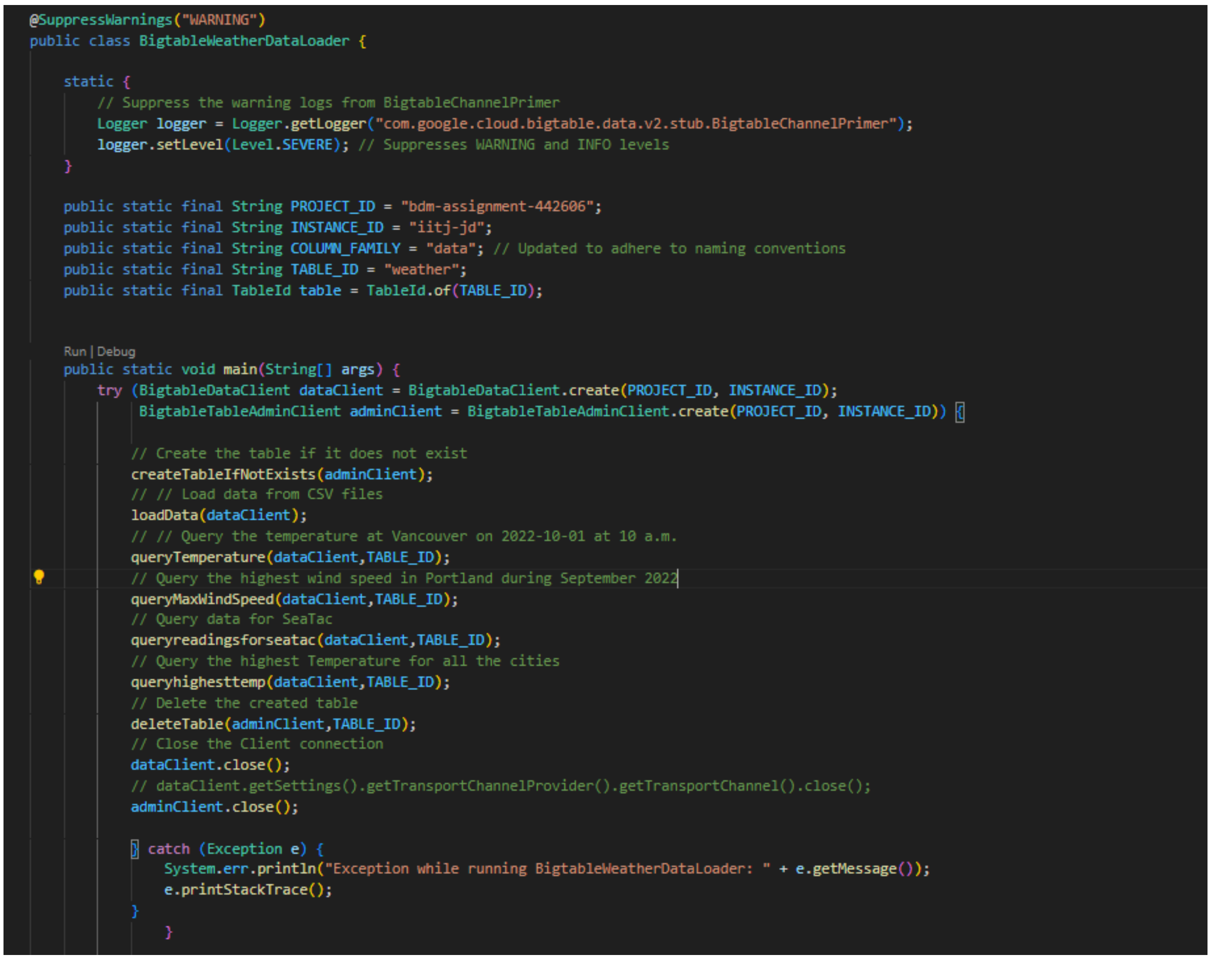
Big data Management

Assignment 4

Jeyadev L

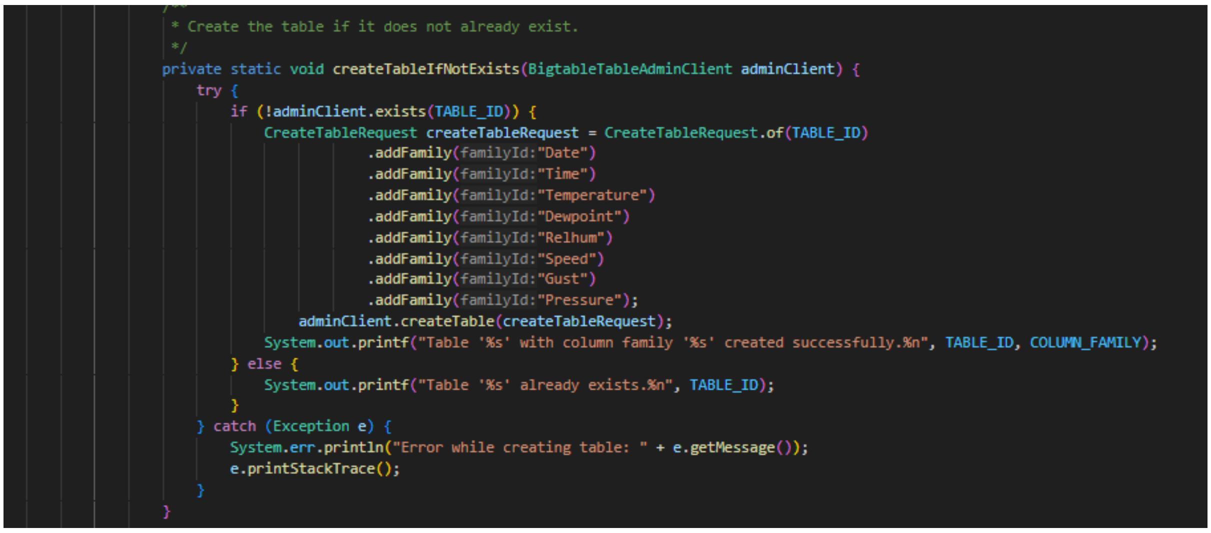
G23AI2071

Main Function



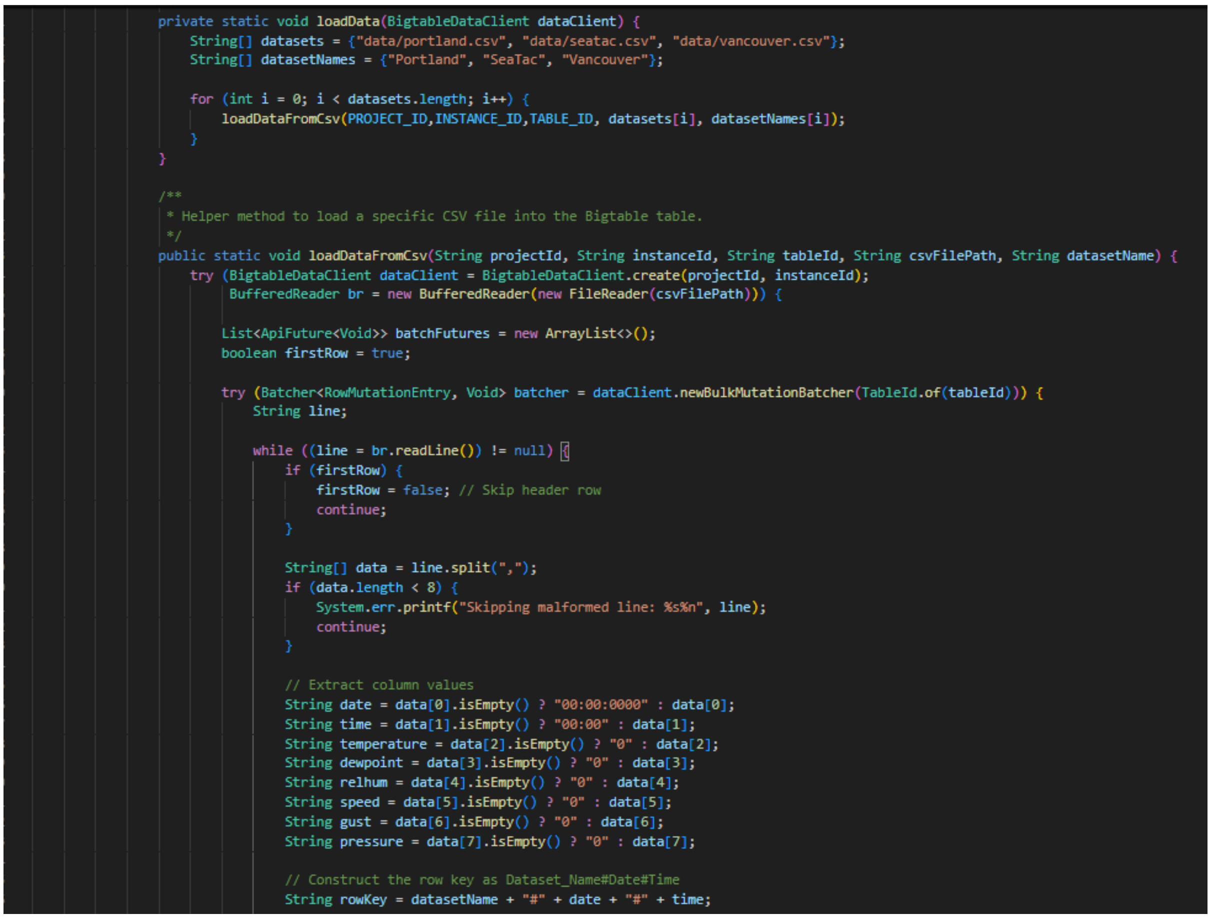
```java
@SuppressWarnings("WARNING")
public class BigtableWeatherDataLoader {

    static {
        // Suppress the warning logs from BigtableChannelPrimer
        Logger logger = Logger.getLogger("com.google.cloud.bigtable.data.v2.stub.BigtableChannelPrimer");
        logger.setLevel(Level.SEVERE); // Suppresses WARNING and INFO levels
    }

    public static final String PROJECT_ID = "bdm-assignment-442606";
    public static final String INSTANCE_ID = "iitj-jd";
    public static final String COLUMN_FAMILY = "data"; // Updated to adhere to naming conventions
    public static final String TABLE_ID = "weather";
    public static final TableId table = TableId.of(TABLE_ID);


    Run | Debug
    public static void main(String[] args) {
        try (BigtableDataClient dataClient = BigtableDataClient.create(PROJECT_ID, INSTANCE_ID);
             BigtableTableAdminClient adminClient = BigtableTableAdminClient.create(PROJECT_ID, INSTANCE_ID)) {

            // Create the table if it does not exist
            createTableIfNotExists(adminClient);
            // // Load data from CSV files
            loadData(dataClient);
            // // Query the temperature at Vancouver on 2022-10-01 at 10 a.m.
            queryTemperature(dataClient,TABLE_ID);
            // Query the highest wind speed in Portland during September 2022
            queryMaxWindSpeed(dataClient,TABLE_ID);
            // Query data for SeaTac
            queryreadingsforseatac(dataClient,TABLE_ID);
            // Query the highest Temperature for all the cities
            queryhighesttemp(dataClient,TABLE_ID);
            // Delete the created table
            deleteTable(adminClient,TABLE_ID);
            // Close the Client connection
            dataClient.close();
            // dataClient.getSettings().getTransportChannelProvider().getTransportChannel().close();
            adminClient.close();

            } catch (Exception e) {
                System.err.println("Exception while running BigtableWeatherDataLoader: " + e.getMessage());
                e.printStackTrace();
            }
                }
```

