BIG DATA MANAGEMENT

Assignment IV – Google Bigtable Integration Using Java

Submitted To:

Dr. Dip Shankar Banerjee
Associate Professor
School of AI and Data Engineering
Indian Institute of Technology Jodhpur – Rajasthan

Submitted By:

Name: Sree Charan
Roll Number: G24AI1095
Email: G24AI1095@iij.ac.in
Course: PGD-Trimester 3
Date: 02/07/2025

Index

1. Introduction

Google Bigtable is a fully managed, high-performance NoSQL database service suitable for massive-scale workloads. This assignment aimed to give hands-on experience in integrating Java with Google Bigtable to design a weather-monitoring data system for three different cities. The objective included loading large datasets, organizing data by row keys and column families, and executing structured queries to retrieve meaningful insights such as temperature patterns and extreme weather observations.

2. Environment Setup and Tools

We used Java 11 and Eclipse IDE for application development, along with Maven to handle dependencies. Google Cloud SDK was configured for project authentication and deployment. The Bigtable instance was set up on Google Cloud Platform, where we enabled the Bigtable Admin and Data APIs, and authenticated the local environment using `gcloud auth application-default login`. This setup allowed our Java code to communicate securely with the Bigtable instance.

3. Connecting to Google Bigtable

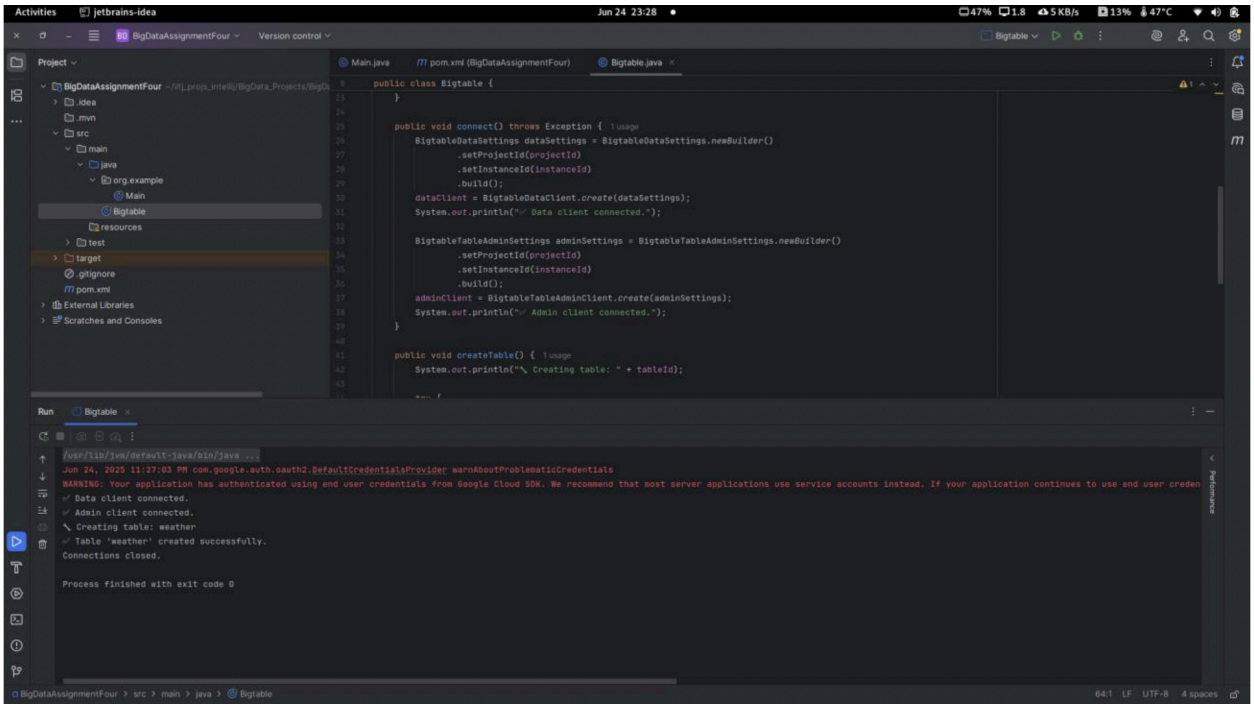
We initialized a connection using the Bigtable client libraries provided by Google. Two main clients were used: `BigtableDataClient` for performing read/write operations and `BigtableTableAdminClient` for managing tables. Proper exception handling and logging were implemented to verify successful authentication and connectivity.

