

Problem Set G Submission Form

Overview

Your Name	Hendi Kushta
Your SU Email	hkushta@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 screen shot guidelines from the first homework.

Remember as you complete the homework 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 which 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.

Create properly designed Cassandra tables with adequate cluster and partition keys. Use Apache Spark to query, import, and export data from Cassandra. Create indexes and materialized views over Cassandra data to improve query performance and avoid ALLOW FILTERING activity.

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.

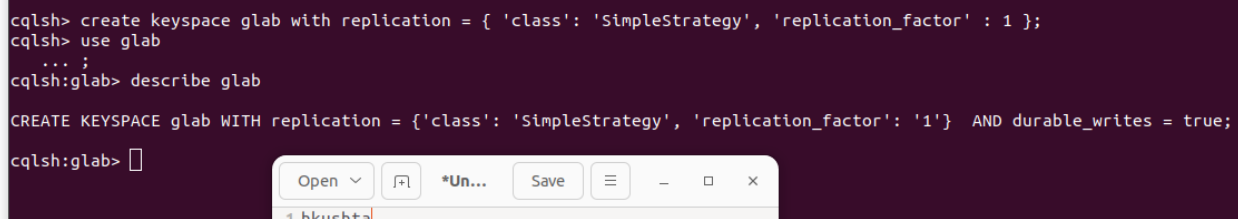
Part II – Questions

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.

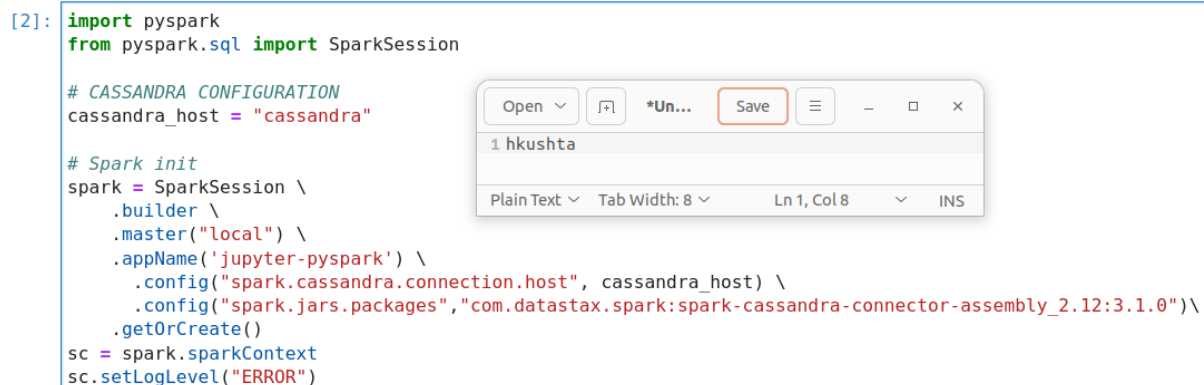
1. InCQL, create a Keyspace called **glab** with a replication factor of 1 and a Simple replication strategy. Use the keyspace.



```
cqlsh> create keyspace glab with replication = { 'class': 'SimpleStrategy', 'replication_factor' : 1 };
cqlsh> use glab
... ;
cqlsh:glab> describe glab

CREATE KEYSPACE glab WITH replication = {'class': 'SimpleStrategy', 'replication_factor': '1'} AND durable_writes = true;
cqlsh:glab> 
```

2. In Spark, setup a spark session that is ready to talk with Cassandra.



```
[2]: import pyspark
from pyspark.sql import SparkSession

# CASSANDRA CONFIGURATION
cassandra_host = "cassandra"

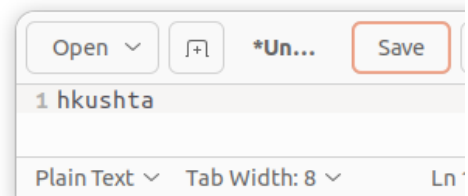
# Spark init
spark = SparkSession \
    .builder \
    .master("local") \
    .appName('jupyter-pyspark') \
    .config("spark.cassandra.connection.host", cassandra_host) \
    .config("spark.jars.packages", "com.datastax.spark:spark-cassandra-connector-assembly_2.12:3.1.0") \
    .getOrCreate()
sc = spark.sparkContext
sc.setLogLevel("ERROR")
```