Create Table Function

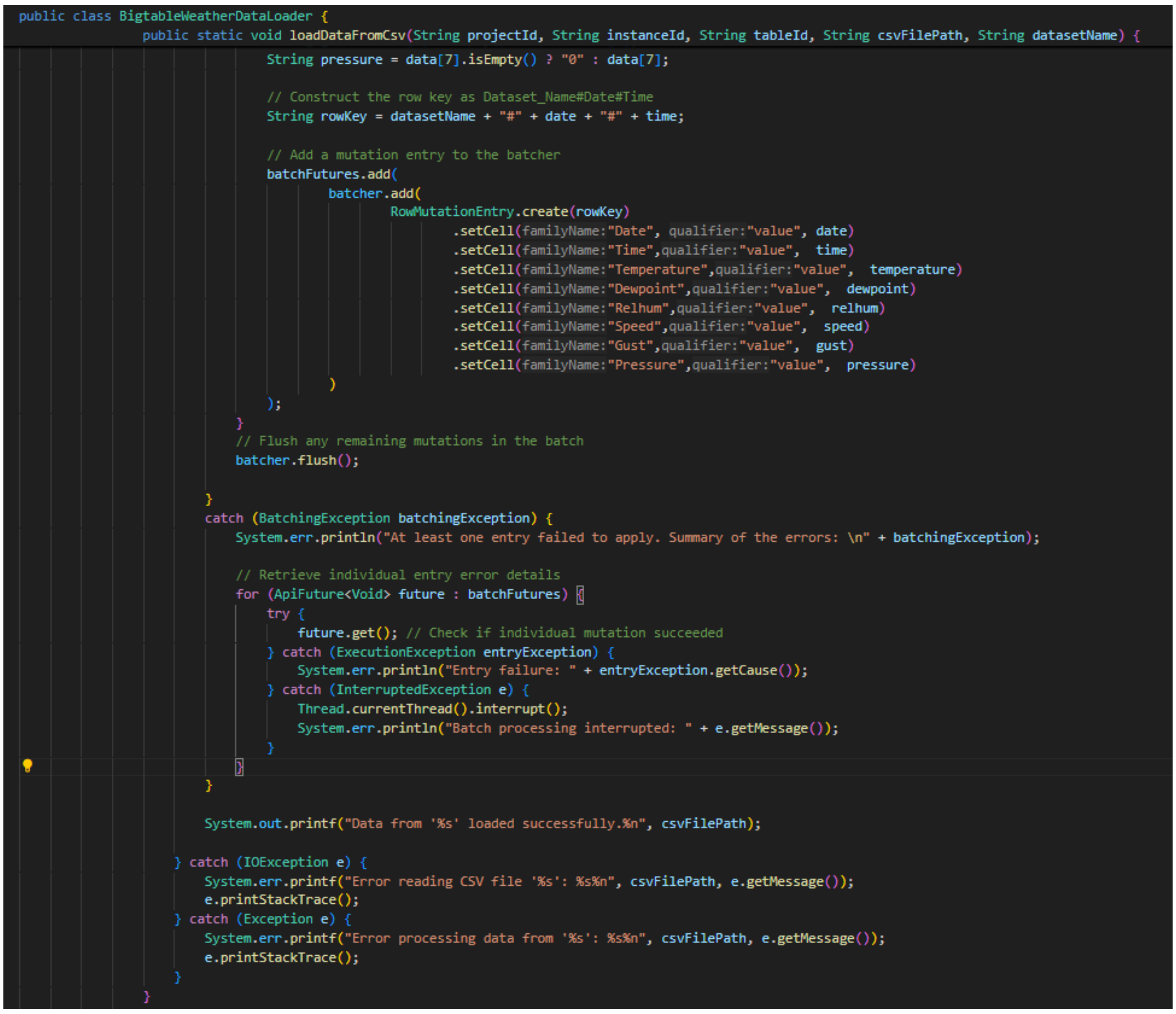


```java
/**
 * Create the table if it does not already exist.
 */
private static void createTableIfNotExists(BigtableTableAdminClient adminClient) {
    try {
        if (!adminClient.exists(TABLE_ID)) {
            CreateTableRequest createTableRequest = CreateTableRequest.of(TABLE_ID)
                        .addFamily(familyId:"Date")
                        .addFamily(familyId:"Time")
                        .addFamily(familyId:"Temperature")
                        .addFamily(familyId:"Dewpoint")
                        .addFamily(familyId:"Relhum")
                        .addFamily(familyId:"Speed")
                        .addFamily(familyId:"Gust")
                        .addFamily(familyId:"Pressure");
                adminClient.createTable(createTableRequest);
            System.out.printf("Table '%s' with column family '%s' created successfully.%n", TABLE_ID, COLUMN_FAMILY);
        } else {
            System.out.printf("Table '%s' already exists.%n", TABLE_ID);
        }
    } catch (Exception e) {
        System.err.println("Error while creating table: " + e.getMessage());
        e.printStackTrace();
    }
}
```