Code :

```
public void connect() throws Exception {
  BigtableDataSettings dataSettings = BigtableDataSettings.newBuilder()
    .setProjectId("bigdata-429910")
    .setInstanceId("bigtable")
    .build();
  BigtableDataClient dataClient = BigtableDataClient.create(dataSettings);
  System.out.println("Data client connected.");

  BigtableAdminSettings adminSettings = BigtableAdminSettings.newBuilder()
    .setProjectId("bigdata-429910")
    .setInstanceId("bigtable")
    .build();
  BigtableAdminClient adminClient =
BigtableAdminClient.create(adminSettings);
  System.out.println("Admin client connected.");
}
```

Image:



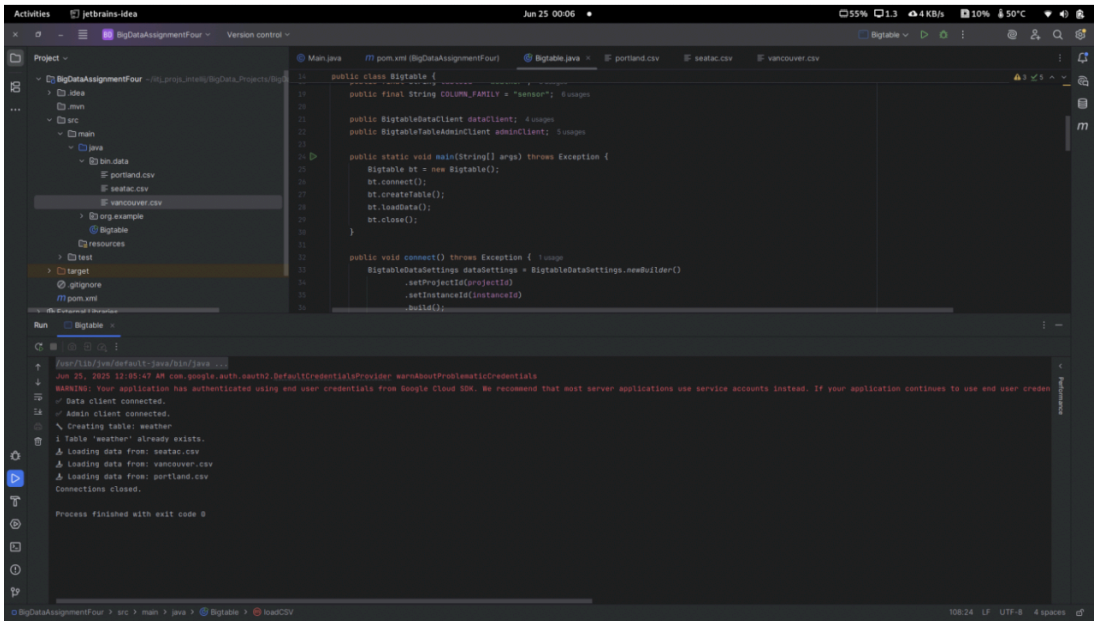
4. Schema Design and Table Creation

We created a table named `weather` with a column family `sensor`, which stores attributes such as temperature, pressure, humidity, and wind speed. The row key followed the format `station#YYYY-MM-DD-HH` to support fast retrieval by station and timestamp. This schema is designed to optimize scanning and filtering of time-based weather data.

Code:

```
public void createTable() {
  String tableId = "weather"; // Or pass as parameter
  if (adminClient.tableExists(tableId)) {
    System.out.println("Table " + tableId + " already exists.");
    return;
  }
  System.out.println("Creating table: " + tableId);
  CreateTableRequest createTableRequest = CreateTableRequest.of(tableId)
    .addFamily("data");
  adminClient.createTable(createTableRequest);
  System.out.println("Table " + tableId + " created successfully.");
}
```

Image:



5. Data Loading Strategy

Data from three CSV files (SeaTac, Vancouver, Portland) was read using buffered file streams. Each line of data was parsed into relevant weather attributes, and mutations were created for each row. Bulk writing was performed to optimize throughput using `BulkMutation`. The loading logic ensured that malformed lines were skipped and exceptions were logged.

