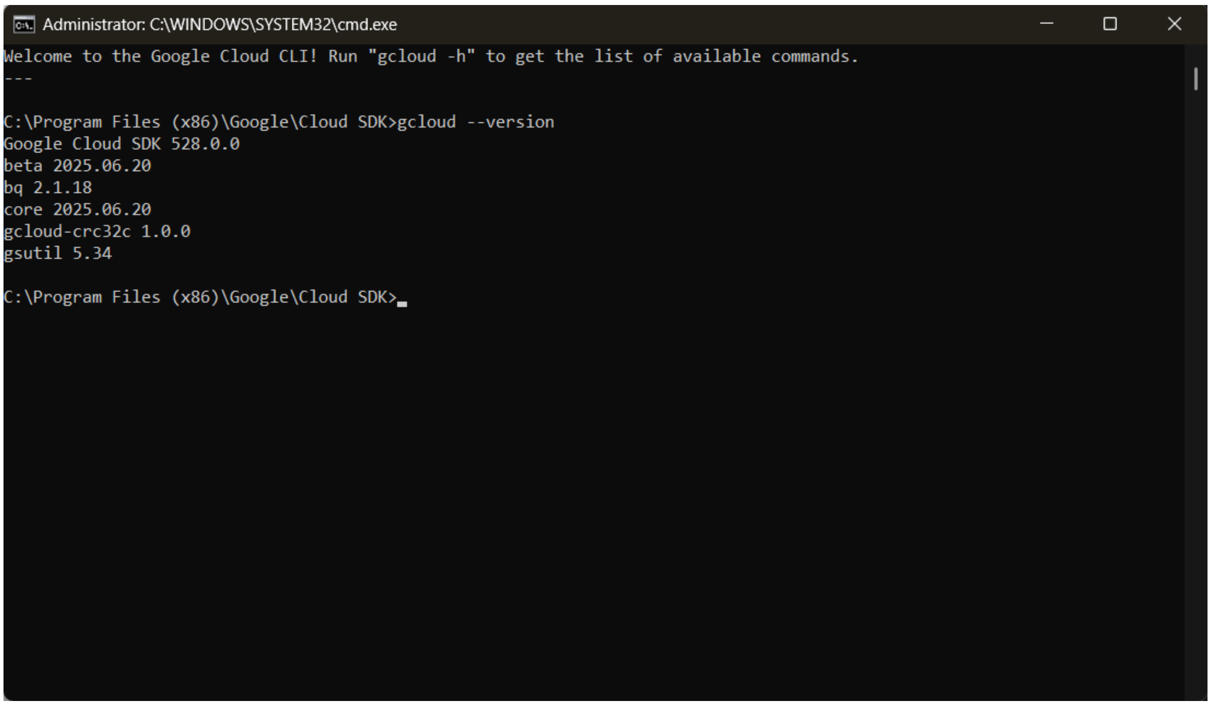
Big Data Management - Assignment 4

Google Big Table

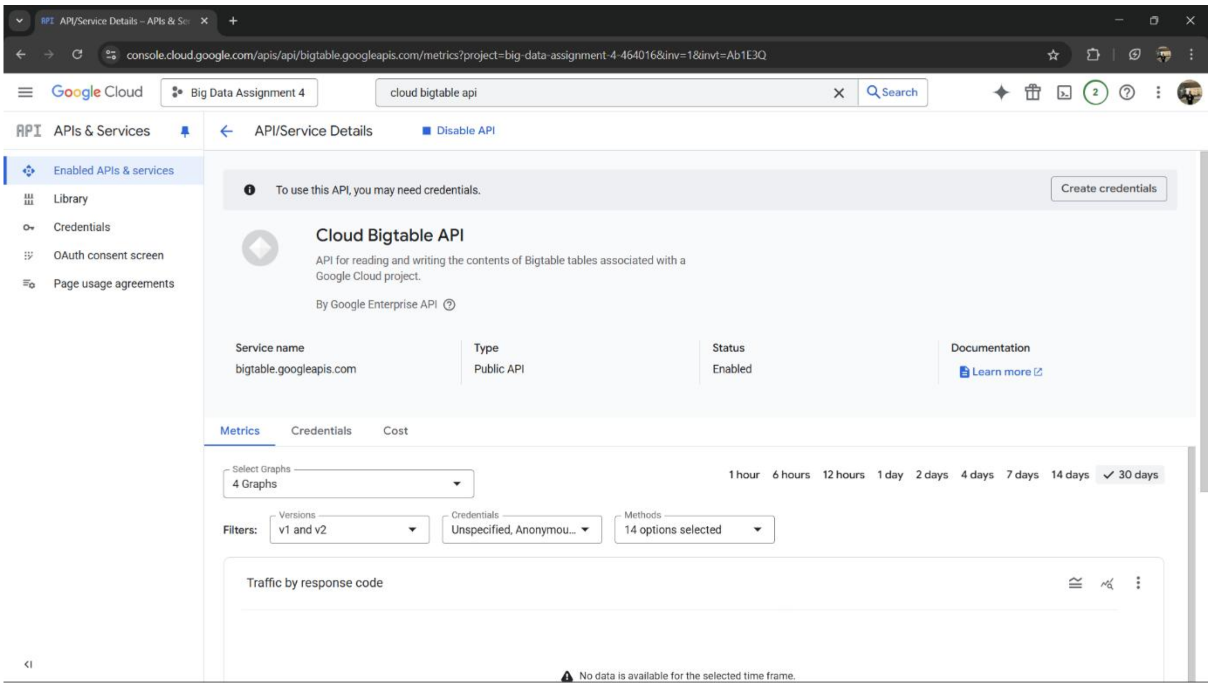
Akshay Kumar (G24AI1033)

Step 1 : Installing Google Cloud Command Line Interface (CLI) that allows our local machine to communicate with Google Cloud services, including Bigtable.



```
Welcome to the Google Cloud CLI! Run "gcloud -h" to get the list of available commands.
---

C:\Program Files (x86)\Google\Cloud SDK>gcloud --version
Google Cloud SDK 528.0.0
beta 2025.06.20
bq 2.1.18
core 2025.06.20
gcloud-crc32c 1.0.0
gsutil 5.34

C:\Program Files (x86)\Google\Cloud SDK>
```

Step 2 : Creating a Google Cloud Project and Enable Bigtable API



To use this API, you may need credentials.

Cloud Bigtable API

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

By Google Enterprise API

Service name: bigtable.googleapis.com
Type: Public API
Status: Enabled
Documentation: Learn more

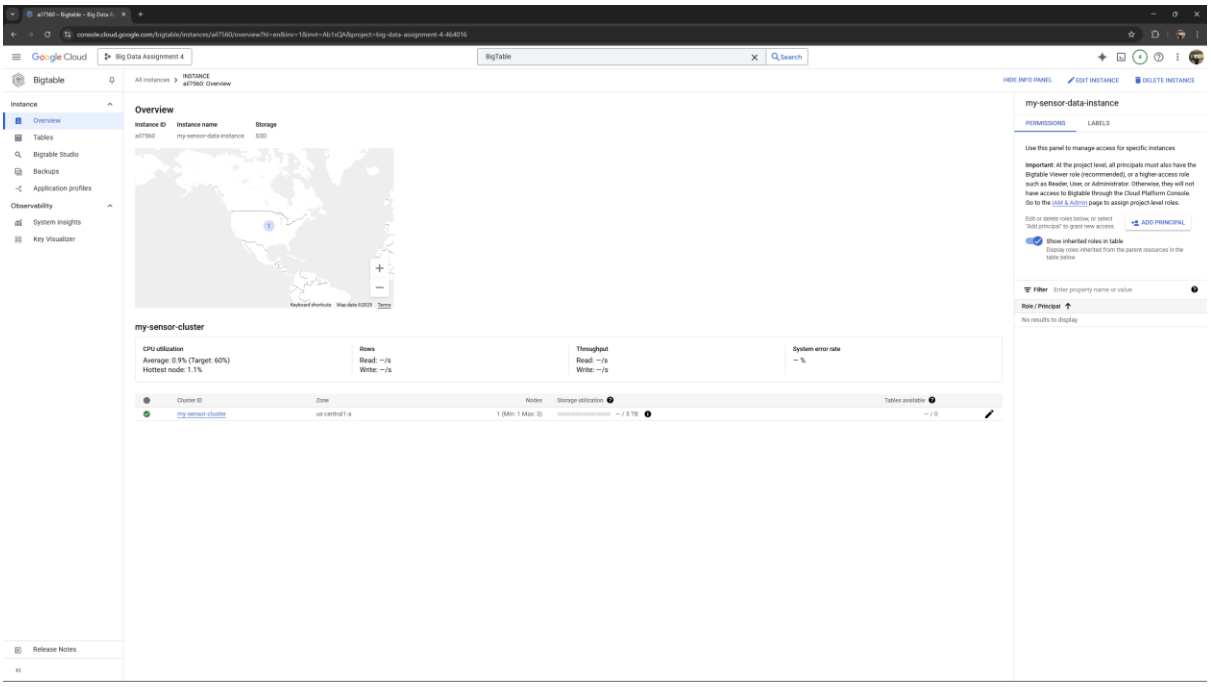
Select Graphs: 4 Graphs

Filters: Versions: v1 and v2 | Credentials: Unspecified, Anonymou... | Methods: 14 options selected

Traffic by response code

No data is available for the selected time frame.