Load Data Function

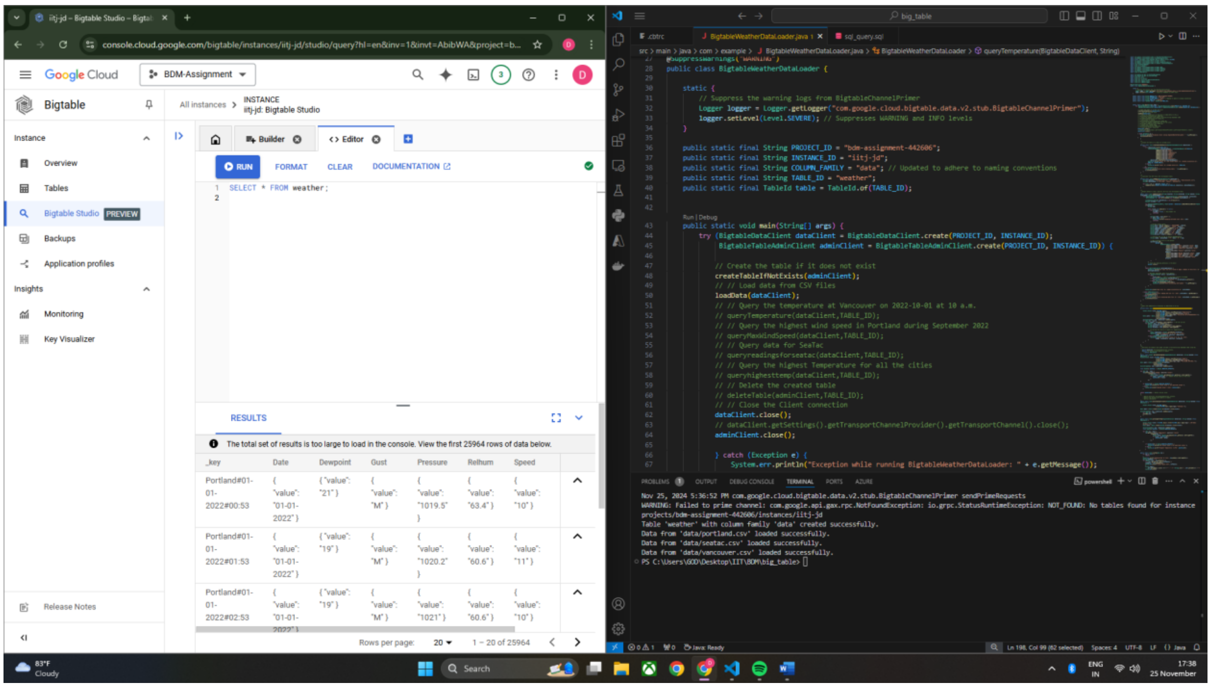


```java
private static void loadData(BigtableDataClient dataClient) {
    String[] datasets = {"data/portland.csv", "data/seatac.csv", "data/vancouver.csv"};
    String[] datasetNames = {"Portland", "SeaTac", "Vancouver"};

    for (int i = 0; i < datasets.length; i++) {
        loadDataFromCsv(PROJECT_ID,INSTANCE_ID,TABLE_ID, datasets[i], datasetNames[i]);
    }
}

/**
 * Helper method to load a specific CSV file into the Bigtable table.
 */
public static void loadDataFromCsv(String projectId, String instanceId, String tableId, String csvFilePath, String datasetName) {
    try (BigtableDataClient dataClient = BigtableDataClient.create(projectId, instanceId);
         BufferedReader br = new BufferedReader(new FileReader(csvFilePath))) {

        List<ApiFuture<Void>> batchFutures = new ArrayList<>();
        boolean firstRow = true;

        try (Batcher<RowMutationEntry, Void> batcher = dataClient.newBulkMutationBatcher(TableId.of(tableId))) {
            String line;

            while ((line = br.readLine()) != null) {
                if (firstRow) {
                    firstRow = false; // Skip header row
                    continue;
                }

                String[] data = line.split(",");
                if (data.length < 8) {
                    System.err.printf("Skipping malformed line: %s%n", line);
                    continue;
                }

                // Extract column values
                String date = data[0].isEmpty() ? "00:00:0000" : data[0];
                String time = data[1].isEmpty() ? "00:00" : data[1];
                String temperature = data[2].isEmpty() ? "0" : data[2];
                String dewpoint = data[3].isEmpty() ? "0" : data[3];
                String relhum = data[4].isEmpty() ? "0" : data[4];
                String speed = data[5].isEmpty() ? "0" : data[5];
                String gust = data[6].isEmpty() ? "0" : data[6];
                String pressure = data[7].isEmpty() ? "0" : data[7];

                // Construct the row key as Dataset_Name#Date#Time
                String rowKey = datasetName + "#" + date + "#" + time;
```



```java
public class BigtableWeatherDataLoader {
                public static void loadDataFromCsv(String projectId, String instanceId, String tableId, String csvFilePath, String datasetName) {
                                String pressure = data[7].isEmpty() ? "0" : data[7];

                                // Construct the row key as Dataset_Name#Date#Time
                                String rowKey = datasetName + "#" + date + "#" + time;

                                // Add a mutation entry to the batcher
                                batchFutures.add(
                                        batcher.add(
                                                RowMutationEntry.create(rowKey)
                                                        .setCell(familyName:"Date", qualifier:"value", date)
                                                        .setCell(familyName:"Time",qualifier:"value",  time)
                                                        .setCell(familyName:"Temperature",qualifier:"value",  temperature)
                                                        .setCell(familyName:"Dewpoint",qualifier:"value",  dewpoint)
                                                        .setCell(familyName:"Relhum",qualifier:"value",  relhum)
                                                        .setCell(familyName:"Speed",qualifier:"value",  speed)
                                                        .setCell(familyName:"Gust",qualifier:"value",  gust)
                                                        .setCell(familyName:"Pressure",qualifier:"value",  pressure)
                                        )
                                );
                            }
                            // Flush any remaining mutations in the batch
                            batcher.flush();

                        }
                        catch (BatchingException batchingException) {
                            System.err.println("At least one entry failed to apply. Summary of the errors: \n" + batchingException);

                            // Retrieve individual entry error details
                            for (ApiFuture<Void> future : batchFutures) {
                                try {
                                    future.get(); // Check if individual mutation succeeded
                                } catch (ExecutionException entryException) {
                                    System.err.println("Entry failure: " + entryException.getCause());
                                } catch (InterruptedException e) {
                                    Thread.currentThread().interrupt();
                                    System.err.println("Batch processing interrupted: " + e.getMessage());
                                }
                            }
                        }

                        System.out.printf("Data from '%s' loaded successfully.%n", csvFilePath);

                    } catch (IOException e) {
                        System.err.printf("Error reading CSV file '%s': %s%n", csvFilePath, e.getMessage());
                        e.printStackTrace();
                    } catch (Exception e) {
                        System.err.printf("Error processing data from '%s': %s%n", csvFilePath, e.getMessage());
                        e.printStackTrace();
                    }
                }
```

Table Creation and Data load



```sql
SELECT * FROM weather;
```

The total set of results is too large to load in the console. View the first 25964 rows of data below.

_key	Date	Dewpoint	Gust	Pressure	Relhum	Speed
Portland#01-01-2022#00:53	{ "value": "01-01-2022" }	{ "value": "21" }	{ "value": "M" }	{ "value": "1019.5" }	{ "value": "63.4" }	{ "value": "10" }
Portland#01-01-2022#01:53	{ "value": "01-01-2022" }	{ "value": "19" }	{ "value": "M" }	{ "value": "1020.2" }	{ "value": "60.6" }	{ "value": "11" }
Portland#01-01-2022#02:53	{ "value": "01-01-2022" }	{ "value": "19" }	{ "value": "M" }	{ "value": "1021" }	{ "value": "60.6" }	{ "value": "10" }

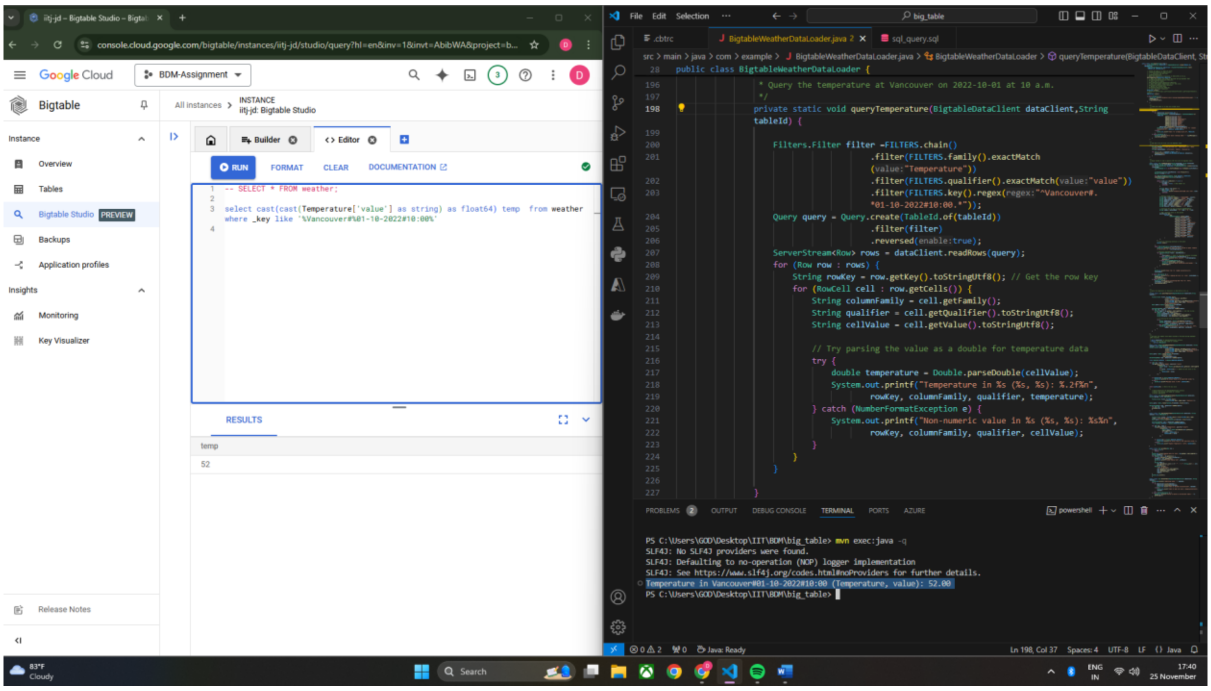
Rows per page: 20   1 – 20 of 25964

```java
@SuppressWarnings("WARNING")
public class BigtableWeatherDataLoader {

    static {
        // Suppress the warning logs from BigtableChannelPrimer
        Logger logger = Logger.getLogger("com.google.cloud.bigtable.data.v2.stub.BigtableChannelPrimer");
        logger.setLevel(Level.SEVERE); // Suppresses WARNING and INFO levels
    }

    public static final String PROJECT_ID = "bdm-assignment-442606";
    public static final String INSTANCE_ID = "iitj-jd";
    public static final String COLUMN_FAMILY = "data"; // Updated to adhere to naming conventions
    public static final String TABLE_ID = "weather";
    public static final TableId table = TableId.of(TABLE_ID);


    public static void main(String[] args) {
        try (BigtableDataClient dataClient = BigtableDataClient.create(PROJECT_ID, INSTANCE_ID);
             BigtableTableAdminClient adminClient = BigtableTableAdminClient.create(PROJECT_ID, INSTANCE_ID)) {

            // Create the table if it does not exist
            createTableIfNotExists(adminClient);
            // // Load data from CSV files
            loadData(dataClient);
            // // Query the temperature at Vancouver on 2022-10-01 at 10 a.m.
            // queryTemperature(dataClient,TABLE_ID);
            // // Query the highest wind speed in Portland during September 2022
            // queryMaxWindSpeed(dataClient,TABLE_ID);
            // // Query data for SeaTac
            // queryreadingsforseatac(dataClient,TABLE_ID);
            // // Query the highest Temperature for all the cities
            // queryhighesttemp(dataClient,TABLE_ID);
            // // Delete the created table
            // deleteTable(adminClient,TABLE_ID);
            // // Close the Client connection
            dataClient.close();
            // dataClient.getSettings().getTransportChannelProvider().getTransportChannel().close();
            adminClient.close();

        } catch (Exception e) {
            System.err.println("Exception while running BigtableWeatherDataLoader: " + e.getMessage());
```

```
Nov 25, 2024 5:36:52 PM com.google.cloud.bigtable.data.v2.stub.BigtableChannelPrimer sendPrimeRequests
WARNING: Failed to prime channel: com.google.api.gax.rpc.NotFoundException: io.grpc.StatusRuntimeException: NOT_FOUND: No tables found for instance
projects/bdm-assignment-442606/instances/iitj-jd
Table 'weather' with column family 'data' created successfully.
Data from 'data/portland.csv' loaded successfully.
Data from 'data/seatac.csv' loaded successfully.
Data from 'data/vancouver.csv' loaded successfully.
PS C:\Users\GOD\Desktop\IIT\BDM\big_table>
```

