Homework Problem Set G Submission Form

# Overview

|  |  |
| --- | --- |
| Your Name | Sana Khan |
| Your SU Email | Skhan53@syr.edu |

# Instructions

Put your name and SU email at the top. Answer these questions all from the lab. When asked to include screenshots, please follow the screenshot guidelines from the first homework.

Remember as you complete the homework that it is not only about getting it right/correct. We will discuss the answers in class so it’s important to articulate anything you would like to contribute to the discussion in your answer:

* If you feel the question is vague, include any assumptions you've made.
* If you feel the answer requires interpretation or justification, provide it.
* If you do not know the answer to the question, articulate what you tried and how you are stuck.
* Highlight any doubts or questions you would like me to review.

This how you receive credit for answering questions that might not be correct. In addition, you must complete the reflection portion of the homework assignment for full credit. Since most answers will be similar this is an important part of your individual submission.

Complete Part II of this document first, then go back and complete the Reflection in Part I.

# Part I: Reflection

Use this section to reflect on your learning. To achieve the highest grade on the assignment, you must be as descriptive and personal as possible with your reflection.

1. As you completed this assignment, identify what you learned.
2. What barriers or challenges did you encounter while completing this assignment?
3. How prepared were you to complete this assignment? What can you do to be better prepared?
4. Rate your comfort level with this week’s material. Use the rubric provided.

4 ==> I understand this material and can explain it to others.  
3 ==> I understand this material.  
2 ==> I somewhat understand the material but sometimes need guidance from others.  
1 ==> I understand very little of this material and need extra help.

# Part II: Questions

Your employer (weather.com) would like you store weather sensor and forecast data. Eventually you will get readings from 2,000 cities worldwide every minute. That's 2.88 million rows each day and 1 billion rows a year! Since the data does not need to be read immediately when written across all nodes, you decide Cassandra is a good choice for this project! This data will be accessible by users so they can get weather information and historical trends for they cities they live in and visit. This should help you figure out how the data will be queried.

**QUESTIONS:   
  
For each question, include a copy of the code required to complete the question along with a screenshot of the code and a screenshot of the output.**

1. InCQL, create a keyspace called **glab** with a replication factor of 1 and a simple replication strategy. Use the keyspace.

**CREATE KEYSPACE glab WITH REPLICATION = {'class':'SimpleStrategy', 'replication\_factor':1};**

**USE glab**

1. In Spark, set up a Spark session that is ready to talk with Cassandra.

**import pyspark**

**from pyspark.sql import SparkSession**

**from pyspark.sql.functions import \***

**from pyspark.sql.types import \***

**# CASSANDRA CONFIGURATION**

**cassandra\_host = "cassandra"**

**spark = SparkSession \**

**.builder \**

**.master("local") \**

**.appName('jupyter-pyspark') \**

**.config("spark.cassandra.connection.host", cassandra\_host) \**

**.config("spark.jars.packages","com.datastax.spark:spark-cassandra-connector-assembly\_2.12:3.1.0")\**

**.getOrCreate()**

**sc = spark.sparkContext**

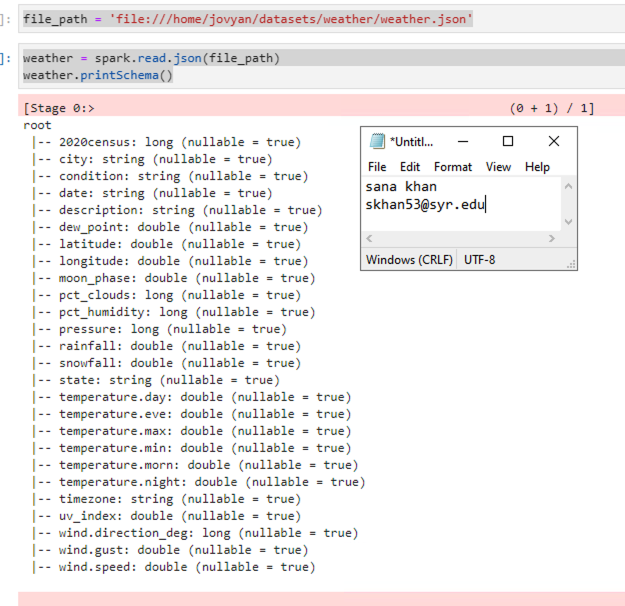
**sc.setLogLevel("ERROR")**

1. To deal with the amount of data associated with the weather.com data set, you decide to start with a smaller sample data set. The data set contains 7 days of weather information for major U.S. cities, with one row being weather information for a single city on a single day. Load the data set located at **/home/jovyan/datasets/weather/weather.json** and use printSchema() to inspect the schema.

**file\_path = 'file:///home/jovyan/datasets/weather/weather.json'**

**weather = spark.read.json(file\_path)**

**weather.printSchema()**



1. Look at rows of data in the sample data set. Profile the data to determine what should be used as the partition and cluster key:
   1. First: Find the minimal candidate key. Which columns serve as a key for each row?
      1. The city and state + date
   2. Next: Prove your key works. In Spark:
      1. Get a count of rows in the entire DataFrame.
      2. Get a count rows when you select your key columns and use distinct() to remove duplicates.
      3. If the row counts are the name, that’s a candidate key. Include the code and output in the screenshot.
         1. **weather.count()**
         2. **weather.select('date').distinct().count()**
         3. **weather.select('city','state').distinct().count()**

**A screenshot of a computer

Description automatically generated with medium confidence**

* 1. A Cassandra row key consists of a partition and cluster key.   
     For this example, use the column that will guarantee to be storing data in increasing order over time (append only) as your cluster key. The other column (or columns) should be the partition key.