Step 3 : Creating a Bigtable Instance



Overview

Instance ID	Instance name	Storage
ail7560	my-sensor-data-instance	SSD

my-sensor-cluster

CPU utilization	Rows	Throughput	System error rate
Average: 0.9% (Target: 60%) Hottest node: 1.1%	Read: —/s Write: —/s	Read: —/s Write: —/s	— %

●	Cluster ID	Zone	Nodes	Storage utilization	Tables available
✓	my-sensor-cluster	us-central1-a	1 (Min: 1 Max: 3)	— / 5 TB	— / 0

my-sensor-data-instance

Use this panel to manage access for specific instances

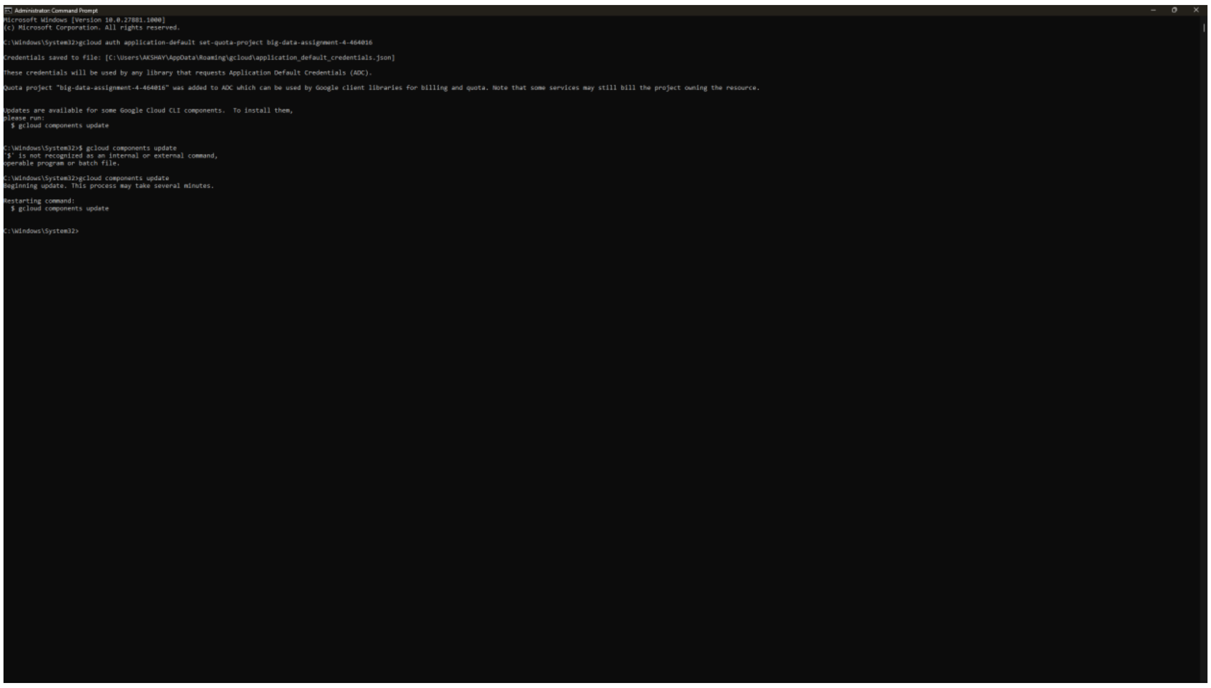
Important: At the project level, all principals must also have the Bigtable Viewer role (recommended), or a higher-access role such as Reader, User, or Administrator. Otherwise, they will not have access to Bigtable through the Cloud Platform Console. Go to the IAM & Admin page to assign project-level roles.

Edit or delete roles below, or select "Add principal" to grant new access.

Show inherited roles in table
Display roles inherited from the parent resources in the table below

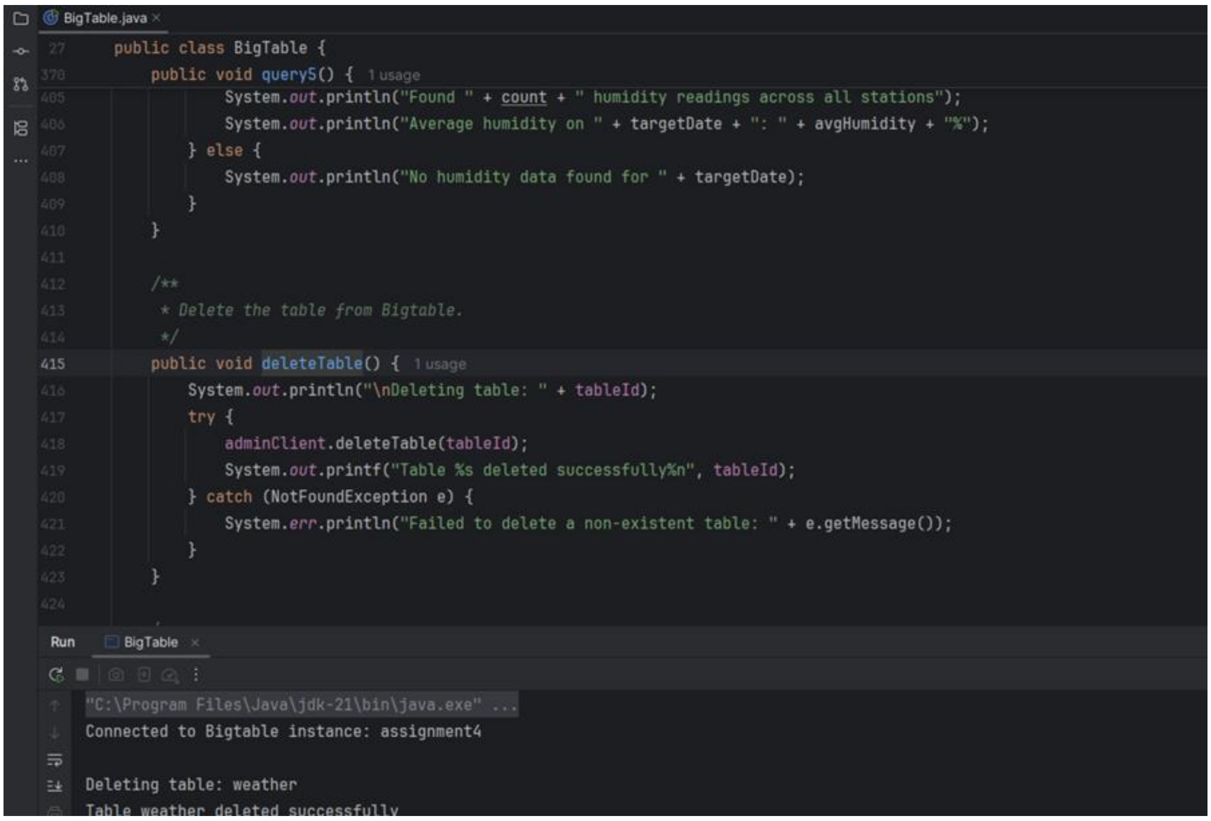
Role / Principal
No results to display

Step 4 :



```
Microsoft Windows [Version 10.0.27881.1000]
(c) Microsoft Corporation. All rights reserved.

C:\Windows\System32>gcloud auth application-default set-quota-project big-data-assignment-4-464016

Credentials saved to file: [C:\Users\AKSHAY\AppData\Roaming\gcloud\application_default_credentials.json]

These credentials will be used by any library that requests Application Default Credentials (ADC).

Quota project "big-data-assignment-4-464016" was added to ADC which can be used by Google client libraries for billing and quota. Note that some services may still bill the project owning the resource.


Updates are available for some Google Cloud CLI components.  To install them,
please run:
  $ gcloud components update


C:\Windows\System32>$ gcloud components update
'$' is not recognized as an internal or external command,
operable program or batch file.

C:\Windows\System32>gcloud components update
Beginning update. This process may take several minutes.

Restarting command:
  $ gcloud components update


C:\Windows\System32>
```

deleteTable()

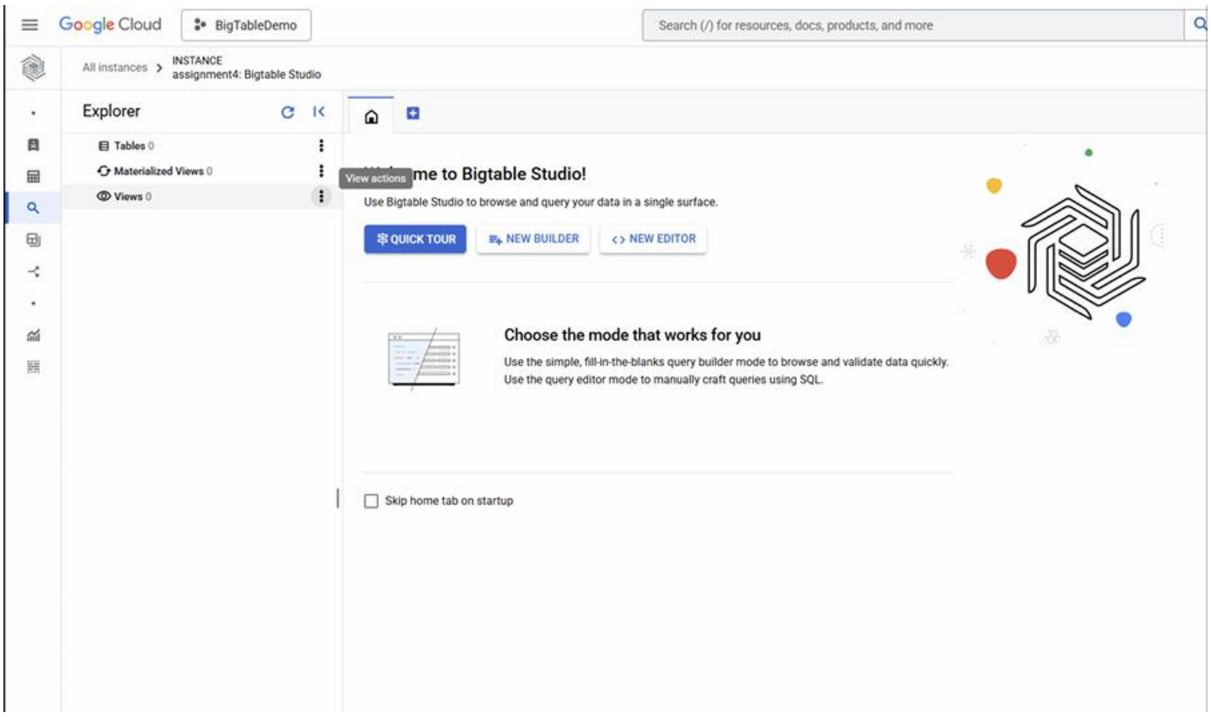


BigTable.java

```
public class BigTable {
    public void query5() {  1 usage
            System.out.println("Found " + count + " humidity readings across all stations");
            System.out.println("Average humidity on " + targetDate + ": " + avgHumidity + "%");
        } else {
            System.out.println("No humidity data found for " + targetDate);
        }
    }

    /**
     * Delete the table from Bigtable.
     */
    public void deleteTable() {  1 usage
        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());
        }
    }
```