Query 1-

Returns the temperature at Vancouver on 2022-10-01 at 10 a.m.



```sql
-- SELECT * FROM weather;

select cast(cast(Temperature['value'] as string) as float64) temp  from weather
where _key like '%Vancouver#%01-10-2022#10:00%'
```

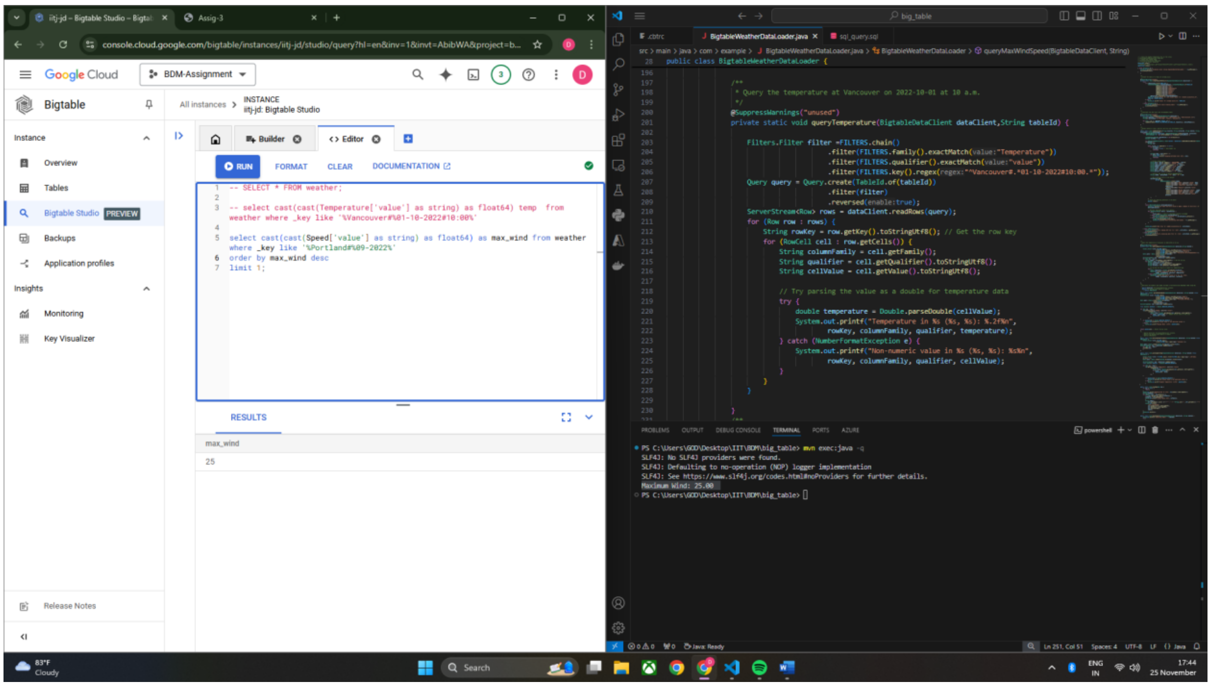
temp
52

```java
public class BigtableWeatherDataLoader {
        * Query the temperature at Vancouver on 2022-10-01 at 10 a.m.
        */
        private static void queryTemperature(BigtableDataClient dataClient,String
        tableId) {

            Filters.Filter filter =FILTERS.chain()
                                .filter(FILTERS.family().exactMatch
                                (value:"Temperature"))
                                .filter(FILTERS.qualifier().exactMatch(value:"value"))
                                .filter(FILTERS.key().regex(regex:"^Vancouver#.
                                *01-10-2022#10:00.*"));
            Query query = Query.create(TableId.of(tableId))
                                .filter(filter)
                                .reversed(enable:true);
            ServerStream<Row> rows = dataClient.readRows(query);
            for (Row row : rows) {
                String rowKey = row.getKey().toStringUtf8(); // Get the row key
                for (RowCell cell : row.getCells()) {
                    String columnFamily = cell.getFamily();
                    String qualifier = cell.getQualifier().toStringUtf8();
                    String cellValue = cell.getValue().toStringUtf8();

                    // Try parsing the value as a double for temperature data
                    try {
                        double temperature = Double.parseDouble(cellValue);
                        System.out.printf("Temperature in %s (%s, %s): %.2f%n",
                                rowKey, columnFamily, qualifier, temperature);
                    } catch (NumberFormatException e) {
                        System.out.printf("Non-numeric value in %s (%s, %s): %s%n",
                                rowKey, columnFamily, qualifier, cellValue);
                    }
                }
            }

        }
```

```
PS C:\Users\GOD\Desktop\IIT\BDM\big_table> mvn exec:java -q
SLF4J: No SLF4J providers were found.
SLF4J: Defaulting to no-operation (NOP) logger implementation
SLF4J: See https://www.slf4j.org/codes.html#noProviders for further details.
Temperature in Vancouver#01-10-2022#10:00 (Temperature, value): 52.00
PS C:\Users\GOD\Desktop\IIT\BDM\big_table>
```

Query 2

Returns the highest wind speed in the month of September 2022 in Portland.



```sql
-- SELECT * FROM weather;

-- select cast(cast(Temperature['value'] as string) as float64) temp  from
weather where _key like '%Vancouver#%01-10-2022#10:00%'

select cast(cast(Speed['value'] as string) as float64) as max_wind from weather
where _key like '%Portland#%09-2022%'
order by max_wind desc
limit 1;
```