3. To deal with the amount of data associated with the weather.com dataset, you decide to start with a smaller sample data set. The dataset contains 7 days of weather information for major US Cities, with one row being weather information for

a single city on a single day. Load the dataset Located at **/home/jovyan/datasets/weather/weather.json** and use `printSchema()` to inspect the schema.

```
weather = spark.read.json("file:///home/jovyan/datasets/weather/weather.json")
weather.printSchema()
```

```
root
|-- 2020census: long (nullable = true)
|-- city: string (nullable = true)
|-- condition: string (nullable = true)
|-- date: string (nullable = true)
|-- description: string (nullable = true)
|-- dew_point: double (nullable = true)
|-- latitude: double (nullable = true)
|-- longitude: double (nullable = true)
|-- moon_phase: double (nullable = true)
|-- pct_clouds: long (nullable = true)
|-- pct_humidity: long (nullable = true)
|-- pressure: long (nullable = true)
|-- rainfall: double (nullable = true)
|-- snowfall: double (nullable = true)
|-- state: string (nullable = true)
|-- temperature.day: double (nullable = true)
|-- temperature.eve: double (nullable = true)
|-- temperature.max: double (nullable = true)
|-- temperature.min: double (nullable = true)
|-- temperature.morn: double (nullable = true)
|-- temperature.night: double (nullable = true)
|-- timezone: string (nullable = true)
|-- uv_index: double (nullable = true)
|-- wind.direction_deg: long (nullable = true)
|-- wind.gust: double (nullable = true)
|-- wind.speed: double (nullable = true)
```



4. 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:
 - a. First: Find the minimal candidate key - which columns serve as a key for each row?
NOTE: You can NOT use 2020census as that is a population figure and coincidentally unique.

Just by trying to understand the data, we can say that a candidate key can be composed by date, city, and state. There are 1600 distinct records, when we select distinct this 3 columns, which is the same number as the number of rows in our dataset.

- b. Next: Prove your key works, in Spark:
- Get a count of rows in the entire DataFrame.



```
: print(weather.count())  
1600
```

- Get a count rows when you select your key columns and use distinct() to remove duplicates.



```
print(weather.select("date").distinct().count())  
[Stage 4:=====] (192 + 1) / 200]  
8  
  
print(weather.select("city", "state").distinct().count())  
[Stage 7:=====] (172 + 1) / 200]  
200  
  
print(weather.select("date", "city", "state").distinct().count())  
[Stage 10:=====] (184 + 1) / 200]  
1600
```

- If the row counts are the name, that's a candidate key. include the code and output in the screenshot.