Run BigTable

```
"C:\Program Files\Java\jdk-21\bin\java.exe" ...
Connected to Bigtable instance: assignment4

Deleting table: weather
Table weather deleted successfully
```



Google Cloud BigTableDemo

Search (/) for resources, docs, products, and more

All instances > INSTANCE assignment4: Bigtable Studio

Explorer

Tables 0

Materialized Views 0

Views 0

View actions me to Bigtable Studio!

Use Bigtable Studio to browse and query your data in a single surface.

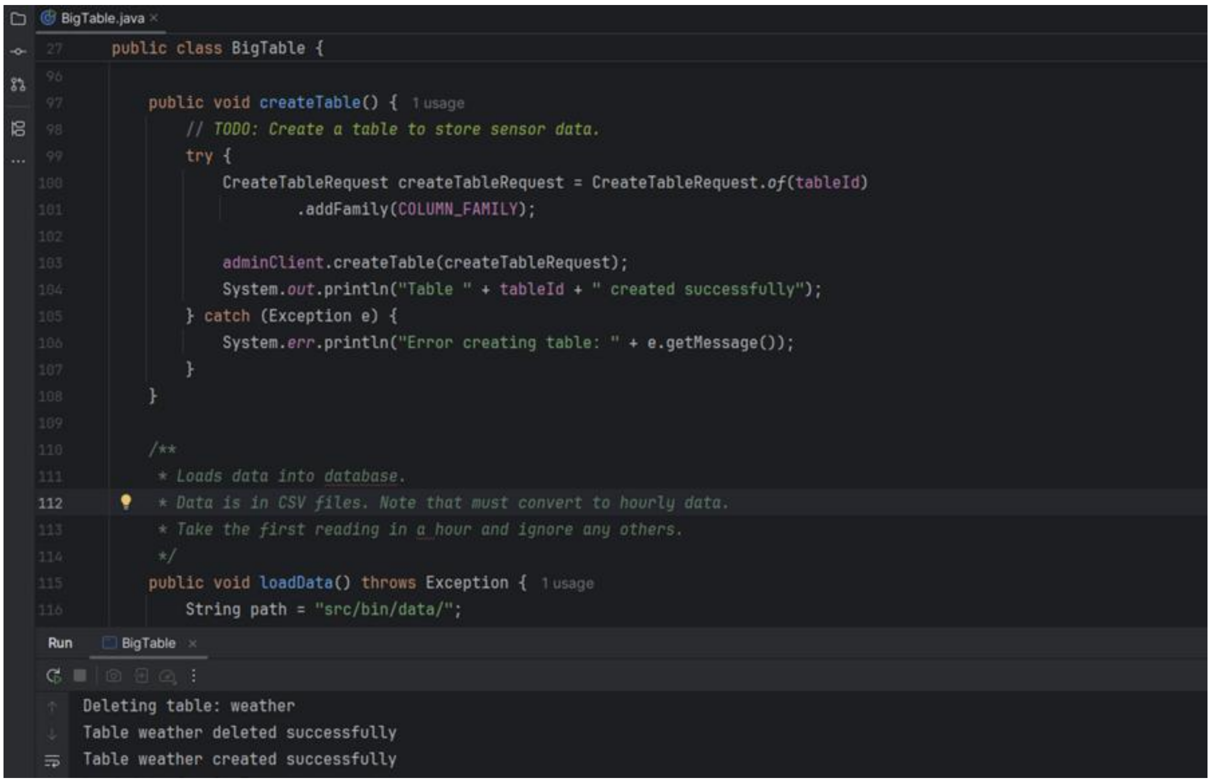
QUICK TOUR NEW BUILDER NEW EDITOR

Choose the mode that works for you

Use the simple, fill-in-the-blanks query builder mode to browse and validate data quickly.
Use the query editor mode to manually craft queries using SQL.

Skip home tab on startup

createTable()

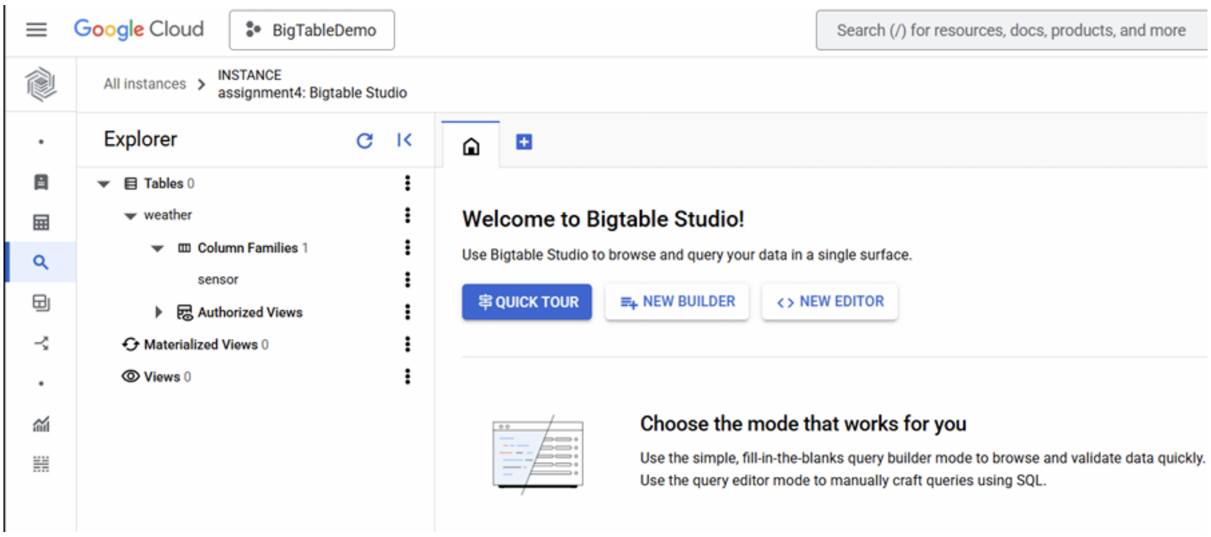


BigTable.java

```java
public class BigTable {

    public void createTable() {  1 usage
        // TODO: Create a table to store sensor data.
        try {
            CreateTableRequest createTableRequest = CreateTableRequest.of(tableId)
                    .addFamily(COLUMN_FAMILY);

            adminClient.createTable(createTableRequest);
            System.out.println("Table " + tableId + " created successfully");
        } catch (Exception e) {
            System.err.println("Error creating table: " + e.getMessage());
        }
    }

    /**
     * Loads data into database.
     * Data is in CSV files. Note that must convert to hourly data.
     * Take the first reading in a hour and ignore any others.
     */
    public void loadData() throws Exception {  1 usage
        String path = "src/bin/data/";
```

Run   BigTable

```
Deleting table: weather
Table weather deleted successfully
Table weather created successfully
```



Google Cloud   BigTableDemo   Search (/) for resources, docs, products, and more

All instances > INSTANCE
assignment4: Bigtable Studio

Explorer

- Tables 0
  - weather
    - Column Families 1
      - sensor
    - Authorized Views
  - Materialized Views 0
  - Views 0

Welcome to Bigtable Studio!