max_wind
25

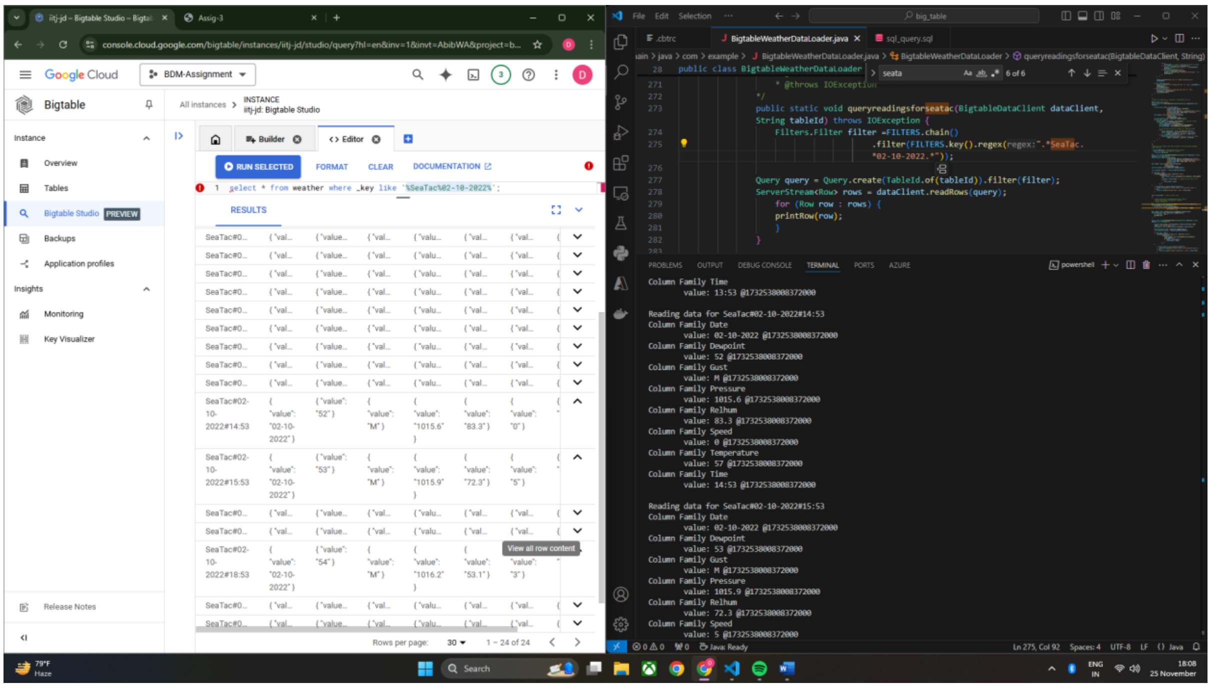
```java
public class BigtableWeatherDataLoader {

        /**
         * Query the temperature at Vancouver on 2022-10-01 at 10 a.m.
         */
        @SuppressWarnings("unused")
        private static void queryTemperature(BigtableDataClient dataClient,String tableId) {

            Filters.Filter filter =FILTERS.chain()
                                .filter(FILTERS.family().exactMatch(value:"Temperature"))
                                .filter(FILTERS.qualifier().exactMatch(value:"value"))
                                .filter(FILTERS.key().regex(regex:"^Vancouver#.*01-10-2022#10:00.*"));
            Query query = Query.create(TableId.of(tableId))
                                .filter(filter)
                                .reversed(enable:true);
            ServerStream<Row> rows = dataClient.readRows(query);
            for (Row row : rows) {
                String rowKey = row.getKey().toStringUtf8(); // Get the row key
                for (RowCell cell : row.getCells()) {
                    String columnFamily = cell.getFamily();
                    String qualifier = cell.getQualifier().toStringUtf8();
                    String cellValue = cell.getValue().toStringUtf8();

                    // Try parsing the value as a double for temperature data
                    try {
                        double temperature = Double.parseDouble(cellValue);
                        System.out.printf("Temperature in %s (%s, %s): %.2f%n",
                                rowKey, columnFamily, qualifier, temperature);
                    } catch (NumberFormatException e) {
                        System.out.printf("Non-numeric value in %s (%s, %s): %s%n",
                                rowKey, columnFamily, qualifier, cellValue);
                    }
                }
            }

        }
```

```
PS C:\Users\GOD\Desktop\IIT\BDM\big_table> mvn exec:java -q
SLF4J: No SLF4J providers were found.
SLF4J: Defaulting to no-operation (NOP) logger implementation
SLF4J: See https://www.slf4j.org/codes.html#noProviders for further details.
Maximum Wind: 25.00
PS C:\Users\GOD\Desktop\IIT\BDM\big_table>
```

Query 3

Returns all the readings for SeaTac for October 2, 2022.

Note :- Output to large to show case in single screenshot



```sql
select * from weather where _key like '%SeaTac%02-10-2022%';
```

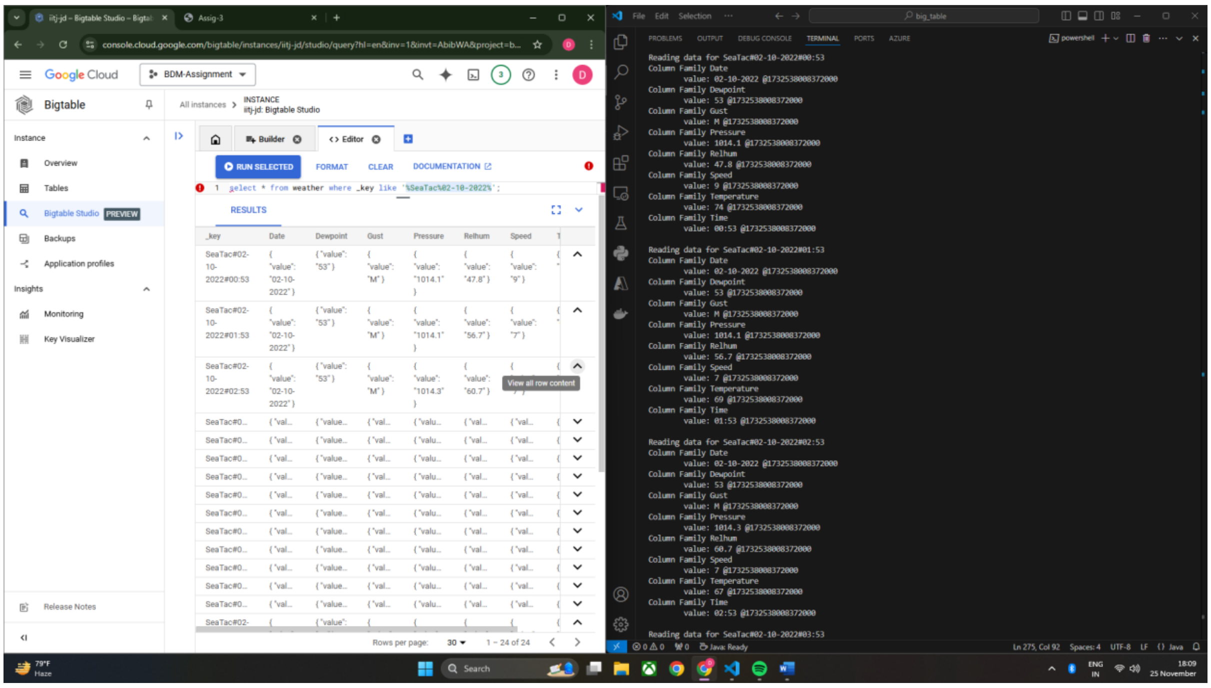
RESULTS

SeaTac#0...	{ "val...	{ "value...	{ "val...	{ "valu...	{ "val...	{ "val...	{
SeaTac#0...	{ "val...	{ "value...	{ "val...	{ "valu...	{ "val...	{ "val...	{
SeaTac#0...	{ "val...	{ "value...	{ "val...	{ "valu...	{ "val...	{ "val...	{
SeaTac#0...	{ "val...	{ "value...	{ "val...	{ "valu...	{ "val...	{ "val...	{
SeaTac#0...	{ "val...	{ "value...	{ "val...	{ "valu...	{ "val...	{ "val...	{
SeaTac#0...	{ "val...	{ "value...	{ "val...	{ "valu...	{ "val...	{ "val...	{
SeaTac#0...	{ "val...	{ "value...	{ "val...	{ "valu...	{ "val...	{ "val...	{
SeaTac#0...	{ "val...	{ "value...	{ "val...	{ "valu...	{ "val...	{ "val...	{
SeaTac#0...	{ "val...	{ "value...	{ "val...	{ "valu...	{ "val...	{ "val...	{
SeaTac#02-10-2022#14:53	{ "value": "02-10-2022" }	{ "value": "52" }	{ "value": "M" }	{ "value": "1015.6" }	{ "value": "83.3" }	{ "value": "0" }	{ "
SeaTac#02-10-2022#15:53	{ "value": "02-10-2022" }	{ "value": "53" }	{ "value": "M" }	{ "value": "1015.9" }	{ "value": "72.3" }	{ "value": "5" }	{ "
SeaTac#0...	{ "val...	{ "value...	{ "val...	{ "valu...	{ "val...	{ "val...	{
SeaTac#0...	{ "val...	{ "value...	{ "val...	{ "valu...	{ "val...	{ "val...	{
SeaTac#02-10-2022#18:53	{ "value": "02-10-2022" }	{ "value": "54" }	{ "value": "M" }	{ "value": "1016.2" }	{ "value": "53.1" }	"value": "3" }	"
SeaTac#0...	{ "val...	{ "value...	{ "val...	{ "valu...	{ "val...	{ "val...	{
SeaTac#0...	{ "val...	{ "value...	{ "val...	{ "valu...	{ "val...	{ "val...	{

