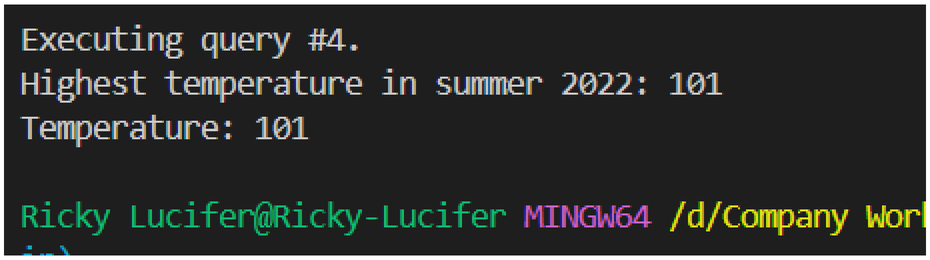
7. 5 marks - Write the method query4() that returns the highest temperature at any station in the summer months of 2022 (July (7), August (8)).

```java
/**
 * Query returns highest temperature at any station in summer months of
 * 2022 (July (7), August (8)).
 *
 * @return highest temperature as an integer.
 * @throws Exception if an error occurs.
 */
public int query4() throws Exception {
    System.out.println("Executing query #4.");

    // Manually set start and end keys for July and August 2022
    String startKey = "portland#2022-07";
    String endKey = "portland#2022-09"; // Exclusive of September

    Query query = Query.create(tableId)
            .range(startKey, endKey);

    int maxTemp = Integer.MIN_VALUE;

    //  query and process results
    ServerStream<Row> rows = dataClient.readRows(query);
    for (Row row : rows) {
            for (RowCell cell : row.getCells(COLUMN_FAMILY, "temperature")) {
                int temperature =
Integer.parseInt(cell.getValue().toStringUtf8());
                if (temperature > maxTemp) {
                    maxTemp = temperature;
                }
            }
        }

        if (maxTemp == Integer.MIN_VALUE) {
            System.out.println("No temperature data found for specified query.");
            return -1; // Returning -1 if no data found
        }

        System.out.println("Highest temperature in summer 2022: " + maxTemp);
        return maxTemp;
    }
```

Output



```
Executing query #2.
Highest wind speed in Portland in September 2022: 25
Windspeed: 25
Executing query #3.
Query #3 completed. Retrieved 24 readings.
2022-10-02 0 74 53 47.8 9 1014.1
2022-10-02 1 69 53 56.7 7 1014.1
2022-10-02 10 58 52 80.4 0 1014.3
2022-10-02 11 55 51 86.3 3 1014.3
2022-10-02 12 57 52 83.3 4 1014.7
2022-10-02 13 56 52 86.4 3 1015.2
2022-10-02 14 57 52 83.3 0 1015.6
2022-10-02 15 62 53 72.3 5 1015.9
2022-10-02 16 66 53 62.9 8 1016.2
2022-10-02 17 70 53 54.8 5 1016.4
2022-10-02 18 72 54 53.1 3 1016.2
2022-10-02 19 76 52 43.1 6 1016
2022-10-02 2 67 53 60.7 7 1014.3
2022-10-02 20 77 53 43.3 5 1015.7
2022-10-02 21 78 53 41.9 5 1015.3
2022-10-02 22 79 52 39.1 5 1015.3
2022-10-02 23 79 51 37.6 4 1015.2
2022-10-02 3 66 53 62.9 7 1014.4
2022-10-02 4 64 53 67.4 7 1014.2
2022-10-02 5 63 52 67.3 7 1014.1
2022-10-02 6 61 52 72.2 8 1014.3
2022-10-02 7 63 51 64.8 9 1014.2
2022-10-02 8 61 53 74.9 4 1014
2022-10-02 9 59 52 77.5 0 1014.2

Executing query #4.