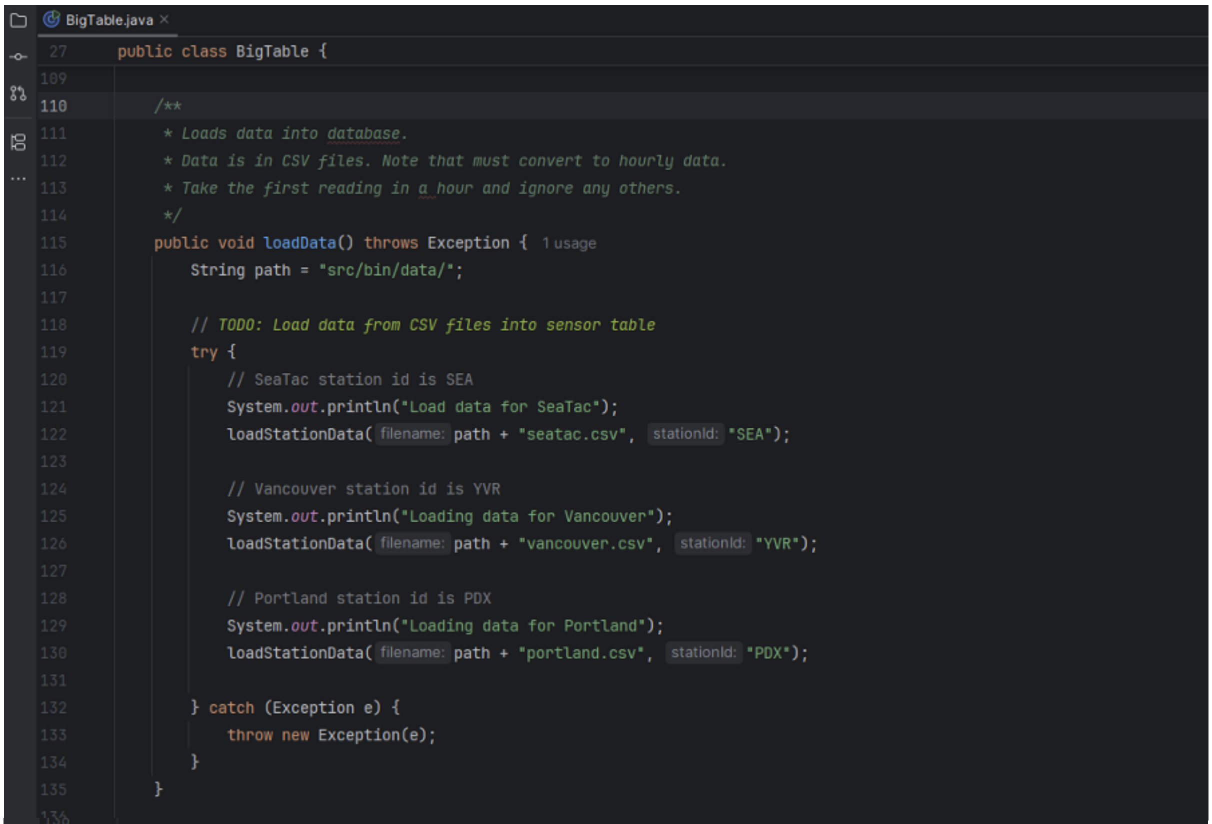
Use Bigtable Studio to browse and query your data in a single surface.

QUICK TOUR   NEW BUILDER   NEW EDITOR

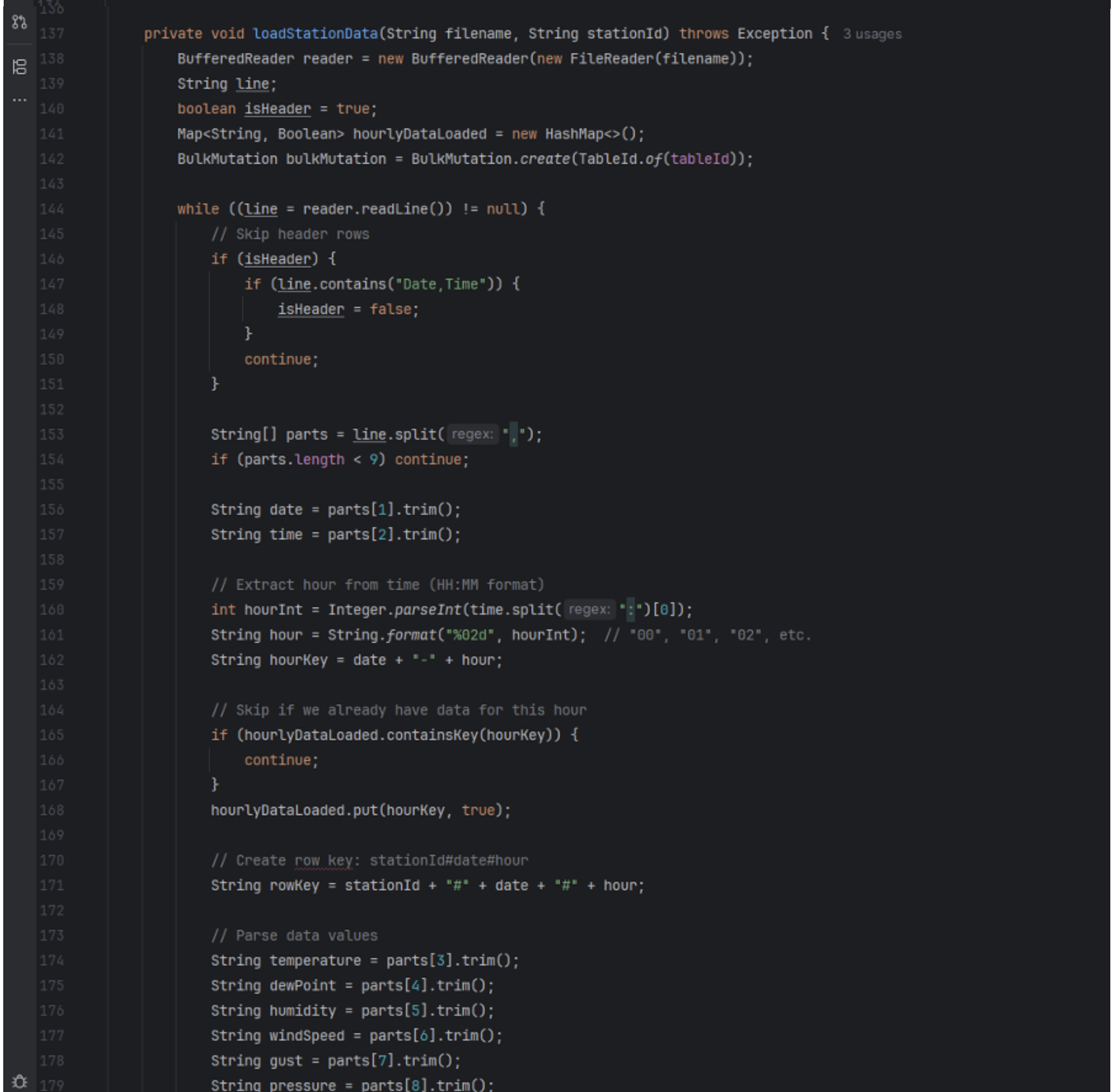
Choose the mode that works for you

Use the simple, fill-in-the-blanks query builder mode to browse and validate data quickly.
Use the query editor mode to manually craft queries using SQL.

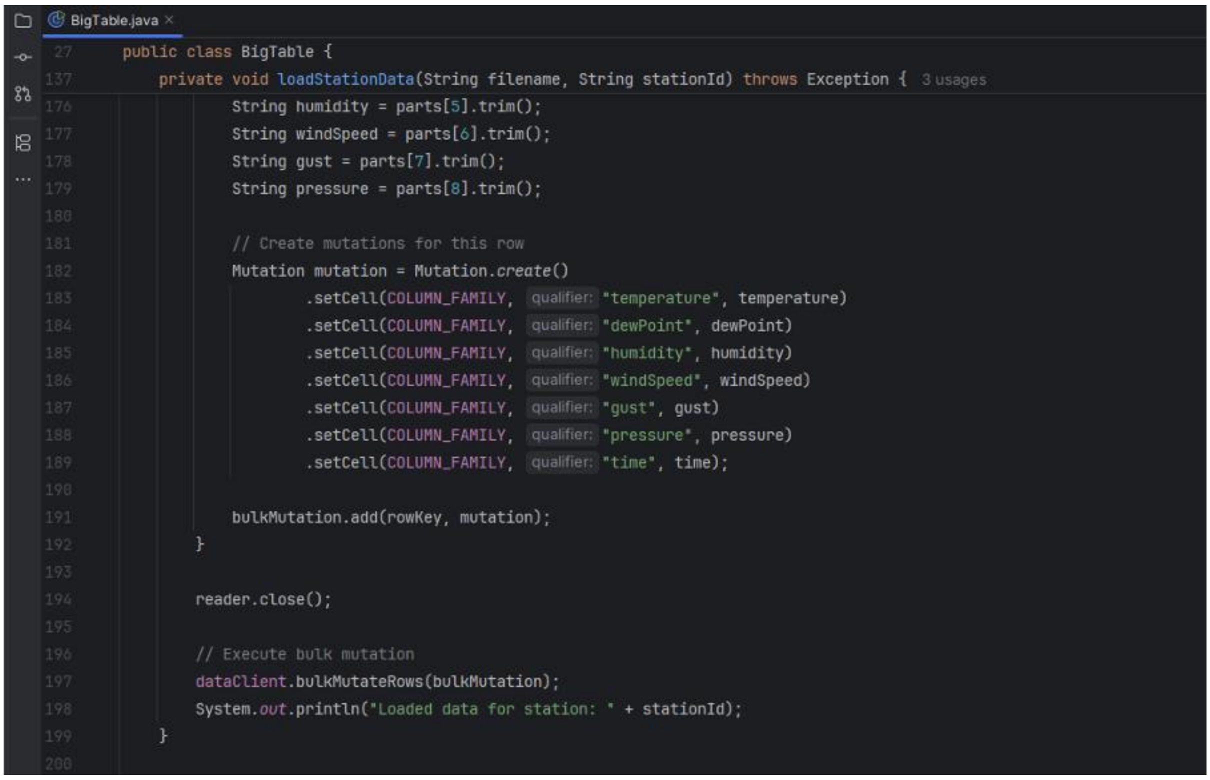
loadData()



```java
public class BigTable {

    /**
     * Loads data into database.
     * Data is in CSV files. Note that must convert to hourly data.
     * Take the first reading in a hour and ignore any others.
     */
    public void loadData() throws Exception {
        String path = "src/bin/data/";

        // TODO: Load data from CSV files into sensor table
        try {
            // SeaTac station id is SEA
            System.out.println("Load data for SeaTac");
            loadStationData(path + "seatac.csv", "SEA");

            // Vancouver station id is YVR
            System.out.println("Loading data for Vancouver");
            loadStationData(path + "vancouver.csv", "YVR");

            // Portland station id is PDX
            System.out.println("Loading data for Portland");
            loadStationData(path + "portland.csv", "PDX");

        } catch (Exception e) {
            throw new Exception(e);
        }
    }
```

