

Big data Management

Assignment 4

Jeyadev L

G23AI2071

## Main Function

A screen shot of a computer program

Description automatically generated

## Create Table Function

A screen shot of a computer code

Description automatically generated

## Load Data Function

A screen shot of a computer program

Description automatically generated

A screen shot of a computer program

Description automatically generated

## Table Creation and Data load

A screenshot of a computer screen

Description automatically generated

## Query 1-

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

A screenshot of a computer

Description automatically generated

## Query 2

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

A screenshot of a computer screen

Description automatically generated

## Query 3

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

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

A screenshot of a computer screen

Description automatically generated

A screenshot of a computer screen

Description automatically generated

## Query 4

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

A screenshot of a computer screen

Description automatically generated

## Delete Table

A screenshot of a computer screen

Description automatically generated

## 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());

                    }

                }

}