Highest temperature in summer 2022: 101
Temperature: 101

Ricky Lucifer@Ricky-Lucifer MINGW64 /d/Company Work/IITJ/iitj/Trimister 3/Big Data Management/Assignment-4/iitj-bdm-as4-java-google (ma
in)
```

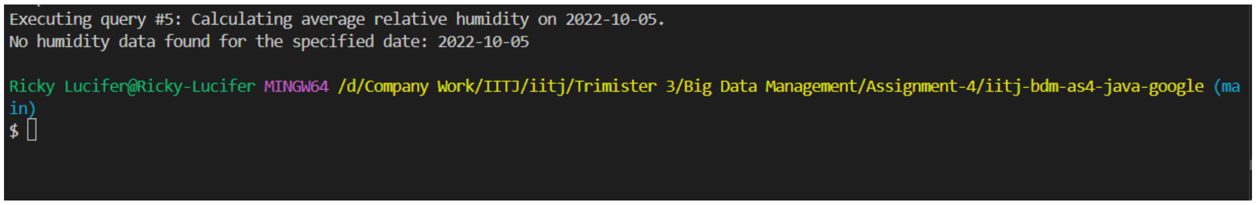


```
Executing query #4.
Highest temperature in summer 2022: 101
Temperature: 101

Ricky Lucifer@Ricky-Lucifer MINGW64 /d/Company Wor
```

8. Additional for Query 5

```
 /**
     * Query calculates average relative humidity for all stations on
     * 2022-10-05.
     *
     * @return average relative humidity as an integer.
     * @throws Exception if an error occurs.
     */
    public int query5() throws Exception {
        System.out.println("Executing query #5: Calculating average relative
humidity on 2022-10-05.");

        // Defining prefix for rows on specific date
        String date = "2022-10-05";
        String rowPrefix = "#" + date;

        // Building query
        Query query = Query.create(tableId)
                .prefix(rowPrefix);

        int totalHumidity = 0;
        int count = 0;

        // Executing query
        ServerStream<Row> rows = dataClient.readRows(query);
        for (Row row : rows) {
            for (RowCell cell : row.getCells(COLUMN_FAMILY, "relhum")) {
                int humidity = Integer.parseInt(cell.getValue().toStringUtf8());
                totalHumidity += humidity;
                count++;
            }
        }

        if (count == 0) {
            System.out.println("No humidity data found for specified date: " +
date);
            return -1; // Return -1 if no data is found
        }

        int avgHumidity = totalHumidity / count;
        System.out.println("Average relative humidity on " + date + ": " +
avgHumidity + "%");
        return avgHumidity;
    }

    /**
     * Delete table from Bigtable.
     */
    public void deleteTable() {
        System.out.println("\nDeleting table: " + tableId);
        try {
            adminClient.deleteTable(tableId);
            System.out.printf("Table %s deleted successfully%n", tableId);
        } catch (NotFoundException e) {
            System.err.println("Failed to delete a non-existent table: " +
e.getMessage());
        }
    }
```

Output



```
Executing query #5: Calculating average relative humidity on 2022-10-05.
No humidity data found for the specified date: 2022-10-05

Ricky Lucifer@Ricky-Lucifer MINGW64 /d/Company Work/IITJ/iitj/Trimister 3/Big Data Management/Assignment-4/iitj-bdm-as4-java-google (ma
in)
$ 
```

Full Code:

```
import com.google.api.gax.rpc.NotFoundException;
import com.google.api.gax.rpc.ServerStream;
import com.google.bigtable.v2.RowRange;
import com.google.cloud.bigtable.admin.v2.BigtableTableAdminClient;
import com.google.cloud.bigtable.admin.v2.BigtableTableAdminSettings;
import com.google.cloud.bigtable.admin.v2.models.CreateTableRequest;
import com.google.cloud.bigtable.data.v2.BigtableDataClient;
import com.google.cloud.bigtable.data.v2.BigtableDataSettings;
import com.google.cloud.bigtable.data.v2.models.BulkMutation;
import com.google.cloud.bigtable.data.v2.models.Mutation;
import com.google.cloud.bigtable.data.v2.models.Query;
import com.google.cloud.bigtable.data.v2.models.Row;
import com.google.cloud.bigtable.data.v2.models.RowCell;
import com.google.cloud.bigtable.data.v2.models.RowMutation;
import com.google.cloud.bigtable.data.v2.models.RowMutationEntry;
import com.google.protobuf.ByteString;
import java.util.logging.Logger;
import java.util.logging.Level;
import java.io.BufferedReader;
import java.io.File;
import java.io.FileReader;
import java.io.IOException;
import java.sql.SQLException;
```

```java
import java.util.ArrayList;
import java.util.List;

/*
 * Use Google Bigtable to store and analyze sensor data.