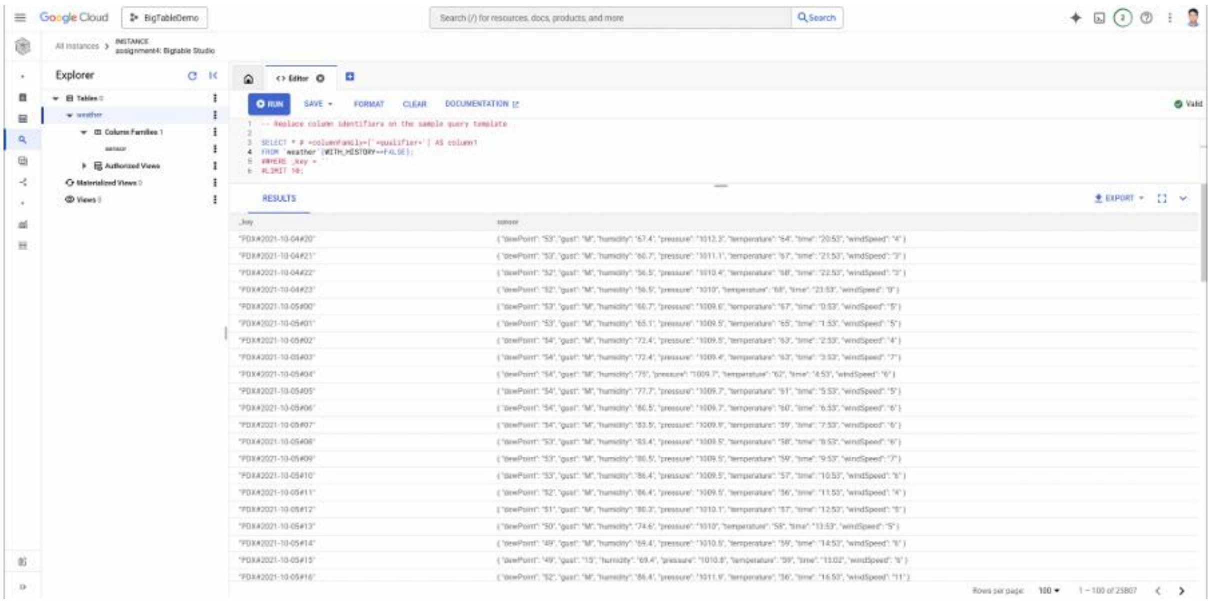


```java
    private void loadStationData(String filename, String stationId) throws Exception {
        BufferedReader reader = new BufferedReader(new FileReader(filename));
        String line;
        boolean isHeader = true;
        Map<String, Boolean> hourlyDataLoaded = new HashMap<>();
        BulkMutation bulkMutation = BulkMutation.create(TableId.of(tableId));

        while ((line = reader.readLine()) != null) {
            // Skip header rows
            if (isHeader) {
                if (line.contains("Date,Time")) {
                    isHeader = false;
                }
                continue;
            }

            String[] parts = line.split(",");
            if (parts.length < 9) continue;

            String date = parts[1].trim();
            String time = parts[2].trim();

            // Extract hour from time (HH:MM format)
            int hourInt = Integer.parseInt(time.split(":")[0]);
            String hour = String.format("%02d", hourInt);  // "00", "01", "02", etc.
            String hourKey = date + "-" + hour;

            // Skip if we already have data for this hour
            if (hourlyDataLoaded.containsKey(hourKey)) {
                continue;
            }
            hourlyDataLoaded.put(hourKey, true);

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

            // Parse data values
            String temperature = parts[3].trim();
            String dewPoint = parts[4].trim();
            String humidity = parts[5].trim();
            String windSpeed = parts[6].trim();
            String gust = parts[7].trim();
            String pressure = parts[8].trim();
```



BigTable.java

```java
public class BigTable {
    private void loadStationData(String filename, String stationId) throws Exception {  3 usages
                String humidity = parts[5].trim();
                String windSpeed = parts[6].trim();
                String gust = parts[7].trim();
                String pressure = parts[8].trim();

                // Create mutations for this row
                Mutation mutation = Mutation.create()
                        .setCell(COLUMN_FAMILY,  qualifier: "temperature", temperature)
                        .setCell(COLUMN_FAMILY,  qualifier: "dewPoint", dewPoint)
                        .setCell(COLUMN_FAMILY,  qualifier: "humidity", humidity)
                        .setCell(COLUMN_FAMILY,  qualifier: "windSpeed", windSpeed)
                        .setCell(COLUMN_FAMILY,  qualifier: "gust", gust)
                        .setCell(COLUMN_FAMILY,  qualifier: "pressure", pressure)
                        .setCell(COLUMN_FAMILY,  qualifier: "time", time);

                bulkMutation.add(rowKey, mutation);
            }

            reader.close();

            // Execute bulk mutation
            dataClient.bulkMutateRows(bulkMutation);
            System.out.println("Loaded data for station: " + stationId);
        }
```

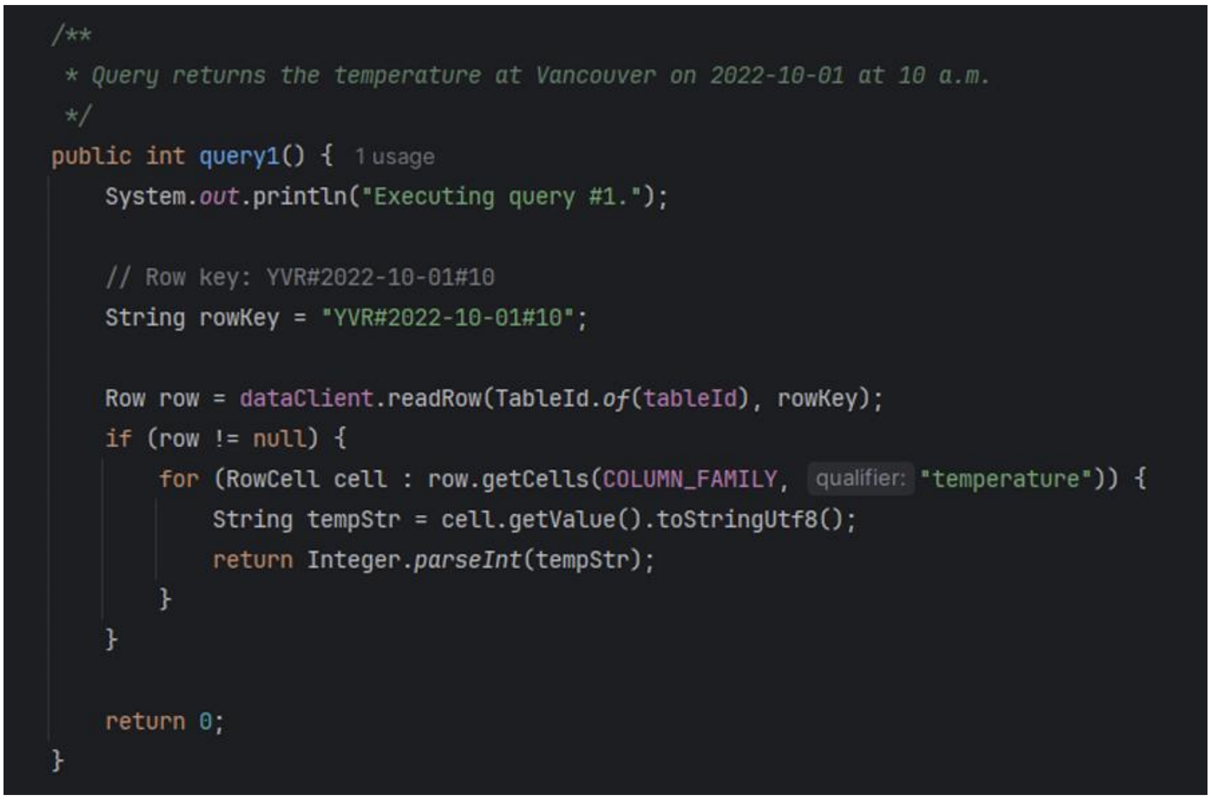


```sql
-- Replace column identifiers on the sample query template

SELECT * # <columnFamily>['<qualifier>'] AS column1
FROM `weather`(WITH_HISTORY=>FALSE);
#WHERE _key = ''
#LIMIT 10;
```