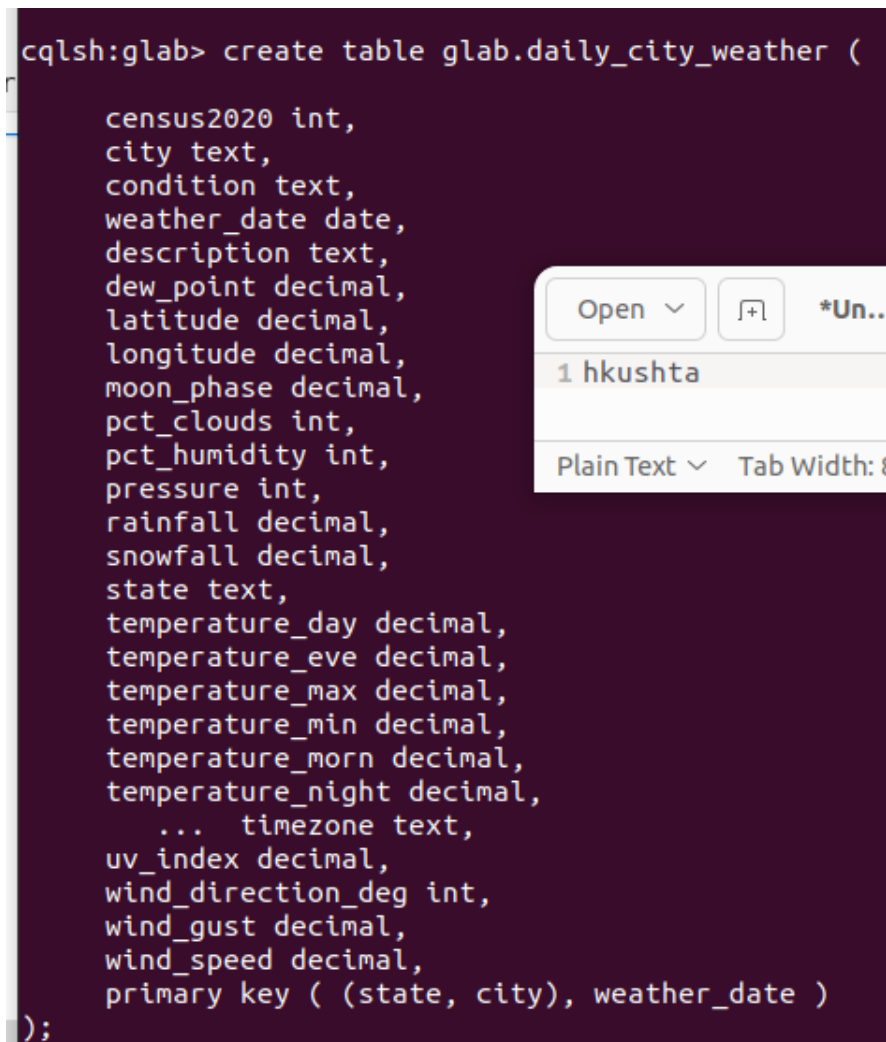
```
print(weather.select("date").distinct().count())  
[Stage 4:=====] (192 + 1) / 200]  
8  
  
print(weather.select("city", "state").distinct().count())  
[Stage 7:=====] (172 + 1) / 200]  
200  
  
print(weather.select("date", "city", "state").distinct().count())  
[Stage 10:=====] (184 + 1) / 200]  
1600
```

- c. 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

In this case, the partition key will be State and City, while the cluster key will be date since it is the column that when storing data, it will be increased by time.

5. With your keys figured out, its 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.



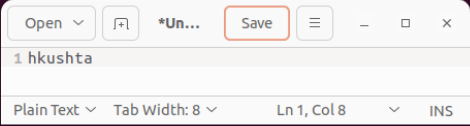
```
cqlsh:glab> create table glab.daily_city_weather (
    census2020 int,
    city text,
    condition text,
    weather_date date,
    description text,
    dew_point decimal,
    latitude decimal,
    longitude decimal,
    moon_phase decimal,
    pct_clouds int,
    pct_humidity int,
    pressure 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), weather_date )
);
```

```

cqlsh:glab> desc daily_city_weather

CREATE TABLE glab.daily_city_weather (
  state text,
  city text,
  weather_date date,
  census2020 int,
  condition text,
  description text,
  dew_point decimal,
  latitude decimal,
  longitude decimal,
  moon_phase decimal,
  pct_clouds int,
  pct_humidity int,
  pressure int,
  rainfall decimal,
  snowfall decimal,
  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), weather_date)
) WITH CLUSTERING ORDER BY (weather_date ASC)
  AND bloom_filter_fp_chance = 0.01
  AND caching = {'keys': 'ALL', 'rows_per_partition': 'NONE'}
  AND comment = ''
  AND compaction = {'class': 'org.apache.cassandra.db.compaction.SizeTieredCompactionStrategy', 'max_threshold': '32', 'min_thresho
ld': '4'}
  AND compression = {'chunk_length_in_kb': '64', 'class': 'org.apache.cassandra.io.compress.LZ4Compressor'}
  AND crc_check_chance = 1.0
  AND dclocal_read_repair_chance = 0.1
  AND default_time_to_live = 0
  AND gc_grace_seconds = 864000
  AND max_index_interval = 2048
  AND memtable_flush_period_in_ms = 0
  AND min_index_interval = 128
  AND read_repair_chance = 0.0
  AND speculative_retry = '99PERCENTILE';