 */
public class Bigtable {
    // TODO: Fill in information for your database
    public final String projectId = "iitjdb";
    public final String instanceId = "ail7560";
    public final String COLUMN_FAMILY = "sensor";
    public final String tableId = "bdm_as4";
    public BigtableDataClient dataClient;
    public BigtableTableAdminClient adminClient;

    public static void main(String[] args) throws Exception {
        Bigtable testbt = new Bigtable();
        testbt.run();
    }

    public void connect() {
        try {
            // Settinging up data client to interact with Bigtable
            BigtableDataSettings dataSettings = BigtableDataSettings.newBuilder()
                    .setProjectId("g23ai2028") // Project ID
                    .setInstanceId("g23ai2028") // Instance ID
                    .build();
            dataClient = BigtableDataClient.create(dataSettings);

            // Settinging up admin client to manage Bigtable resources
            BigtableTableAdminSettings adminSettings =
BigtableTableAdminSettings.newBuilder()
                    .setProjectId("g23ai2028") // Project ID
                    .setInstanceId("g23ai2028") // Instance ID
                    .build();
            adminClient = BigtableTableAdminClient.create(adminSettings);

            System.out.println("Successfully connected to Bigtable instance:
g23ai2028");
        } catch (Exception e) {
            System.out.println("Error: Unable to connect to Bigtable instance.");
            e.printStackTrace();
        }
    }

    public void run() throws Exception {
        System.out.println("Connecting .. ");
        connect();

        // TODO: Comment or uncomment these as you proceed. Once load data,
comment them
        // out.
        System.out.println("deleting table .. ");
        deleteTable();
        System.out.println("Creating table .. ");
        createTable();
        System.out.println("Loading Data .. ");
        loadData();

        int temp = query1();
        System.out.println("Temperature: " + temp);

        int windspeed = query2();
        System.out.println("Windspeed: " + windspeed);

        ArrayList<Object[]> data = query3();
        StringBuffer buf = new StringBuffer();
        for (int i = 0; i < data.size(); i++) {
            Object[] vals = data.get(i);
            for (int j = 0; j < vals.length; j++) {
                buf.append(vals[j].toString() + " ");
            }
            buf.append("\n");
        }
        System.out.println(buf.toString());

        temp = query4();
        System.out.println("Temperature: " + temp);

        query5();
        close();
    }

    /**
     * Close data and admin clients
     */
    public void close() {
        dataClient.close();
        adminClient.close();
    }

    public void createTable() {
        try {
            if (!adminClient.exists(tableId)) {
                CreateTableRequest request = CreateTableRequest.of(tableId)
                        .addFamily(COLUMN_FAMILY); // Adding single column
family: "sensor"
                adminClient.createTable(request);
                System.out.println("Table created successfully: " + tableId);
            } else {
                System.out.println("Table already exists: " + tableId);
            }
        } catch (Exception e) {
            System.err.println("Error creating table: " + e.getMessage());
            e.printStackTrace();
        }
    }

    /**
     * Loads data into database.
     * Data is in CSV files. Note that it must be converted to hourly data.
     * Takes first reading in an hour and ignores any others.