1. With your keys figured out, it’s time to create your table. Using the CQL Shell, write an CQL query to create a table called **daily\_city\_weather**. Include all columns in the source data set, and make sure to set your partition and cluster keys, as designed. Show the CQL query and the output in the screenshot. Include an additional screenshot of the describe command on this table.  
   ADVICE: Write your create table in a text editor then paste it into CQL, as the command line can be a tad unforgiving.

create\_table\_sql = '''

CREATE TABLE glab.daily\_city\_weather

(

"2020census" int,

city text,

condition text,

weatherdate text,

description text,

dew\_point decimal,

latitude decimal,

intitude decimal,

moon\_phase decimal,

pct\_clouds int,

pct\_humidity int,

pressure int,

rainfall decimal,

snowfall decimal,

state text,

temperature\_day decimal,

temperature\_eve decimal,

temperature\_max decimal,

temperature\_min decimal,

temperature\_morn decimal,

temperature\_night decimal,

timezone text,

uv\_index decimal,

wind\_direction\_deg int,

wind\_gust decimal,

wind\_speed decimal,

PRIMARY KEY ((state, city), weatherdate)

);

'''

from cassandra.cluster import Cluster

with Cluster([cassandra\_host]) as cluster:

session = cluster.connect()

session.execute(create\_table\_sql)

print(create\_table\_sql)

A screenshot of a computer program

Description automatically generated with medium confidence

1. Write Spark code to save the JSON DataFrame into your Cassandra table. Make sure the column names are the same. Read the data back out and make sure you have the same number of rows in the DataFrame and in the Cassandra table. This will be further proof that your Cassandra row key is set up correctly. Provide Spark code to save the data to Cassandra and then a screenshot of a select statement and output in the CQL shell.

**weather\_2 = weather.toDF("2020census",**

**"city",**

**"condition",**

**"weatherdate",**

**"description",**

**"dew\_point",**

**"latitude",**

**"intitude",**

**"moon\_phase",**

**"pct\_clouds",**

**"pct\_humidity",**

**"pressure",**

**"rainfall",**

**"snowfall",**

**"state",**

**"temperature\_day",**

**"temperature\_eve",**

**"temperature\_max",**

**"temperature\_min",**

**"temperature\_morn",**

**"temperature\_night",**

**"timezone",**

**"uv\_index",**

**"wind\_direction\_deg",**

**"wind\_gust",**

**"wind\_speed");**

**weather\_2.write.format("org.apache.spark.sql.cassandra")\**

**.mode("Append")\**

**.option("table", "weather")\**

**.option("keyspace", "glab")\**

**.save()**

**weather\_3 =spark.read.format("org.apache.spark.sql.cassandra")\**

**.option("table", "weather")\**

**.option("keyspace","glab")\**

**.load()**

**weather\_3.count()**

**A screenshot of a computer

Description automatically generated with medium confidence**

1. Write a CQL shell query to get the condition, description, and daytime temperatures for "Syracuse, NY"; include all dates.

**SELECT city, state, condition, description, temperature\_day FROM weather WHERE city = 'Syracuse' and state =**

**'New York';**

**A screenshot of a computer

Description automatically generated with medium confidence**

1. Write the same query as Question 7, but using Spark SQL. Register the data from Cassandra as the Temp View **daily\_city\_weather**, then use Spark SQL to filter on “Syracuse, NY.” Instead of showing the output, **explain()** the Spark query to prove the filter is being passed-through to Cassandra. (The filter should NOT be happening in Spark—Welcome to big data country!)

**weather\_3.createOrReplaceTempView("weather")**

**query = '''**

**SELECT city, state, condition, description, temperature\_day FROM weather WHERE city = 'Syracuse' and state =**

**'New York';**

**'''**

**spark.sql(query).explain()**

****

1. Your company would like to now allow users to find cities where it is raining on a specific date. Specifically, they would like a query to show the city and state name, date, condition, and description for only those cities where its not raining on the given date. Write this query in Spark or Spark SQL. Which Cassandra filters are used? Show with explain and highlight in your screenshot.

**query = '''**

**SELECT city, state, condition, description, temperature\_day FROM weather WHERE condition = 'Rain' and weatherdate =**

**'2021-08-07';**

**'''**

**spark.sql(query).explain()**

A screenshot of a computer

Description automatically generated with low confidence

1. Run the same query in Question 9 from the CQL command line; obviously it requires ALLOW FILTERING. Figure out how you can do an index or materialized view to avoid a costly ALLOW FILTERING operation. Include your CQL to create the index or materialized view and then include a query demonstrating it works in CQL.   
   NOTE: Our version of Cassandra and the Spark Connector does not support materialized views.

A picture containing text, screenshot, font

Description automatically generated