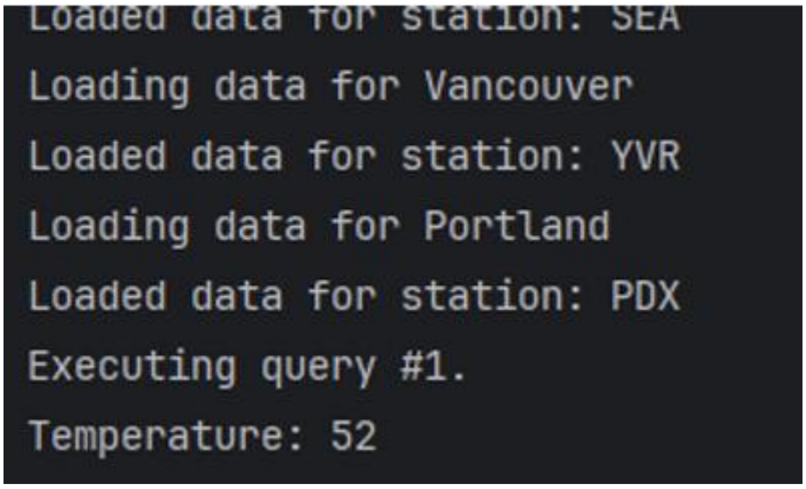
_key	sensor
"FDX#2021-10-04#20"	{ "dewPoint": "53", "gust": "M", "humidity": "67.4", "pressure": "1012.3", "temperature": "64", "time": "20:53", "windSpeed": "4" }
"FDX#2021-10-04#21"	{ "dewPoint": "53", "gust": "M", "humidity": "60.7", "pressure": "1011.1", "temperature": "67", "time": "21:53", "windSpeed": "3" }
"FDX#2021-10-04#22"	{ "dewPoint": "52", "gust": "M", "humidity": "56.5", "pressure": "1010.4", "temperature": "68", "time": "22:53", "windSpeed": "3" }
"FDX#2021-10-04#23"	{ "dewPoint": "52", "gust": "M", "humidity": "56.5", "pressure": "1010", "temperature": "68", "time": "23:53", "windSpeed": "0" }
"FDX#2021-10-05#00"	{ "dewPoint": "53", "gust": "M", "humidity": "60.7", "pressure": "1009.6", "temperature": "67", "time": "0:53", "windSpeed": "5" }
"FDX#2021-10-05#01"	{ "dewPoint": "53", "gust": "M", "humidity": "65.1", "pressure": "1009.5", "temperature": "65", "time": "1:53", "windSpeed": "5" }
"FDX#2021-10-05#02"	{ "dewPoint": "54", "gust": "M", "humidity": "72.4", "pressure": "1009.5", "temperature": "63", "time": "2:53", "windSpeed": "4" }
"FDX#2021-10-05#03"	{ "dewPoint": "54", "gust": "M", "humidity": "72.4", "pressure": "1009.4", "temperature": "63", "time": "3:53", "windSpeed": "7" }
"FDX#2021-10-05#04"	{ "dewPoint": "54", "gust": "M", "humidity": "75", "pressure": "1009.7", "temperature": "62", "time": "4:53", "windSpeed": "6" }
"FDX#2021-10-05#05"	{ "dewPoint": "54", "gust": "M", "humidity": "77.7", "pressure": "1009.7", "temperature": "61", "time": "5:53", "windSpeed": "5" }
"FDX#2021-10-05#06"	{ "dewPoint": "54", "gust": "M", "humidity": "80.5", "pressure": "1009.7", "temperature": "60", "time": "6:53", "windSpeed": "6" }
"FDX#2021-10-05#07"	{ "dewPoint": "54", "gust": "M", "humidity": "83.5", "pressure": "1009.9", "temperature": "59", "time": "7:53", "windSpeed": "6" }
"FDX#2021-10-05#08"	{ "dewPoint": "53", "gust": "M", "humidity": "83.4", "pressure": "1009.5", "temperature": "58", "time": "8:53", "windSpeed": "6" }
"FDX#2021-10-05#09"	{ "dewPoint": "53", "gust": "M", "humidity": "80.5", "pressure": "1009.5", "temperature": "59", "time": "9:53", "windSpeed": "7" }
"FDX#2021-10-05#10"	{ "dewPoint": "53", "gust": "M", "humidity": "86.4", "pressure": "1009.5", "temperature": "57", "time": "10:53", "windSpeed": "6" }
"FDX#2021-10-05#11"	{ "dewPoint": "52", "gust": "M", "humidity": "86.4", "pressure": "1009.5", "temperature": "56", "time": "11:53", "windSpeed": "4" }
"FDX#2021-10-05#12"	{ "dewPoint": "51", "gust": "M", "humidity": "80.3", "pressure": "1010.1", "temperature": "57", "time": "12:53", "windSpeed": "8" }
"FDX#2021-10-05#13"	{ "dewPoint": "50", "gust": "M", "humidity": "74.6", "pressure": "1010", "temperature": "58", "time": "13:53", "windSpeed": "5" }
"FDX#2021-10-05#14"	{ "dewPoint": "49", "gust": "M", "humidity": "69.4", "pressure": "1010.5", "temperature": "59", "time": "14:53", "windSpeed": "6" }
"FDX#2021-10-05#15"	{ "dewPoint": "49", "gust": "15", "humidity": "69.4", "pressure": "1010.8", "temperature": "59", "time": "15:02", "windSpeed": "6" }
"FDX#2021-10-05#16"	{ "dewPoint": "52", "gust": "M", "humidity": "86.4", "pressure": "1011.9", "temperature": "56", "time": "16:53", "windSpeed": "11" }

Rows per page: 100 1 – 100 of 25807

query1()

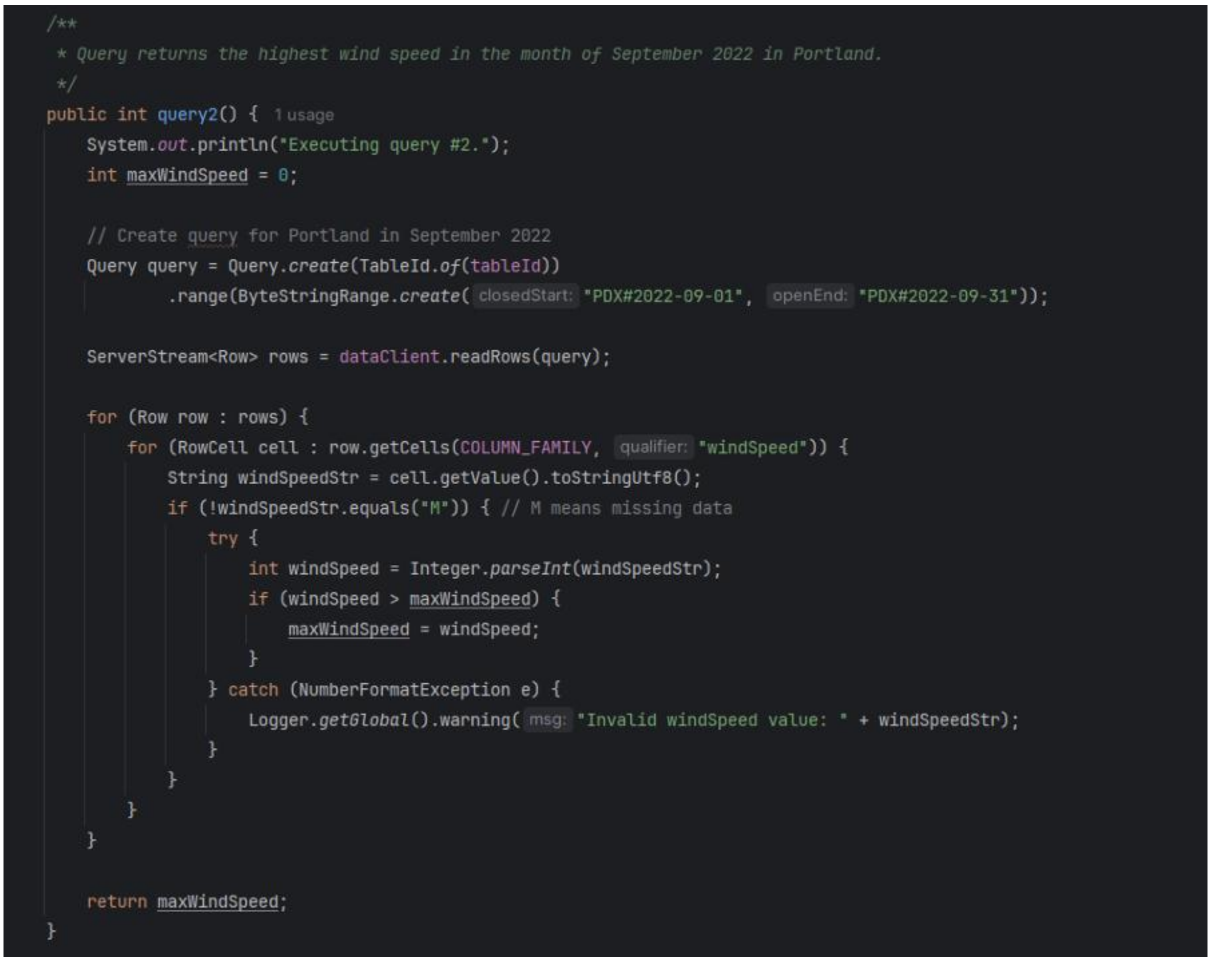


```java
/**
 * Query returns the temperature at Vancouver on 2022-10-01 at 10 a.m.
 */
public int query1() {  1 usage
    System.out.println("Executing query #1.");

    // Row key: YVR#2022-10-01#10
    String rowKey = "YVR#2022-10-01#10";

    Row row = dataClient.readRow(TableId.of(tableId), rowKey);
    if (row != null) {
        for (RowCell cell : row.getCells(COLUMN_FAMILY,  qualifier: "temperature")) {
            String tempStr = cell.getValue().toStringUtf8();
            return Integer.parseInt(tempStr);
        }
    }

    return 0;
}
```

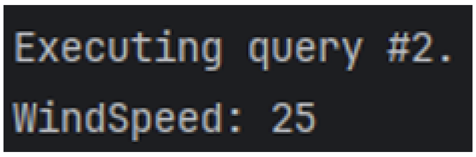


```
Loaded data for station: SEA
Loading data for Vancouver
Loaded data for station: YVR
Loading data for Portland
Loaded data for station: PDX
Executing query #1.
Temperature: 52
```

query2()