Rows per page: 30 1 – 24 of 24

```java
public class BigtableWeatherDataLoader
     * @throws IOException
     */
    public static void queryreadingsforseatac(BigtableDataClient dataClient,
    String tableId) throws IOException {
        Filters.Filter filter =FILTERS.chain()
                            .filter(FILTERS.key().regex(regex:".*SeaTac.
                            *02-10-2022.*"));

    Query query = Query.create(TableId.of(tableId)).filter(filter);
    ServerStream<Row> rows = dataClient.readRows(query);
        for (Row row : rows) {
        printRow(row);
        }
    }
```

```
Column Family Time
        value: 13:53 @1732538008372000

Reading data for SeaTac#02-10-2022#14:53
Column Family Date
        value: 02-10-2022 @1732538008372000
Column Family Dewpoint
        value: 52 @1732538008372000
Column Family Gust
        value: M @1732538008372000
Column Family Pressure
        value: 1015.6 @1732538008372000
Column Family Relhum
        value: 83.3 @1732538008372000
Column Family Speed
        value: 0 @1732538008372000
Column Family Temperature
        value: 57 @1732538008372000
Column Family Time
        value: 14:53 @1732538008372000

Reading data for SeaTac#02-10-2022#15:53
Column Family Date
        value: 02-10-2022 @1732538008372000
Column Family Dewpoint
        value: 53 @1732538008372000
Column Family Gust
        value: M @1732538008372000
Column Family Pressure
        value: 1015.9 @1732538008372000
Column Family Relhum
        value: 72.3 @1732538008372000
Column Family Speed
        value: 5 @1732538008372000
```



```sql
select * from weather where _key like '%SeaTac%02-10-2022%';
```

RESULTS

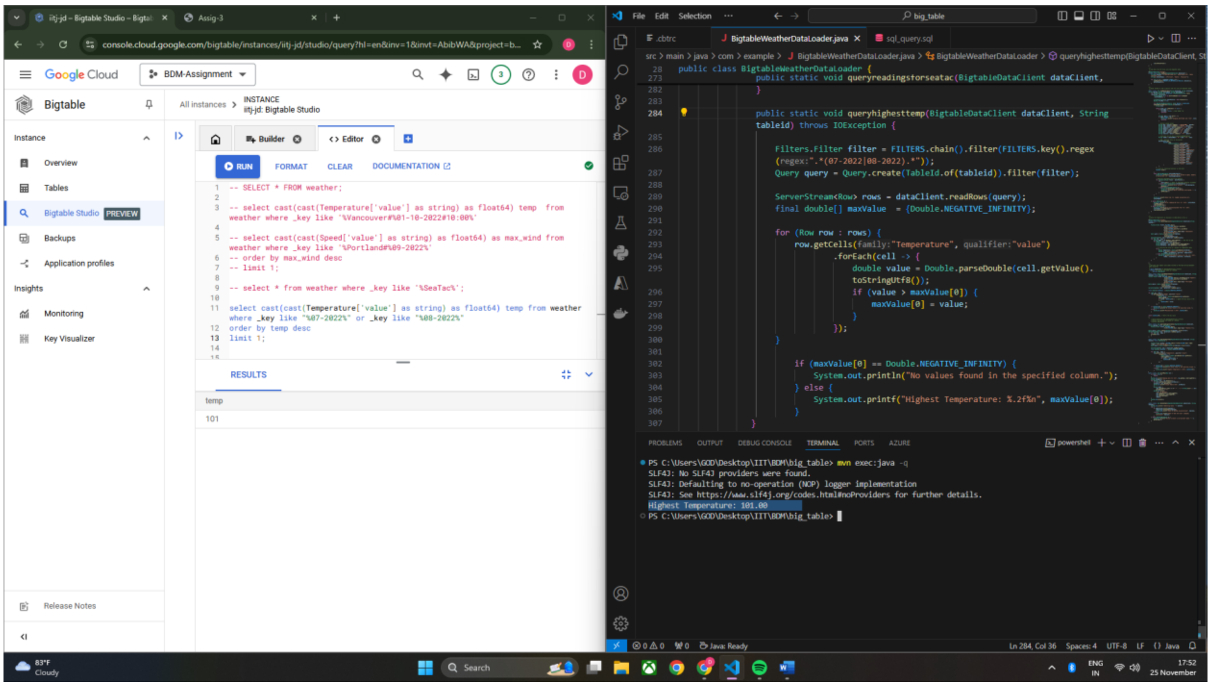
_key	Date	Dewpoint	Gust	Pressure	Relhum	Speed	T
SeaTac#02-10-2022#00:53	{ "value": "02-10-2022" }	{ "value": "53" }	{ "value": "M" }	{ "value": "1014.1" }	{ "value": "47.8" }	{ "value": "9" }	{ "
SeaTac#02-10-2022#01:53	{ "value": "02-10-2022" }	{ "value": "53" }	{ "value": "M" }	{ "value": "1014.1" }	{ "value": "56.7" }	{ "value": "7" }	{ "
SeaTac#02-10-2022#02:53	{ "value": "02-10-2022" }	{ "value": "53" }	{ "value": "M" }	{ "value": "1014.3" }	{ "value": "60.7" }	{ "7" }	{
SeaTac#0...	{ "val...	{ "value...	{ "val...	{ "valu...	{ "val...	{ "val...	{
SeaTac#0...	{ "val...	{ "value...	{ "val...	{ "valu...	{ "val...	{ "val...	{
SeaTac#0...	{ "val...	{ "value...	{ "val...	{ "valu...	{ "val...	{ "val...	{
SeaTac#0...	{ "val...	{ "value...	{ "val...	{ "valu...	{ "val...	{ "val...	{
SeaTac#0...	{ "val...	{ "value...	{ "val...	{ "valu...	{ "val...	{ "val...	{
SeaTac#0...	{ "val...	{ "value...	{ "val...	{ "valu...	{ "val...	{ "val...	{
SeaTac#0...	{ "val...	{ "value...	{ "val...	{ "valu...	{ "val...	{ "val...	{
SeaTac#0...	{ "val...	{ "value...	{ "val...	{ "valu...	{ "val...	{ "val...	{
SeaTac#0...	{ "val...	{ "value...	{ "val...	{ "valu...	{ "val...	{ "val...	{
SeaTac#0...	{ "val...	{ "value...	{ "val...	{ "valu...	{ "val...	{ "val...	{
SeaTac#0...	{ "val...	{ "value...	{ "val...	{ "valu...	{ "val...	{ "val...	{
SeaTac#02-	{	{ "value":	{	{	{	{	{

Rows per page: 30 1 – 24 of 24

```
Reading data for SeaTac#02-10-2022#00:53
Column Family Date
        value: 02-10-2022 @1732538008372000
Column Family Dewpoint
        value: 53 @1732538008372000
Column Family Gust
        value: M @1732538008372000
Column Family Pressure
        value: 1014.1 @1732538008372000
Column Family Relhum
        value: 47.8 @1732538008372000
Column Family Speed
        value: 9 @1732538008372000
Column Family Temperature
        value: 74 @1732538008372000
Column Family Time
        value: 00:53 @1732538008372000

Reading data for SeaTac#02-10-2022#01:53
Column Family Date
        value: 02-10-2022 @1732538008372000
Column Family Dewpoint
        value: 53 @1732538008372000
Column Family Gust
        value: M @1732538008372000
Column Family Pressure
        value: 1014.1 @1732538008372000
Column Family Relhum
        value: 56.7 @1732538008372000
Column Family Speed
        value: 7 @1732538008372000
Column Family Temperature
        value: 69 @1732538008372000
Column Family Time
        value: 01:53 @1732538008372000

Reading data for SeaTac#02-10-2022#02:53
Column Family Date
        value: 02-10-2022 @1732538008372000
Column Family Dewpoint
        value: 53 @1732538008372000
Column Family Gust
        value: M @1732538008372000
Column Family Pressure
        value: 1014.3 @1732538008372000
Column Family Relhum
        value: 60.7 @1732538008372000
Column Family Speed
        value: 7 @1732538008372000
Column Family Temperature
        value: 67 @1732538008372000
Column Family Time
        value: 02:53 @1732538008372000

Reading data for SeaTac#02-10-2022#03:53
```

Query 4

Returns the highest temperature at any station in the summer months of 2022 (July (7), August (8)



```sql
-- SELECT * FROM weather;

-- select cast(cast(Temperature['value'] as string) as float64) temp  from weather where _key like '%Vancouver#%01-10-2022#10:00%'

-- select cast(cast(Speed['value'] as string) as float64) as max_wind from weather where _key like '%Portland#%09-2022%'
-- order by max_wind desc
-- limit 1;

-- select * from weather where _key like '%SeaTac%';

select cast(cast(Temperature['value'] as string) as float64) temp from weather
where _key like "%07-2022%" or _key like "%08-2022%"
order by temp desc
limit 1;
```