     */
    public void loadData() throws Exception {
        // Hardcoding  files paths for specific data files so no mismathc happns
        String[] files = {
                "data/portland.csv",
                "data/seatac.csv",
                "data/vancouver.csv"
        };

        System.out.println("Loading Data...");

        for (String filePath : files) {
            // Extracting station ID from file name
            String stationId = filePath.substring(filePath.lastIndexOf("/") + 1,
filePath.lastIndexOf("."));
            System.out.println("Processing file: " + filePath);

            try (BufferedReader reader = new BufferedReader(new
FileReader(filePath))) {
                String line = reader.readLine(); // Skiping header row so we can
save data
                String lastHour = ""; // To track hourly data
                BulkMutation bulkMutation = BulkMutation.create(tableId); // Use
bulk mutation for better detail view

                while ((line = reader.readLine()) != null) {
                    String[] fields = line.split(",");

                    if (fields.length < 9)
```

```
                        continue; // Skiping invalid rows

                    String date = fields[1]; // Date
                    String time = fields[2]; // Time
                    String hour = time.split(":")[0]; // Extracting hour

                    // Skiping non-hourly data
                    if (hour.equals(lastHour))
                        continue;
                    lastHour = hour;

                    // Constructing row key
                    String rowKey = stationId + "#" + date + "#" + hour;

                    // Adding mutation entry for current row
                    bulkMutation.add(RowMutationEntry.create(rowKey)
                            .setCell(COLUMN_FAMILY, "temperature", fields[3]) //
Temperature
                            .setCell(COLUMN_FAMILY, "dewpoint", fields[4]) //
Dewpoint
                            .setCell(COLUMN_FAMILY, "relhum", fields[5]) //
Relative Humidity
                            .setCell(COLUMN_FAMILY, "speed", fields[6]) // Wind
Speed
                            .setCell(COLUMN_FAMILY, "gust", fields[7]) // Wind
Gust
                            .setCell(COLUMN_FAMILY, "pressure", fields[8])); //
Atmospheric Pressure
                }

                // Executing all mutations for current file
                dataClient.bulkMutateRows(bulkMutation);
                System.out.println("Data loaded successfully for station: " +
stationId);
            } catch (IOException e) {
                System.err.println("Error reading file: " + filePath + " - " +
e.getMessage());
            }
        }

        System.out.println("Data loading completed.");
    }

    /**
     * Query returns temperature at Vancouver on 2022-10-01 at 10 a.m.
     *
     * @return Temperature as an integer
```

```java
     * @throws Exception if an error occurs
     */
    public int query1() throws Exception {
        System.out.println("Executing query #1.");

        // Constructing row key for Vancouver at specific date and time
        String stationId = "vancouver";
        String date = "2022-10-01";
        String hour = "10";
        String rowKey = stationId + "#" + date + "#" + hour;

        // Read row from Bigtable
        Row row = dataClient.readRow(tableId, rowKey);

        // If row is null, it means no data is available for given key
        if (row == null) {
            System.out.println("No data found for specified query.");
            return -1; // Return -1 or any other value to indicate no data
        }

        // Extracting temperature from "temperature" column in row
        String temperatureValue = "";
        for (RowCell cell : row.getCells(COLUMN_FAMILY, "temperature")) {
            temperatureValue = cell.getValue().toStringUtf8();
        }

        // Convert temperature value to integer
        int temperature = Integer.parseInt(temperatureValue);

        System.out.println("Temperature at Vancouver on 2022-10-01 at 10 a.m.: "
+ temperature);
        return temperature;
    }

    /**
     * Query returns highest wind speed in month of September 2022 in
     * Portland.
     *
     * @return highest wind speed as an integer
     * @throws Exception if an error occurs
     */
    public int query2() throws Exception {
        System.out.println("Executing query #2.");

        // Station ID and date range for query
        String stationId = "portland";
        String startDate = "2022-09-01";
        String endDate = "2022-09-30";

        // Prefix for row keys to filter relevant data
        String prefix = stationId + "#2022-09";

        // Query rows with prefix for station in September 2022
        Query query = Query.create(tableId).prefix(prefix);

        // Variable to store highest wind speed
        int maxWindSpeed = Integer.MIN_VALUE;

        // Execute query and process results
        ServerStream<Row> rows = dataClient.readRows(query);
        for (Row row : rows) {
            // Extracting wind speed value from "speed" column
            for (RowCell cell : row.getCells(COLUMN_FAMILY, "speed")) {
                String windSpeedValue = cell.getValue().toStringUtf8();
                int windSpeed = Integer.parseInt(windSpeedValue);

                // Update maximum wind speed if current value is greater
                if (windSpeed > maxWindSpeed) {
                    maxWindSpeed = windSpeed;
                }
            }
        }

        if (maxWindSpeed == Integer.MIN_VALUE) {
            System.out.println("No wind speed data found for specified query.");
            return -1; // Return -1 to indicate no data found
        }

        System.out.println("Highest wind speed in Portland in September 2022: " +
maxWindSpeed);
        return maxWindSpeed;
    }

    /**
     * Query returns all readings for SeaTac for October 2, 2022. Return as an
     * ArrayList of objects arrays.