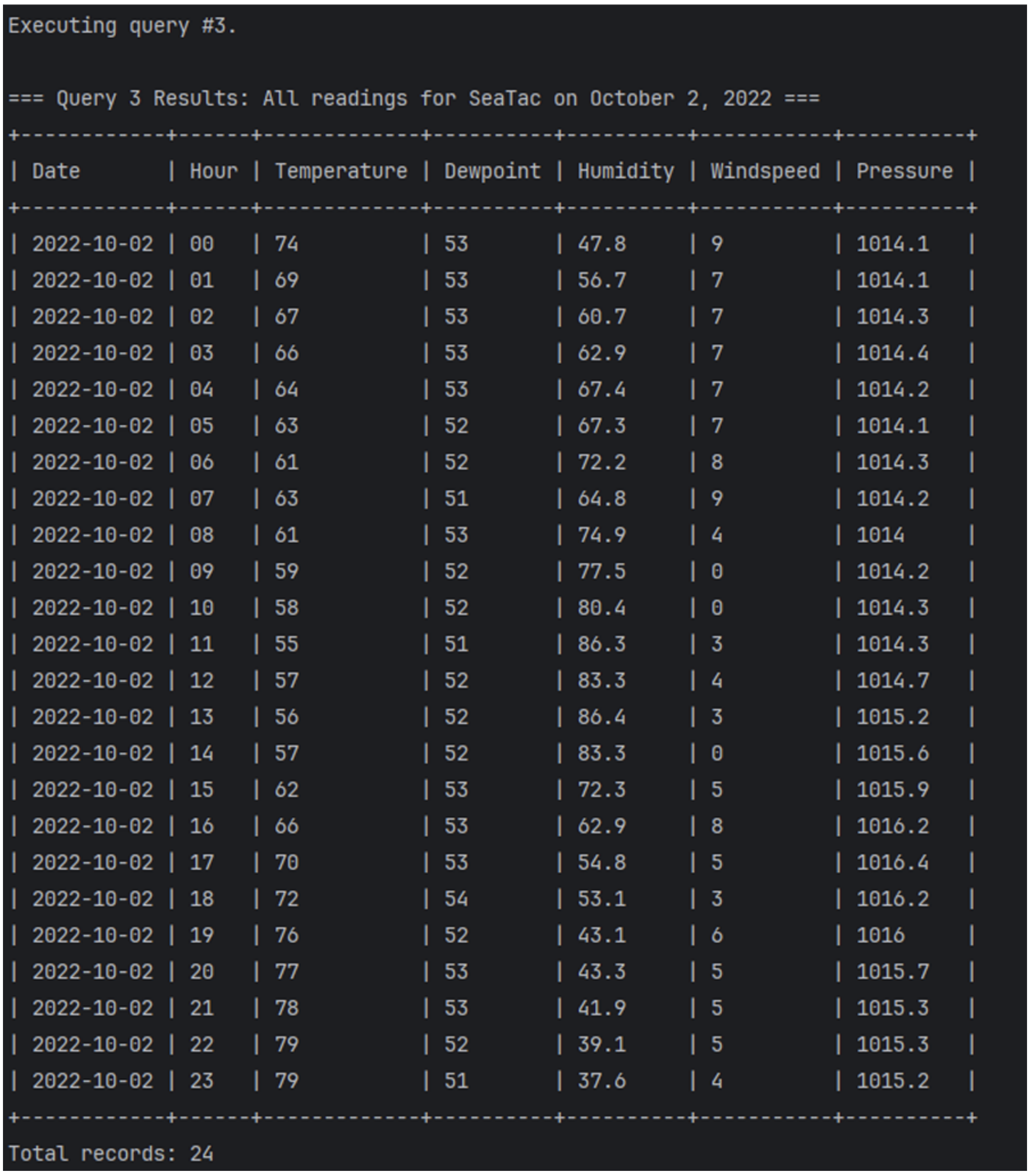
```java
/**
 * Query returns the highest wind speed in the month of September 2022 in Portland.
 */
public int query2() {  1 usage
    System.out.println("Executing query #2.");
    int maxWindSpeed = 0;

    // Create query for Portland in September 2022
    Query query = Query.create(TableId.of(tableId))
            .range(ByteStringRange.create( closedStart: "PDX#2022-09-01",  openEnd: "PDX#2022-09-31"));

    ServerStream<Row> rows = dataClient.readRows(query);

    for (Row row : rows) {
        for (RowCell cell : row.getCells(COLUMN_FAMILY,  qualifier: "windSpeed")) {
            String windSpeedStr = cell.getValue().toStringUtf8();
            if (!windSpeedStr.equals("M")) { // M means missing data
                try {
                    int windSpeed = Integer.parseInt(windSpeedStr);
                    if (windSpeed > maxWindSpeed) {
                        maxWindSpeed = windSpeed;
                    }
                } catch (NumberFormatException e) {
                    Logger.getGlobal().warning( msg: "Invalid windSpeed value: " + windSpeedStr);
                }
            }
        }
    }

    return maxWindSpeed;
}
```



```
Executing query #2.
WindSpeed: 25
```

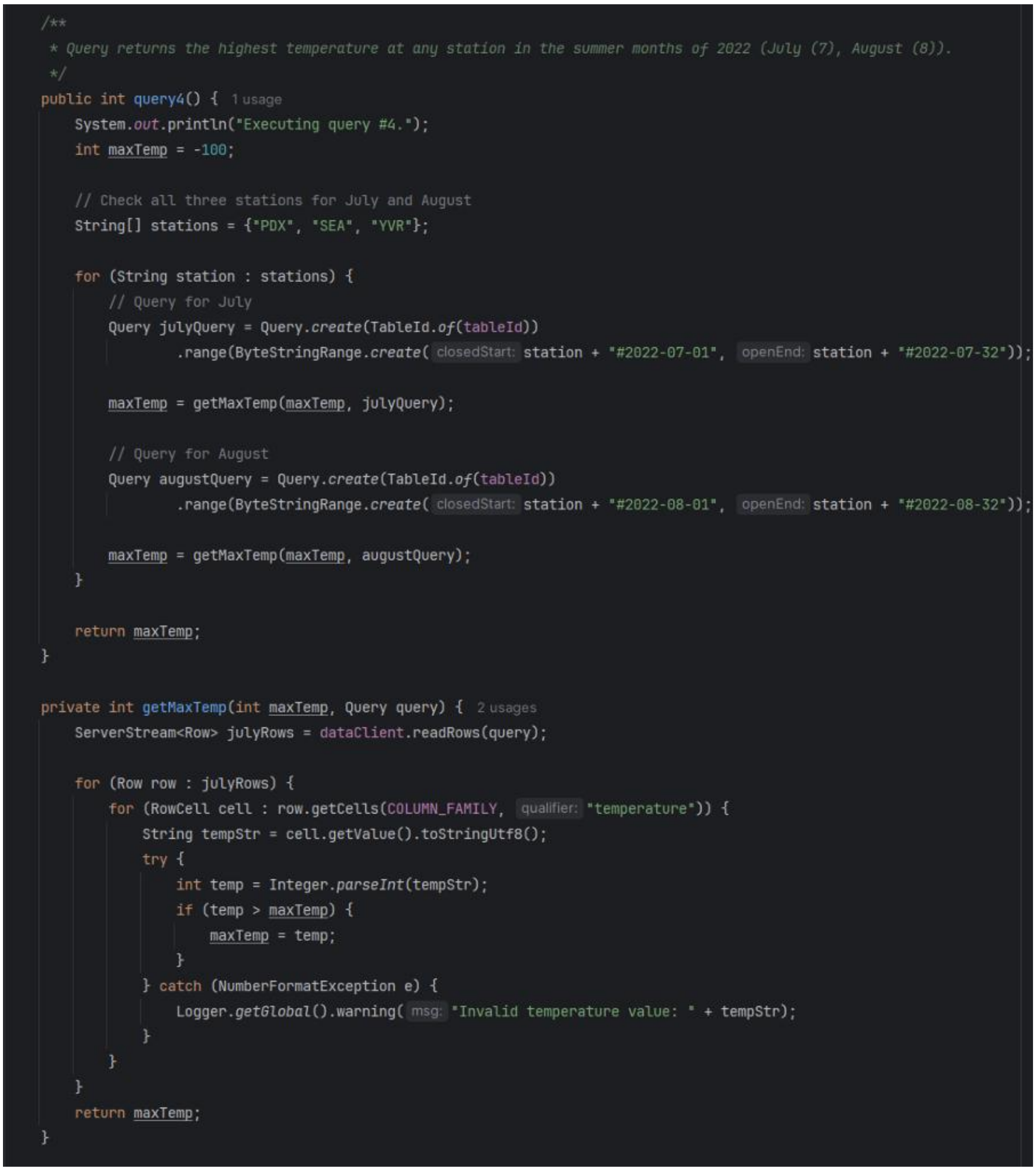
query3()

```java
/**
     * Query returns all the readings for SeaTac for October 2, 2022.
     */
    public ArrayList<Object[]> query3() {
        System.out.println("Executing query #3.");
        ArrayList<Object[]> data = new ArrayList<>();

        // Create query for SeaTac on October 2, 2022

        Query query = Query.create(TableId.of(tableId))
                .range(ByteStringRange.create("SEA#2022-10-02#00",
"SEA#2022-10-02#24"));

        ServerStream<Row> rows = dataClient.readRows(query);

        for (Row row : rows) {
            // Extract date and hour from row key
            String rowKey = row.getKey().toStringUtf8();
            String[] keyParts = rowKey.split("#");
            String date = keyParts[1];
            String hour = keyParts[2];

            // Get all sensor values
            String temperature = "";
            String dewpoint = "";
            String humidity = "";
            String windSpeed = "";
            String pressure = "";

            for (RowCell cell : row.getCells()) {
                String qualifier = cell.getQualifier().toStringUtf8();
                String value = cell.getValue().toStringUtf8();

                switch (qualifier) {
                    case "temperature":
                        temperature = value;
                        break;
                    case "dewPoint":
                        dewpoint = value;
                        break;
                    case "humidity":
                        humidity = value;
                        break;
                    case "windSpeed":
                        windSpeed = value;
                        break;
                    case "pressure":
                        pressure = value;
                        break;
                }
            }
```