```



- 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 setup correctly. Provide spark code to save the data to Cassandra and then a screenshot of a select statement and output in the CQL Shell.

```
w = weather.toDF( "census2020", "city",
    "condition",
    "weather_date",
    "description",
    "dew_point",
    "latitude",
    "longitude",
    "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")
```

```
: w.write.format("org.apache.spark.sql.cassandra") \
    .mode("Append") \
    .option("table", "daily_city_weather") \
    .option("keyspace", "glab") \
    .save()
```

```
cqlsh:glab> select * from daily_city_weather limit 5
... ;
```

state	city	weather_date	census2020	condition	description	dew_point	latitude	longitude	moon_phase
pct_clouds	pct_humidity	pressure	rainfall	snowfall	temperature_day	temperature_eve	temperature_max	temperature_min	temperature_morn
temperature_night	timezone	uv_index	wind_direction_deg	wind_gust	wind_speed				
California	Oxnard	2021-10-19	202063	Clear	clear sky	40.68	34.1976308	-119.1803818	0.47
0	49	1017	0.0	0.0	62.04	64.31	64.31	64.31	56.44
56.7	58.3	America/Los_Angeles	5.0	281	9.17	7.83			
California	Oxnard	2021-10-20	202063	Clouds	few clouds	45.37	34.1976308	-119.1803818	0.5
22	55	1019	0.0	0.0	63.79	61.75	64.09	64.09	57.31
57.65	59.97	America/Los_Angeles	5.05	253	11.25	9.26			
California	Oxnard	2021-10-21	202063	Clouds	broken clouds	43.56	34.1976308	-119.1803818	0.54
82	49	1019	0.0	0.0	67.3	63.97	67.3	67.3	59.7
60.73	61.75	America/Los_Angeles	4.63	249	7.83	8.14			
California	Oxnard	2021-10-22	202063	Clouds	broken clouds	52.36	34.1976308	-119.1803818	0.57
80	69	1016	0.0	0.0	64.87	64.65	66.27	66.27	59.88
60.08	61.79	America/Los_Angeles	4.94	266	19.91	13.15			
California	Oxnard	2021-10-23	202063	Clouds	overcast clouds	53.83	34.1976308	-119.1803818	0.6
99	76	1012	0.0	0.0	62.33	64.15	64.15	64.15	60.3
60.53	60.3	America/Los_Angeles	4.99	256	14.36	11.3			

```
(5 rows)
```

```
cqlsh:glab> SELECT COUNT(*) FROM daily_city_weather;
```

```
count
-----
1600
```

```
(1 rows)
```


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

```
cqlsh:glab> SELECT city, state, weather_date, condition, description, temperature_day
FROM daily_city_weather
WHERE city = 'Syracuse' AND state = 'New York';
```

city	state	weather_date	condition	description	temperature_day
Syracuse	New York	2021-10-19	Clear	clear sky	57.47
Syracuse	New York	2021-10-20	Clouds	few clouds	64.71
Syracuse	New York	2021-10-21	Rain	moderate rain	63.57
Syracuse	New York	2021-10-22	Rain	light rain	50.63
Syracuse	New York	2021-10-23	Rain	light rain	50.52
Syracuse	New York	2021-10-24	Rain	light rain	44.62
Syracuse	New York	2021-10-25	Rain	light rain	44.19
Syracuse	New York	2021-10-26	Rain	moderate rain	48.83

(8 rows)