     * Each object array should have fields: date (string), hour (string),
     * temperature (int), dewpoint (int), humidity (string), windspeed (string),
     * pressure (string).
     *
     * @return ArrayList<Object[]> containing readings.
     * @throws Exception if an error occurs.
     */
    public ArrayList<Object[]> query3() throws Exception {
        System.out.println("Executing query #3.");

        // Prefix for row keys for SeaTac on October 2, 2022
        String prefix = "seatac#2022-10-02";
        ArrayList<Object[]> data = new ArrayList<>();

        // Quering rows with prefix
        Query query = Query.create(tableId).prefix(prefix);

        // Execute query and process results
        ServerStream<Row> rows = dataClient.readRows(query);
        for (Row row : rows) {
            String rowKey = row.getKey().toStringUtf8();
            String[] keyParts = rowKey.split("#");

            if (keyParts.length < 3)
                continue; // Skip invalid keys

            String date = keyParts[1]; // Extracting date
            String hour = keyParts[2]; // Extracting hour

            // Extracting cell values for each reading
            String temperature = getCellValue(row, "temperature");
            String dewpoint = getCellValue(row, "dewpoint");
            String humidity = getCellValue(row, "relhum");
            String windspeed = getCellValue(row, "speed");
            String pressure = getCellValue(row, "pressure");

            // data to list
            data.add(new Object[] { date, hour, Integer.parseInt(temperature),
Integer.parseInt(dewpoint), humidity,
                    windspeed, pressure });
        }

        System.out.println("Query #3 completed. Retrieved " + data.size() + "
readings.");
        return data;
    }

    // method to extracting cell values
    private String getCellValue(Row row, String columnQualifier) {
        for (RowCell cell : row.getCells(COLUMN_FAMILY, columnQualifier)) {
            return cell.getValue().toStringUtf8();
        }
        return "";
    }
    /**
     * Query returns highest temperature at any station in summer months of
     * 2022 (July (7), August (8)).
     *
     * @return highest temperature as an integer.
     * @throws Exception if an error occurs.
     */
    public int query4() throws Exception {
        System.out.println("Executing query #4.");

        // Manually set start and end keys for July and August 2022
        String startKey = "portland#2022-07";
        String endKey = "portland#2022-09"; // Exclusive of September

        Query query = Query.create(tableId)
                .range(startKey, endKey);

        int maxTemp = Integer.MIN_VALUE;

        //  query and process results
        ServerStream<Row> rows = dataClient.readRows(query);
        for (Row row : rows) {
            for (RowCell cell : row.getCells(COLUMN_FAMILY, "temperature")) {
                int temperature =
Integer.parseInt(cell.getValue().toStringUtf8());
                if (temperature > maxTemp) {
                    maxTemp = temperature;
                }
            }
        }

        if (maxTemp == Integer.MIN_VALUE) {
            System.out.println("No temperature data found for specified query.");
            return -1; // Returning -1 if no data found
        }

        System.out.println("Highest temperature in summer 2022: " + maxTemp);
        return maxTemp;
    }

    /**
     * Query calculates average relative humidity for all stations on
     * 2022-10-05.
     *
     * @return average relative humidity as an integer.
     * @throws Exception if an error occurs.