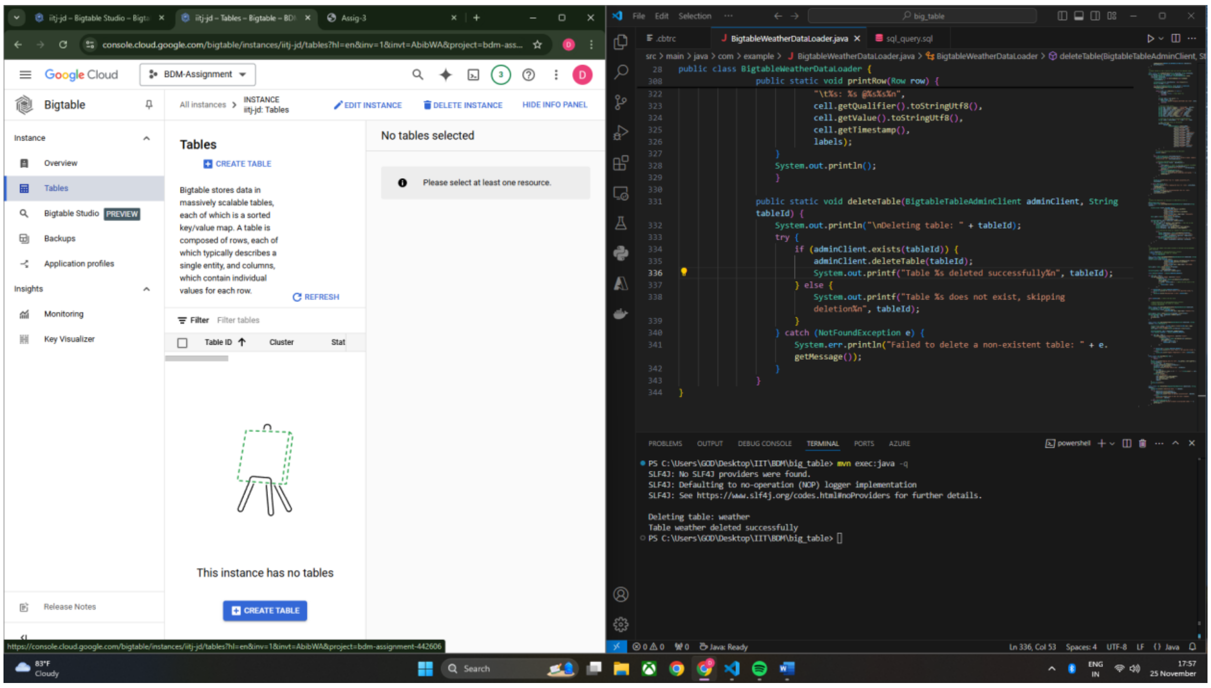
RESULTS

```
temp
101
```

```java
public class BigtableWeatherDataLoader {
        public static void queryreadingsforseatac(BigtableDataClient dataClient,
        }

        public static void queryhighesttemp(BigtableDataClient dataClient, String
        tableid) throws IOException {

            Filters.Filter filter = FILTERS.chain().filter(FILTERS.key().regex
            (regex:".*(07-2022|08-2022).*"));
            Query query = Query.create(TableId.of(tableid)).filter(filter);

            ServerStream<Row> rows = dataClient.readRows(query);
            final double[] maxValue  = {Double.NEGATIVE_INFINITY};

            for (Row row : rows) {
                row.getCells(family:"Temperature", qualifier:"value")
                        .forEach(cell -> {
                            double value = Double.parseDouble(cell.getValue().
                            toStringUtf8());
                            if (value > maxValue[0]) {
                                maxValue[0] = value;
                            }
                        });
            }

                if (maxValue[0] == Double.NEGATIVE_INFINITY) {
                    System.out.println("No values found in the specified column.");
                } else {
                    System.out.printf("Highest Temperature: %.2f%n", maxValue[0]);
                }
        }
```

```
PS C:\Users\GOD\Desktop\IIT\BDM\big_table> mvn exec:java -q
SLF4J: No SLF4J providers were found.
SLF4J: Defaulting to no-operation (NOP) logger implementation
SLF4J: See https://www.slf4j.org/codes.html#noProviders for further details.
Highest Temperature: 101.00
PS C:\Users\GOD\Desktop\IIT\BDM\big_table>
```

Delete Table



Tables

Bigtable stores data in massively scalable tables, each of which is a sorted key/value map. A table is composed of rows, each of which typically describes a single entity, and columns, which contain individual values for each row.

No tables selected

Please select at least one resource.

This instance has no tables

```java
public class BigtableWeatherDataLoader {
        public static void printRow(Row row) {
                        "\t%s: %s @%s%s%n",
                        cell.getQualifier().toStringUtf8(),
                        cell.getValue().toStringUtf8(),
                        cell.getTimestamp(),
                        labels);
            }
            System.out.println();
            }

        public static void deleteTable(BigtableTableAdminClient adminClient, String
        tableId) {
            System.out.println("\nDeleting table: " + tableId);
            try {
                if (adminClient.exists(tableId)) {
                    adminClient.deleteTable(tableId);
                    System.out.printf("Table %s deleted successfully%n", tableId);
                } else {
                    System.out.printf("Table %s does not exist, skipping
                    deletion%n", tableId);
                }
            } catch (NotFoundException e) {
                System.err.println("Failed to delete a non-existent table: " + e.
                getMessage());
            }
        }
}
```

```
PS C:\Users\GOD\Desktop\IIT\BDM\big_table> mvn exec:java -q
SLF4J: No SLF4J providers were found.
SLF4J: Defaulting to no-operation (NOP) logger implementation
SLF4J: See https://www.slf4j.org/codes.html#noProviders for further details.

Deleting table: weather
Table weather deleted successfully
PS C:\Users\GOD\Desktop\IIT\BDM\big_table>
```