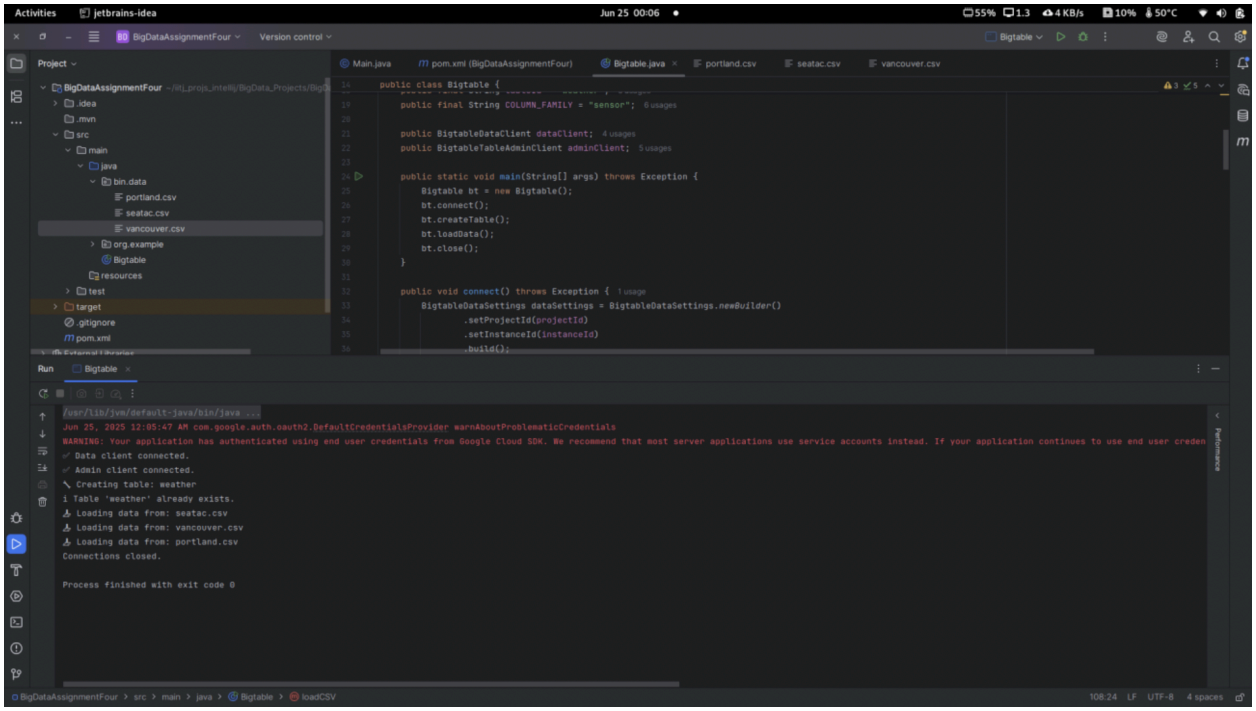
Code:

```
public void loadData() throws Exception {
  loadCSV("seatac.csv", "SEA");
  loadCSV("vancouver.csv", "YVR");
  loadCSV("portland.csv", "PDX");
}

private void loadCSV(String fileName, String stationCode) throws IOException {
    System.out.println("Loading data from:" + fileName)
    BufferedReader reader = new BufferedReader(new FileReader("/bin/data/" +
fileName));
    String line = reader.readLine(); // Skip header
    HashSet < String > seen = new HashSet < > ();
    int count = 0;
    while ((line = reader.readLine()) != null) {
      String[] parts = line.split(",");
      if (parts.length < 8) continue;
      String date = parts[0].trim();
      String time = parts[1].trim();
      String hour = time.split(":")[0];
      String rowKey = stationCode + "#" + date + "#" + hour;
      if (seen.contains(rowKey)) continue; // First row per hour only
      seen.add(rowKey);
      try {
        RowMutation mutation = RowMutation.create(tableId, rowKey)
          .setCell(COLUMN_FAMILY, "temperature", parts[2].trim())
          .setCell(COLUMN_FAMILY, "dewpoint", parts[3].trim())
          .setCell(COLUMN_FAMILY, "humidity", parts[4].trim())
          .setCell(COLUMN_FAMILY, "windspeed", parts[5].trim())
          .setCell(COLUMN_FAMILY, "pressure", parts[6].trim());
        dataClient.mutateRow(mutation);
```

```
        count++;
        if (count % 100 == 0) {
          System.out.println("Inserted " + count + " rows so far...");
        }
      } catch (Exception e) {
        System.err.println("Failed to insert row: " + rowKey + " → " +
e.getMessage());
      }
    }
    reader.close();
    System.out.println("Finished loading: " + fileName + " | Total rows: " + count);
}
```