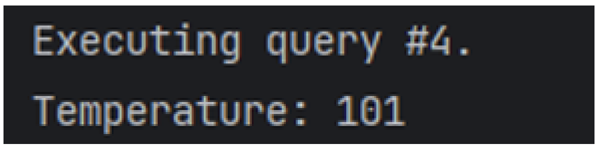


```
Executing query #3.

=== Query 3 Results: All readings for SeaTac on October 2, 2022 ===
+------------+------+-------------+----------+----------+-----------+----------+
| Date       | Hour | Temperature | Dewpoint | Humidity | Windspeed | Pressure |
+------------+------+-------------+----------+----------+-----------+----------+
| 2022-10-02 | 00   | 74          | 53       | 47.8     | 9         | 1014.1   |
| 2022-10-02 | 01   | 69          | 53       | 56.7     | 7         | 1014.1   |
| 2022-10-02 | 02   | 67          | 53       | 60.7     | 7         | 1014.3   |
| 2022-10-02 | 03   | 66          | 53       | 62.9     | 7         | 1014.4   |
| 2022-10-02 | 04   | 64          | 53       | 67.4     | 7         | 1014.2   |
| 2022-10-02 | 05   | 63          | 52       | 67.3     | 7         | 1014.1   |
| 2022-10-02 | 06   | 61          | 52       | 72.2     | 8         | 1014.3   |
| 2022-10-02 | 07   | 63          | 51       | 64.8     | 9         | 1014.2   |
| 2022-10-02 | 08   | 61          | 53       | 74.9     | 4         | 1014     |
| 2022-10-02 | 09   | 59          | 52       | 77.5     | 0         | 1014.2   |
| 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 | 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   |
+------------+------+-------------+----------+----------+-----------+----------+
Total records: 24
```

query4()

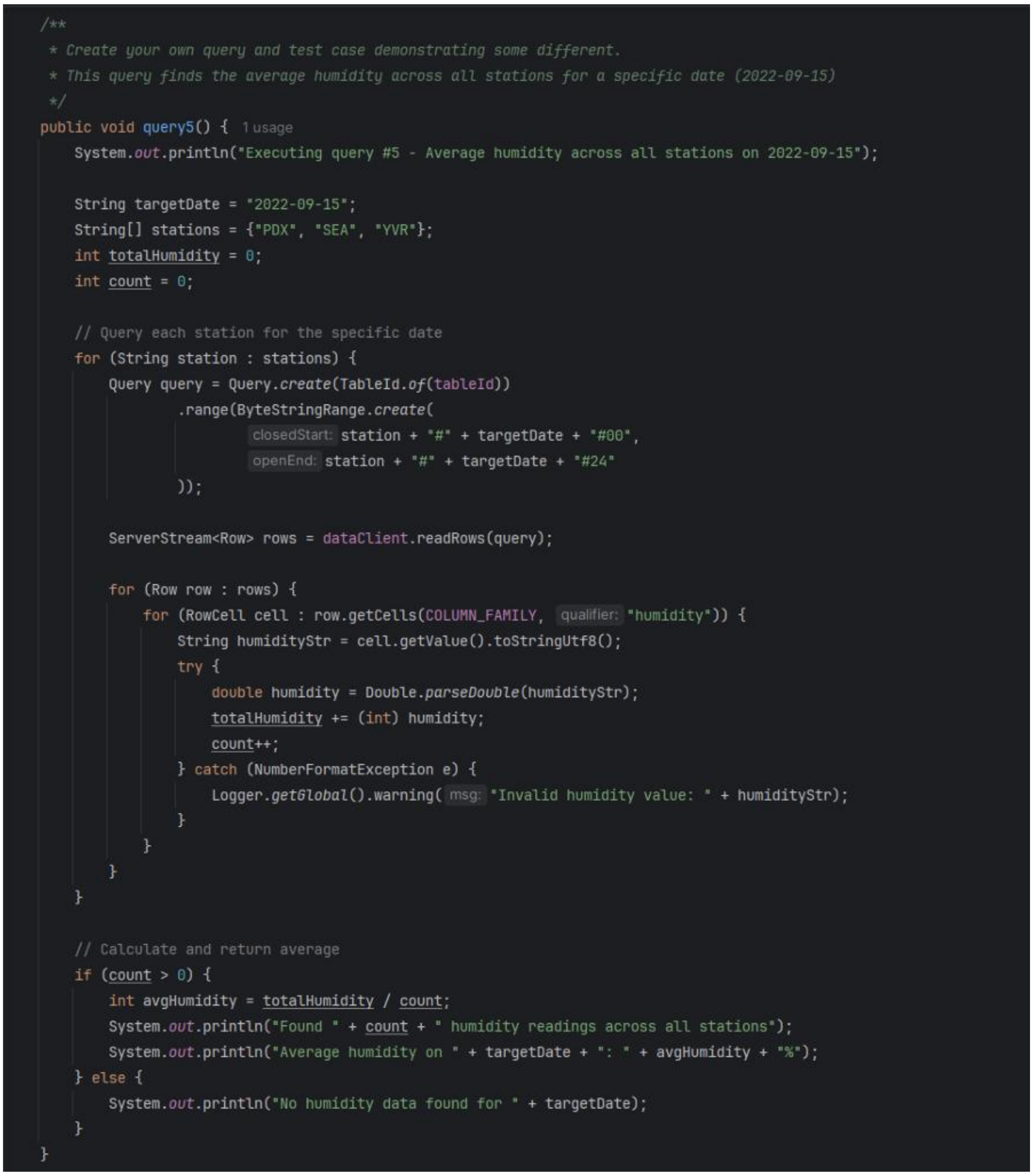


```
/**
 * Query returns the highest temperature at any station in the summer months of 2022 (July (7), August (8)).
 */
public int query4() {  1 usage
    System.out.println("Executing query #4.");
    int maxTemp = -100;

    // Check all three stations for July and August
    String[] stations = {"PDX", "SEA", "YVR"};

    for (String station : stations) {
        // Query for July
        Query julyQuery = Query.create(TableId.of(tableId))
                .range(ByteStringRange.create( closedStart: station + "#2022-07-01",  openEnd: station + "#2022-07-32"));

        maxTemp = getMaxTemp(maxTemp, julyQuery);

        // Query for August
        Query augustQuery = Query.create(TableId.of(tableId))
                .range(ByteStringRange.create( closedStart: station + "#2022-08-01",  openEnd: station + "#2022-08-32"));

        maxTemp = getMaxTemp(maxTemp, augustQuery);
    }

    return maxTemp;
}

private int getMaxTemp(int maxTemp, Query query) {  2 usages
    ServerStream<Row> julyRows = dataClient.readRows(query);

    for (Row row : julyRows) {
        for (RowCell cell : row.getCells(COLUMN_FAMILY,  qualifier: "temperature")) {
            String tempStr = cell.getValue().toStringUtf8();
            try {
                int temp = Integer.parseInt(tempStr);
                if (temp > maxTemp) {
                    maxTemp = temp;
                }
            } catch (NumberFormatException e) {
                Logger.getGlobal().warning( msg: "Invalid temperature value: " + tempStr);
            }
        }
    }
    return maxTemp;
}
```

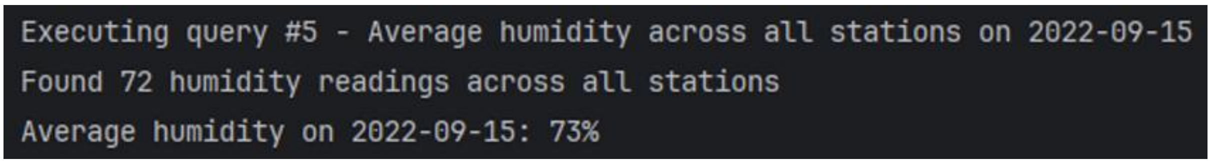


```
Executing query #4.
Temperature: 101
```

query5()



```
/**
 * Create your own query and test case demonstrating some different.
 * This query finds the average humidity across all stations for a specific date (2022-09-15)
 */
public void query5() {  1 usage
    System.out.println("Executing query #5 - Average humidity across all stations on 2022-09-15");

    String targetDate = "2022-09-15";
    String[] stations = {"PDX", "SEA", "YVR"};
    int totalHumidity = 0;
    int count = 0;

    // Query each station for the specific date
    for (String station : stations) {
        Query query = Query.create(TableId.of(tableId))
                .range(ByteStringRange.create(
                        closedStart: station + "#" + targetDate + "#00",
                        openEnd: station + "#" + targetDate + "#24"
                ));

        ServerStream<Row> rows = dataClient.readRows(query);

        for (Row row : rows) {
            for (RowCell cell : row.getCells(COLUMN_FAMILY,  qualifier: "humidity")) {
                String humidityStr = cell.getValue().toStringUtf8();
                try {
                    double humidity = Double.parseDouble(humidityStr);
                    totalHumidity += (int) humidity;
                    count++;
                } catch (NumberFormatException e) {
                    Logger.getGlobal().warning( msg: "Invalid humidity value: " + humidityStr);
                }
            }
        }
    }

    // Calculate and return average
    if (count > 0) {
        int avgHumidity = totalHumidity / count;
        System.out.println("Found " + count + " humidity readings across all stations");
        System.out.println("Average humidity on " + targetDate + ": " + avgHumidity + "%");
    } else {
        System.out.println("No humidity data found for " + targetDate);
    }
}
```



```
Executing query #5 - Average humidity across all stations on 2022-09-15
Found 72 humidity readings across all stations
Average humidity on 2022-09-15: 73%
```