8. Write the same query as 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!)

```
w.createOrReplaceTempView("daily_city_weather")

query = '''
SELECT city, state, weather_date, condition, description, temperature_day
FROM daily_city_weather
WHERE city = 'Syracuse' AND state = 'New York';
'''

spark.sql(query).explain()

== Physical Plan ==
*(1) Project [city#8, state#21, date#10 AS weather_date#126, condition#9, description#11, temperature.day#22 AS temperature_day#138]
+- *(1) Filter (((isnotnull(city#8) AND isnotnull(state#21)) AND (city#8 = Syracuse)) AND (state#21 = New York))
   +- FileScan json [city#8,condition#9,date#10,description#11,state#21,temperature.day#22] Batched: false, DataFilters: [isnotnull(city#8), isnotnull(state#21), (city#8 = Syracuse), (state#21 = New York)], Format: JSON, Location: InMemoryFileIndex[file:/home/jovyan/datasets/weather/weather.json], PartitionFilters: [], PushedFilters: [IsNotNull(city), IsNotNull(state), EqualTo(city,Syracuse), EqualTo(state,New York)], ReadSchema: struct<city:string,condition:string,date:string,description:string,state:string,temperature.day:d...
```

9. 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, weather_date, condition, description
FROM daily_city_weather
WHERE condition = 'Rain' and weather_date = '2021-10-23';
'''

spark.sql(query).toPandas()
```

	city	state	weather_date	condition	description
0	New York	New York	2021-10-23	Rain	light rain
1	Houston	Texas	2021-10-23	Rain	light rain
2	Philadelphia	Pennsylvania	2021-10-23	Rain	light rain
3	San Antonio	Texas	2021-10-23	Rain	light rain
4	San Diego	California	2021-10-23	Rain	light rain
...
85	Olathe	Kansas	2021-10-23	Rain	light rain
86	West Valley City	Utah	2021-10-23	Rain	light rain
87	Warren	Michigan	2021-10-23	Rain	light rain
88	Pasadena	California	2021-10-23	Rain	light rain
89	Waco	Texas	2021-10-23	Rain	light rain

90 rows × 5 columns

```
query = '''
SELECT city, state, weather_date, condition, description
FROM daily_city_weather
WHERE condition = 'Rain' and weather_date = '2021-10-23';
'''

spark.sql(query).explain()
```

```
== Physical Plan ==
*(1) Project [city#8, state#21, date#10 AS weather_date#126, condition#9, description#11]
+- *(1) Filter (((isnotnull(condition#9) AND isnotnull(date#10)) AND (condition#9 = Rain)) AND (date#10 = 2021-10-23))
   +- FileScan json [city#8,condition#9,date#10,description#11,state#21] Batched: false, DataFilters: [isnotnull(condition#9), isnotnull(date#10), (condition#9 = Rain), (date#10 = 2021-10-23)], Format: JSON, Location: InMemoryFileIndex[file:/home/jovyan/datasets/weather/weather.json], PartitionFilters: [], PushedFilters: [IsNotNull(condition), IsNotNull(date), EqualTo(condition,Rain), EqualTo(date,2021-10-23)], ReadSchema: struct<city:string,condition:string,date:string,description:string,state:string>
```

The cassandra filter that is used is only the cluster key which in this case is weather_date.

10. Run the same query in 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.

```
InvalidRequest: Error from server: code=2200 [Invalid query] message= Primary key column 'city' is required
cqlsh:glab> CREATE MATERIALIZED VIEW daily_city_weather_by_condition_date AS
  SELECT city, state, weather_date, condition, description
  FROM daily_city_weather
  WHERE condition IS NOT NULL AND weather_date IS NOT NULL AND city IS NOT NULL AND state IS NOT NULL
  PRIMARY KEY ((condition, weather_date), city, state);

Warnings :
Materialized views are experimental and are not recommended for production use.

cqlsh:glab> SELECT city, state, weather_date, condition, description
FROM daily_city_weather_by_condition_date
WHERE condition = 'Rain' AND weather_date = '2021-10-23';
```

city	state	weather_date	condition	description
Akron	Ohio	2021-10-23	Rain	moderate rain
Anchorage	Alaska	2021-10-23	Rain	light rain
Austin	Texas	2021-10-23	Rain	light rain
Baltimore	Maryland	2021-10-23	Rain	light rain
Bellevue	Washington	2021-10-23	Rain	light rain
Boise	Idaho	2021-10-23	Rain	light rain
Boston	Massachusetts	2021-10-23	Rain	light rain
Bridgeport	Connecticut	2021-10-23	Rain	light rain