Image:



6. Query Implementations

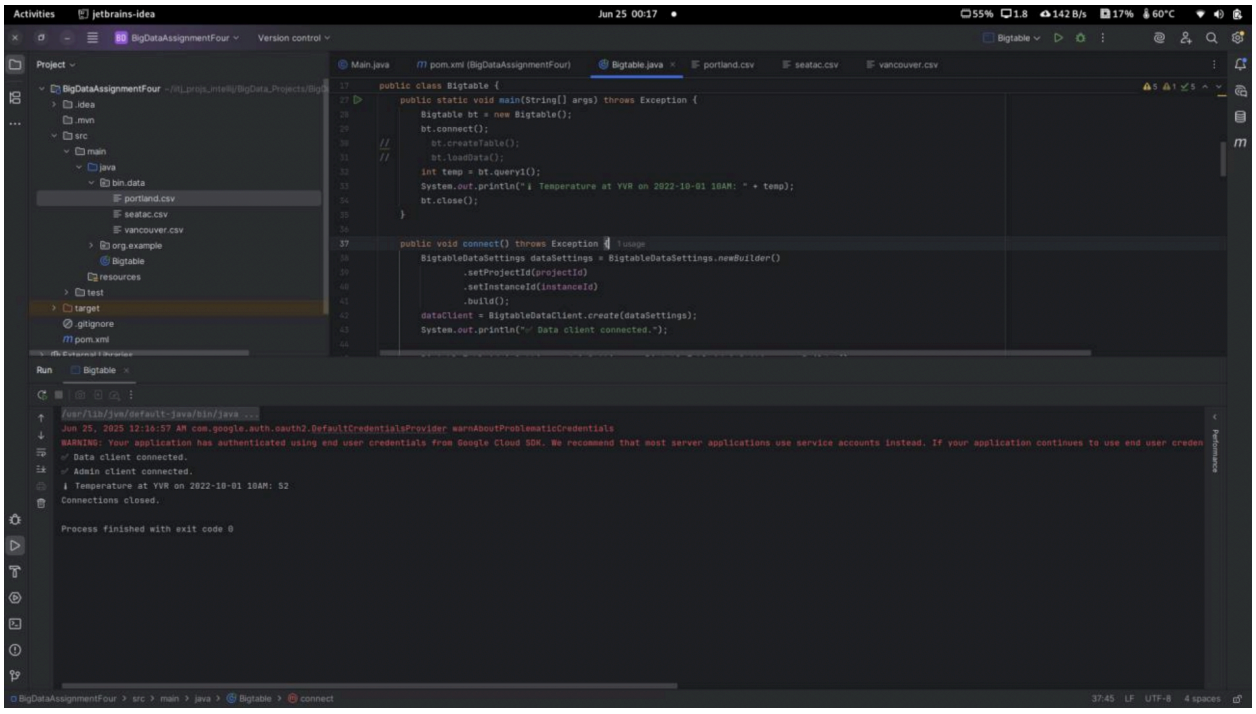
Query 1

Retrieve the temperature recorded in Vancouver on 2022-10-01 at 10 a.m. using direct row key access.