Code

```
package com.example;
import com.google.api.core.ApiFuture;
import com.google.api.gax.batching.Batcher;
import com.google.api.gax.batching.BatchingException;
import com.google.cloud.bigtable.admin.v2.BigtableTableAdminClient;
import com.google.cloud.bigtable.data.v2.BigtableDataClient;
import static com.google.cloud.bigtable.data.v2.models.Filters.FILTERS;
import com.google.cloud.bigtable.data.v2.models.Filters;
import com.google.cloud.bigtable.data.v2.models.RowMutationEntry;
import com.google.cloud.bigtable.admin.v2.models.CreateTableRequest;
import java.io.BufferedReader;
import java.io.FileReader;
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.TableId;
import com.google.cloud.bigtable.data.v2.models.Query;
import java.io.IOException;

import com.google.api.gax.rpc.NotFoundException;
import com.google.api.gax.rpc.ServerStream;
import java.util.List;
import java.util.ArrayList;
import java.util.concurrent.ExecutionException;
import java.util.logging.Level;
import java.util.logging.Logger;

@SuppressWarnings("WARNING")
public class BigtableWeatherDataLoader {

  static {
    // Suppress the warning logs from BigtableChannelPrimer
    Logger logger = Logger.getLogger("com.google.cloud.bigtable.data.v2.stub.BigtableChannelPrimer");
    logger.setLevel(Level.SEVERE); // Suppresses WARNING and INFO levels
```

```
}

public static final String PROJECT_ID = "bdm-assignment-442606";
public static final String INSTANCE_ID = "iitj-jd";
public static final String COLUMN_FAMILY = "data"; // Updated to adhere to naming conventions
public static final String TABLE_ID = "weather";
public static final TableId table = TableId.of(TABLE_ID);

public static void main(String[] args) {
  try (BigtableDataClient dataClient = BigtableDataClient.create(PROJECT_ID, INSTANCE_ID);
     BigtableTableAdminClient adminClient = BigtableTableAdminClient.create(PROJECT_ID, INSTANCE_ID)) {

    // Create the table if it does not exist
    createTableIfNotExists(adminClient);
    // Load data from CSV files
    loadData(dataClient);
    // Query the temperature at Vancouver on 2022-10-01 at 10 a.m.
    queryTemperature(dataClient,TABLE_ID);
    // Query the highest wind speed in Portland during September 2022
    queryMaxWindSpeed(dataClient,TABLE_ID);
    // Query data for SeaTac
    queryreadingsforseatac(dataClient,TABLE_ID);
    // Query the highest Temperature for all the cities
    queryhighesttemp(dataClient,TABLE_ID);
    // Delete the created table
    deleteTable(adminClient,TABLE_ID);
    // // Close the Client connection
    dataClient.close();
    adminClient.close();

    } catch (Exception e) {
      System.err.println("Exception while running BigtableWeatherDataLoader: " + e.getMessage());
      e.printStackTrace();
    }
    }

    /**
     * Create the table if it does not already exist.
     */
    @SuppressWarnings("unused")
    private static void createTableIfNotExists(BigtableTableAdminClient adminClient) {
        try {
            if (!adminClient.exists(TABLE_ID)) {
                CreateTableRequest createTableRequest = CreateTableRequest.of(TABLE_ID)
                        .addFamily("Date")
                        .addFamily("Time")
                        .addFamily("Temperature")
                        .addFamily("Dewpoint")
                        .addFamily("Relhum")
                        .addFamily("Speed")
                        .addFamily("Gust")
                        .addFamily("Pressure");
                    adminClient.createTable(createTableRequest);
                System.out.printf("Table '%s' with column family '%s' created successfully.%n", TABLE_ID,
COLUMN_FAMILY);
            } else {
                System.out.printf("Table '%s' already exists.%n", TABLE_ID);
            }
        } catch (Exception e) {
            System.err.println("Error while creating table: " + e.getMessage());
            e.printStackTrace();
        }
    }

    /**
     * Load data from CSV files into the Bigtable table.
     */
    @SuppressWarnings("unused")
    private static void loadData(BigtableDataClient dataClient) {
        String[] datasets = {"data/portland.csv", "data/seatac.csv", "data/vancouver.csv"};
        String[] datasetNames = {"Portland", "SeaTac", "Vancouver"};

        for (int i = 0; i < datasets.length; i++) {
            loadDataFromCsv(PROJECT_ID,INSTANCE_ID,TABLE_ID, datasets[i], datasetNames[i]);
        }
    }

    /**
     * Helper method to load a specific CSV file into the Bigtable table.
     */
    public static void loadDataFromCsv(String projectId, String instanceId, String tableId, String csvFilePath, String
datasetName) {
        try (BigtableDataClient dataClient = BigtableDataClient.create(projectId, instanceId);
             BufferedReader br = new BufferedReader(new FileReader(csvFilePath))) {

            List<ApiFuture<Void>> batchFutures = new ArrayList<>();
            boolean firstRow = true;

            try (Batcher<RowMutationEntry, Void> batcher = dataClient.newBulkMutationBatcher(TableId.of(tableId))) {
                String line;

                while ((line = br.readLine()) != null) {
                    if (firstRow) {
                        firstRow = false; // Skip header row
                        continue;
                    }

                    String[] data = line.split(",");
                    if (data.length < 8) {
                        System.err.printf("Skipping malformed line: %s%n", line);
                        continue;
                    }

                    // Extract column values
                    String date = data[0].isEmpty() ? "00:00:0000" : data[0];
                    String time = data[1].isEmpty() ? "00:00" : data[1];
                    String temperature = data[2].isEmpty() ? "0" : data[2];
                    String dewpoint = data[3].isEmpty() ? "0" : data[3];
                    String relhum = data[4].isEmpty() ? "0" : data[4];
                    String speed = data[5].isEmpty() ? "0" : data[5];
                    String gust = data[6].isEmpty() ? "0" : data[6];
                    String pressure = data[7].isEmpty() ? "0" : data[7];

                    // Construct the row key as Dataset_Name#Date#Time
                    String rowKey = datasetName + "#" + date + "#" + time;

                    // Add a mutation entry to the batcher
                    batchFutures.add(
                        batcher.add(
                            RowMutationEntry.create(rowKey)
                                .setCell("Date", "value", date)
                                .setCell("Time","value",  time)
                                .setCell("Temperature","value",  temperature)
                                .setCell("Dewpoint","value",  dewpoint)
                                .setCell("Relhum","value",  relhum)
                                .setCell("Speed","value",  speed)
                                .setCell("Gust","value",  gust)
                                .setCell("Pressure","value",  pressure)
                        )
                    );
                }
                // Flush any remaining mutations in the batch
                batcher.flush();

            }
            catch (BatchingException batchingException) {
                System.err.println("At least one entry failed to apply. Summary of the errors: \n" + batchingException);
```

```
            // Retrieve individual entry error details
            for (ApiFuture<Void> future : batchFutures) {
                try {
                    future.get(); // Check if individual mutation succeeded
                } catch (ExecutionException entryException) {
                    System.err.println("Entry failure: " + entryException.getCause());
                } catch (InterruptedException e) {
                    Thread.currentThread().interrupt();
                    System.err.println("Batch processing interrupted: " + e.getMessage());
                }
            }
        }

        System.out.printf("Data from '%s' loaded successfully.%n", csvFilePath);

    } catch (IOException e) {
        System.err.printf("Error reading CSV file '%s': %s%n", csvFilePath, e.getMessage());
        e.printStackTrace();
    } catch (Exception e) {
        System.err.printf("Error processing data from '%s': %s%n", csvFilePath, e.getMessage());
        e.printStackTrace();
    }
}



/**
 * Query the temperature at Vancouver on 2022-10-01 at 10 a.m.
 */
@SuppressWarnings("unused")
private static void queryTemperature(BigtableDataClient dataClient,String tableId) {

    Filters.Filter filter =FILTERS.chain()
                        .filter(FILTERS.family().exactMatch("Temperature"))
                        .filter(FILTERS.qualifier().exactMatch("value"))
                    .filter(FILTERS.key().regex("^Vancouver#.*01-10-2022#10:00.*"));
    Query query = Query.create(TableId.of(tableId))
                    .filter(filter)
                    .reversed(true);
    ServerStream<Row> rows = dataClient.readRows(query);
    for (Row row : rows) {
        String rowKey = row.getKey().toStringUtf8(); // Get the row key
        for (RowCell cell : row.getCells()) {
            String columnFamily = cell.getFamily();
            String qualifier = cell.getQualifier().toStringUtf8();
            String cellValue = cell.getValue().toStringUtf8();

            // Try parsing the value as a double for temperature data
            try {
                double temperature = Double.parseDouble(cellValue);
                System.out.printf("Temperature in %s (%s, %s): %.2f%n",
                        rowKey, columnFamily, qualifier, temperature);
            } catch (NumberFormatException e) {
                System.out.printf("Non-numeric value in %s (%s, %s): %s%n",
                        rowKey, columnFamily, qualifier, cellValue);
            }
        }
    }

}
/**
 * Retrieves the highest wind speed recorded in Portland during September 2022 using SQL.
 *
 * @param dataClient The BigtableDataClient instance.
 * @return The maximum wind speed as a double.
     * @throws IOException
*/
public static double queryMaxWindSpeed(BigtableDataClient dataClient, String tableId) throws IOException {
Filters.Filter filter =FILTERS.chain()
```

```
                    .filter(FILTERS.family().exactMatch("Speed"))
                    .filter(FILTERS.qualifier().exactMatch("value"))
                    .filter(FILTERS.key().regex("^Portland#.*09-2022.*"));
Query query = Query.create(TableId.of(tableId))
                  .filter(filter)
                  .reversed(true);
ServerStream<Row> rows = dataClient.readRows(query);

final double[] maxValue  = {Double.NEGATIVE_INFINITY};

for (Row row : rows) {
    row.getCells("Speed", "value")
            .forEach(cell -> {
                double value = Double.parseDouble(cell.getValue().toStringUtf8());
                if (value > maxValue[0]) { // Update the max value in the array
                    maxValue[0] = value;
                }
            });
}
    if (maxValue[0] == Double.NEGATIVE_INFINITY) {
        System.out.println("No values found in the specified column.");
    } else {
        System.out.printf("Maximum Wind: %.2f%n", maxValue[0]);
    }


return maxValue[0]; // Return the max value
}
/*
    * @param dataClient The BigtableDataClient instance.
    * @return The maximum wind speed as a double.
    * @throws IOException
*/
public static void queryreadingsforseatac(BigtableDataClient dataClient, String tableId) throws IOException {
    Filters.Filter filter =FILTERS.chain()
                    .filter(FILTERS.key().regex(".*SeaTac.*02-10-2022.*"));

  Query query = Query.create(TableId.of(tableId)).filter(filter);
  ServerStream<Row> rows = dataClient.readRows(query);
    for (Row row : rows) {
    printRow(row);
    }
  }

  public static void queryhighesttemp(BigtableDataClient dataClient, String tableid) throws IOException {

    Filters.Filter filter = FILTERS.chain().filter(FILTERS.key().regex(".*(07-2022|08-2022).*"));
    Query query = Query.create(TableId.of(tableid)).filter(filter);

    ServerStream<Row> rows = dataClient.readRows(query);
    final double[] maxValue  = {Double.NEGATIVE_INFINITY};

    for (Row row : rows) {
      row.getCells("Temperature", "value")
          .forEach(cell -> {
            double value = Double.parseDouble(cell.getValue().toStringUtf8());
            if (value > maxValue[0]) {
              maxValue[0] = value;
            }
          });
    }

      if (maxValue[0] == Double.NEGATIVE_INFINITY) {
        System.out.println("No values found in the specified column.");
      } else {
        System.out.printf("Highest Temperature: %.2f%n", maxValue[0]);
      }
  }
```

```
public static void printRow(Row row) {
  if (row == null) {
    return;
  }
  System.out.printf("Reading data for %s%n", row.getKey().toStringUtf8());
  String colFamily = "";
  for (RowCell cell : row.getCells()) {
    if (!cell.getFamily().equals(colFamily)) {
    colFamily = cell.getFamily();
    System.out.printf("Column Family %s%n", colFamily);
    }
    String labels =
      cell.getLabels().size() == 0 ? "" : " [" + String.join(",", cell.getLabels()) + "]";
    System.out.printf(
      "\t%s: %s @%s%s%n",
      cell.getQualifier().toStringUtf8(),
      cell.getValue().toStringUtf8(),
      cell.getTimestamp(),
      labels);
  }
  System.out.println();
  }

public static void deleteTable(BigtableTableAdminClient adminClient, String tableId) {
  System.out.println("\nDeleting table: " + tableId);
  try {
    if (adminClient.exists(tableId)) {
      adminClient.deleteTable(tableId);
      System.out.printf("Table %s deleted successfully%n", tableId);
    } else {
      System.out.printf("Table %s does not exist, skipping deletion%n", tableId);
    }
  } catch (NotFoundException e) {
    System.err.println("Failed to delete a non-existent table: " + e.getMessage());
```

```
                }
              }
      }
```