     */
    public int query5() throws Exception {
        System.out.println("Executing query #5: Calculating average relative
humidity on 2022-10-05.");

        // Defining prefix for rows on specific date
        String date = "2022-10-05";
        String rowPrefix = "#" + date;

        // Building query
        Query query = Query.create(tableId)
                .prefix(rowPrefix);

        int totalHumidity = 0;
        int count = 0;

        // Executing query
        ServerStream<Row> rows = dataClient.readRows(query);
        for (Row row : rows) {
            for (RowCell cell : row.getCells(COLUMN_FAMILY, "relhum")) {
                int humidity = Integer.parseInt(cell.getValue().toStringUtf8());
                totalHumidity += humidity;
                count++;
            }
        }

        if (count == 0) {
            System.out.println("No humidity data found for specified date: " +
date);
            return -1; // Return -1 if no data is found
        }

        int avgHumidity = totalHumidity / count;
        System.out.println("Average relative humidity on " + date + ": " +
avgHumidity + "%");
        return avgHumidity;
    }

    /**
     * Delete table from Bigtable.
     */
    public void deleteTable() {
        System.out.println("\nDeleting table: " + tableId);
        try {
            adminClient.deleteTable(tableId);
            System.out.printf("Table %s deleted successfully%n", tableId);
        } catch (NotFoundException e) {
            System.err.println("Failed to delete a non-existent table: " +
e.getMessage());
        }
    }
}
```

Output

```
cd d:\\Company\ Work\\IITJ\\iitj\\Trimister\ 3\\Big\ Data\ Management\\Assignment-4\\iitj-bdm-
as4-java-google ; /usr/bin/env C:\\Program\ Files\\Java\\jdk-23\\bin\\java.exe
@C:\\Users\\RICKYL~1\\AppData\\Local\\Temp\\cp_blf301b7l4eka0wwqoh5tq1b7.argfile
Bigtable
Connecting ..
Successfully connected to Bigtable instance: g23ai2028
deleting table ..

Deleting table: bdm_as4
Table bdm_as4 deleted successfully
Creating table ..
Table created successfully: bdm_as4
Loading Data ..
Loading Data...
Processing file: data/portland.csv
Data loaded successfully for station: portland
Processing file: data/seatac.csv
Data loaded successfully for station: seatac
Processing file: data/vancouver.csv
Data loaded successfully for station: vancouver
Data loading completed.
Creating table ..
Loading Data ..
Executing query #1.
Temperature at Vancouver on 2022-10-01 at 10 a.m.: 52
Temperature: 52
Executing query #2.
Highest wind speed in Portland in September 2022: 25
Windspeed: 25
Executing query #3.
Query #3 completed. Retrieved 24 readings.
2022-10-02 0 74 53 47.8 9 1014.1
2022-10-02 1 69 53 56.7 7 1014.1
2022-10-02 10 58 52 80.4 0 1014.3
2022-10-02 11 55 51 86.3 3 1014.3
```

```
2022-10-02 12 57 52 83.3 4 1014.7
2022-10-02 13 56 52 86.4 3 1015.2
2022-10-02 14 57 52 83.3 0 1015.6
2022-10-02 15 62 53 72.3 5 1015.9
2022-10-02 16 66 53 62.9 8 1016.2
2022-10-02 17 70 53 54.8 5 1016.4
2022-10-02 18 72 54 53.1 3 1016.2
2022-10-02 19 76 52 43.1 6 1016
2022-10-02 2 67 53 60.7 7 1014.3
2022-10-02 20 77 53 43.3 5 1015.7
2022-10-02 21 78 53 41.9 5 1015.3
2022-10-02 22 79 52 39.1 5 1015.3
2022-10-02 23 79 51 37.6 4 1015.2
2022-10-02 3 66 53 62.9 7 1014.4
2022-10-02 4 64 53 67.4 7 1014.2
2022-10-02 5 63 52 67.3 7 1014.1
2022-10-02 6 61 52 72.2 8 1014.3
2022-10-02 7 63 51 64.8 9 1014.2
2022-10-02 8 61 53 74.9 4 1014
2022-10-02 9 59 52 77.5 0 1014.2

Executing query #4.
Highest temperature in summer 2022: 101
Temperature: 101
Executing query #5: Calculating average relative humidity on 2022-10-05.
No humidity data found for the specified date: 2022-10-05
```

Thank you sir for such a good hands on Assignment 4.
Regards

Shubham Raj
Roll No : G23AI2028