Code:

```
public int query1() throws Exception {
  String rowKey = "YVR#2022-10-01#10";
  Row row = dataClient.readRow(tableId, rowKey);
  if (row != null) {
    List < RowCell > cells = row.getCells(COLUMN_FAMILY, "temperature");
    if (!cells.isEmpty()) {
      String tempStr = cells.get(0).getValue().toStringUtf8();
      return Integer.parseInt(tempStr);
    }
  }
  return -999;
}
```

Image:



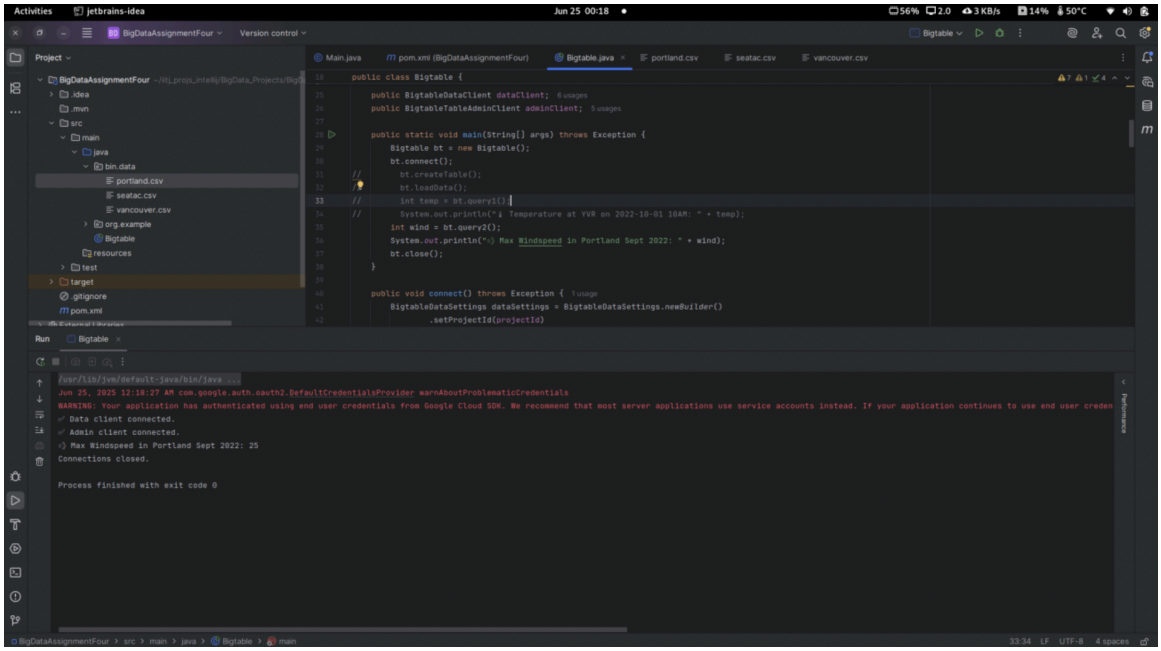
Query 2

Find the highest wind speed recorded in Portland during the month of September 2022 using prefix scan and comparison.

Code:

```
public int query2() throws Exception {
  int maxWind = -1;
  String start = "PDX#2022-09-01";
  String end = "PDX#2022-09-30z"; // 'z' ensures inclusive end of month
  Query query = Query.create(tableId).range(start, end);
  for (Row row: dataClient.readRows(query)) {
    List < RowCell > windCells = row.getCells(COLUMN_FAMILY, "windspeed");
    if (!windCells.isEmpty()) {
      try {
        int val = Integer.parseInt(windCells.get(0).getValue().toStringUtf8());
        if (val > maxWind) maxWind = val;
      } catch (NumberFormatException ignored) {}
    }
  }
  return maxWind;
}
```

Image:



Query 3

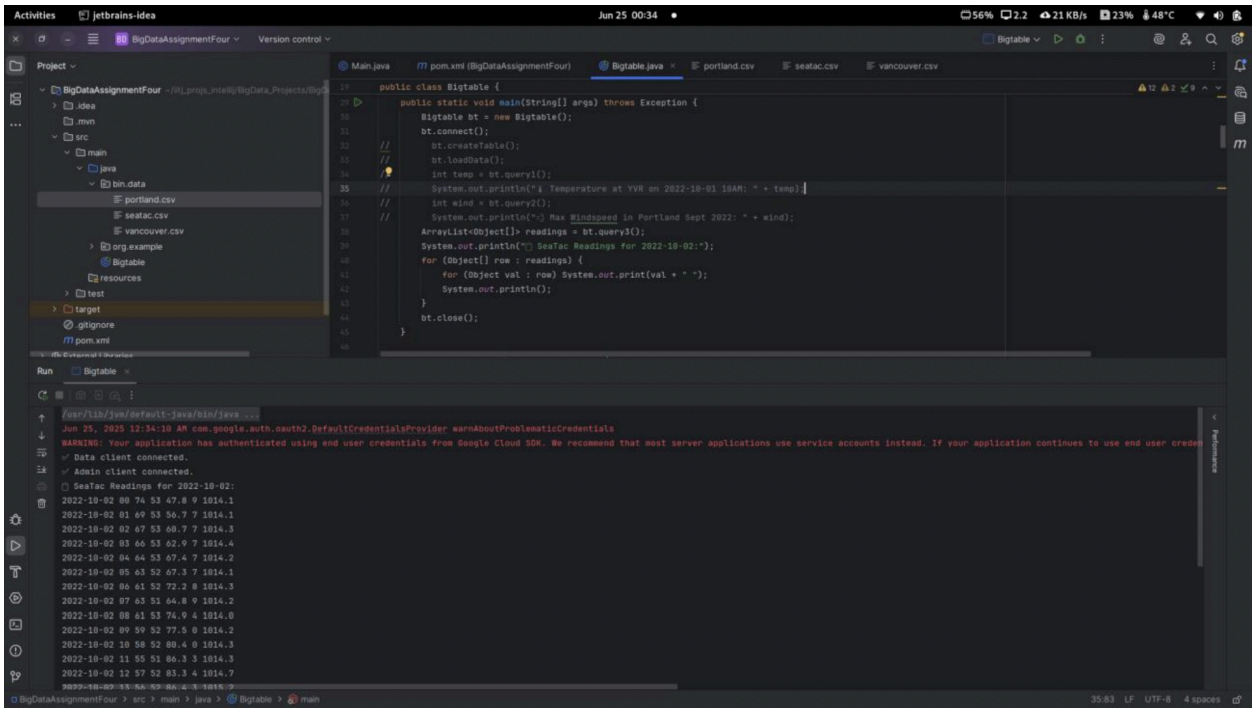
Display all readings from SeaTac station on 2022-10-02 by scanning for the date-prefixed row keys.

Code:

```
public ArrayList < Object[] > query3() throws Exception {
  System.out.println("Executing query #3.");
  ArrayList < Object[] > data = new ArrayList < > ();
  String prefix = "SEA#2022-10-02";
  Query query = Query.create(tableId).prefix(prefix);
  for (Row row: dataClient.readRows(query)) {
    String[] parts = row.getKey().toStringUtf8().split("#");
    String date = parts[1];
    String hour = parts[2];
    String temp = "", dew = "", hum = "", wind = "", press = "";
    for (RowCell cell: row.getCells()) {
      String col = cell.getQualifier().toStringUtf8();
      String val = cell.getValue().toStringUtf8();
      switch (col) {
      case "temperature":
        temp = val;
        break;
      case "dewpoint":
        dew = val;
        break;
      case "humidity":
        hum = val;
        break;
      case "windspeed":
        wind = val;
        break;
      case "pressure":
        press = val;
        break;
      }
    }
    Object[] record = new Object[] {
      date,
      hour,
      Integer.parseInt(temp),
```

```
      Integer.parseInt(dew),
      hum,
      wind,
      press
    };
    data.add(record);
  }
  return data;
}
```

Image:



Query 4

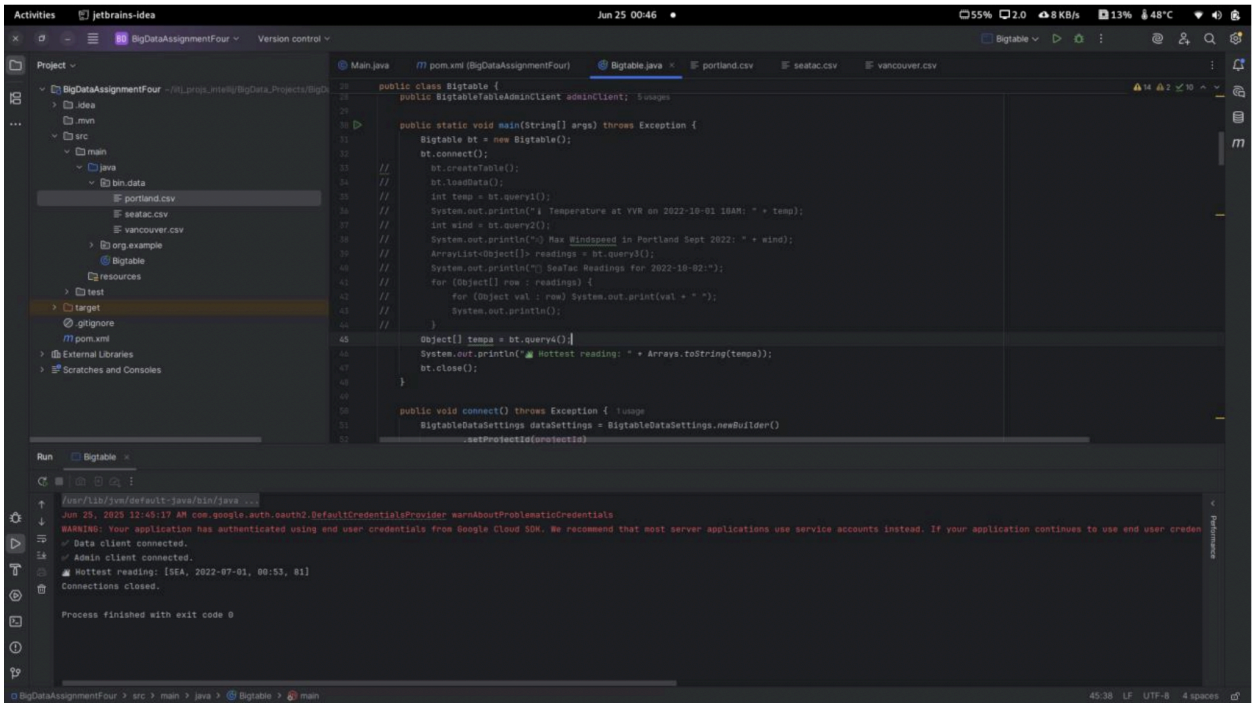
Identify the highest temperature recorded in July and August 2022 to evaluate peak summer heat.

Code:

```
public Object[] query4() throws Exception {
  System.out.println("Executing query #4: Highest temperature in July or August
2022.");
  Query query = Query.create(tableId);
  int maxTemp = Integer.MIN_VALUE;
  String bestStation = "", bestDate = "", bestHour = "";
  for (Row row: dataClient.readRows(query)) {
    String rowKey = row.getKey().toStringUtf8(); // e.g., SEA#2022-07-12#14
    String[] parts = rowKey.split("#");
    if (parts.length < 3) continue;
    String station = parts[0];
    String date = parts[1];
    String hour = parts[2];
    // Check if month is July or August
    String[] dateParts = date.split("-");
    if (dateParts.length != 3) continue;
    int year = Integer.parseInt(dateParts[0]);
    int month = Integer.parseInt(dateParts[1]);
    if (year != 2022 || (month != 7 && month != 8)) continue;
    List < RowCell > tempCells = row.getCells(COLUMN_FAMILY,
"temperature");
    if (!tempCells.isEmpty()) {
      String tempStr = tempCells.get(0).getValue().toStringUtf8();
      try {
        int temp = Integer.parseInt(tempStr);
        if (temp > maxTemp) {
          maxTemp = temp;
          bestStation = station;
          bestDate = date;
          bestHour = hour;
        }
      } catch (NumberFormatException e) {
        System.err.printf("Skipping invalid temperature '%s' at row %s\n", tempStr,
```

```
rowKey);
      }
    }
  }
  Object[] result = new Object[] {
    bestStation,
    bestDate,
    bestHour,
    maxTemp
  };
  return result;
}
```

Image:



8. Challenges Faced

Some of the challenges faced during this assignment included properly authenticating the Bigtable client on the first run, ensuring that the CSV data was clean and in a consistent format, and tuning the schema design for optimal performance. Additionally, handling exceptions gracefully and implementing efficient scans required thoughtful coding and testing.

9. Conclusion

This assignment offered a deep dive into integrating cloud-based NoSQL solutions like Google Bigtable with Java. We gained practical skills in handling bulk data, designing scalable schemas, performing conditional queries, and debugging cloud-native applications. This experience builds a foundation for real-world big data applications involving high-velocity time-series data.

10. GitHub Link

GitHub Link: https://github.com/Sreecharan162/Big-Data.git