Chapter 46.

October 2020

Welcome to the October 2020 edition of DataStax Developer’s Notebook (DDN). This month we answer the following question(s);

> I love the GraphQL, Python/Flask, OpenStreetView, geo-spatial discussion this series has had of late. I’m having trouble putting it all together. Any chance you can put it all in one deliverable. Can you help ?
>
> *Excellent question ! In this article, we assemble all of the pieces we’ve recently discussed, putting them all in one coordinated deliverable. We’ll detail the data format, start up scripts, the program proper, and even any HTML related to OpenStreetView. (Eg., not Google Maps.)*

Software versions

The primary DataStax software component used in this edition of DDN is DataStax Enterprise (DSE), currently release 6.8.5, or DataStax Astra (Apache Cassandra version 4.0.0.682), as required. All of the steps outlined below can be run on one laptop with 16 GB of RAM, or if you prefer, run these steps on Amazon Web Services (AWS), Microsoft Azure, or similar, to allow yourself a bit more resource.

For isolation and (simplicity), we develop and test all systems inside virtual machines using a hypervisor (Oracle Virtual Box, VMWare Fusion version 8.5, or similar). The guest operating system we use is Ubuntu Desktop version 18.04, 64 bit.

46.1 Terms and core concepts

As stated above, ultimately the end goal is to (tie it all together); a Web form, DataStax Astra, GraphQL, geo-spatial, mapping UI, other. We’ve dived into most of these topics over the past few issues in this document series. Here, we’ll do a code review of all of the relevant pieces.

Example 46-1lists the database schema used in these examples. A code review follows.

Example 46-1 Our Schema File (our DDL file)

```




USE my_keyspace;


DROP TABLE IF EXISTS my_mapdata;

CREATE TABLE my_mapdata
   (
   md_pk                 TEXT PRIMARY KEY,
   md_lat                TEXT,
   md_lng                TEXT,
   geo_hash10            TEXT,
   md_name               TEXT,
   md_address            TEXT,
   md_city               TEXT,
   md_province           TEXT,
   md_postcode           TEXT,
   md_phone              TEXT,
   md_category           TEXT,
   md_subcategory        TEXT,
      //
   geo_hash5             TEXT,
   geo_hash6             TEXT,
   geo_hash7         TEXT,
   name3             TEXT,
   name5             TEXT,
   name7             TEXT
   );

CONSISTENCY LOCAL_QUORUM

COPY my_mapdata
   (
   md_pk             ,
   md_lat            ,
   md_lng            ,
   geo_hash10        ,
   md_name           ,
   md_address        ,
   md_city           ,
   md_province       ,
   md_postcode       ,
   md_phone          ,
   md_category       ,
   md_subcategory    ,
   geo_hash5         ,
   geo_hash6         ,
   geo_hash7         ,
   name3             ,
   name5             ,
   name7
   )
   FROM '26_mapData_CO.pipe'
   WITH HEADER = TRUE
   AND DELIMITER = '|'
   AND MAXBATCHSIZE = 20
   AND INGESTRATE = 100;
```

```

CREATE CUSTOM INDEX geo_hash5_idx
   ON my_mapdata
   (
   geo_hash5
   )
   USING 'StorageAttachedIndex'
   WITH OPTIONS = { 'case_sensitive': true, 'normalize': false };
      //
CREATE CUSTOM INDEX geo_hash6_idx
   ON my_mapdata
   (
   geo_hash6
   )
   USING 'StorageAttachedIndex'
   WITH OPTIONS = { 'case_sensitive': true, 'normalize': false };
      //
CREATE CUSTOM INDEX geo_hash7_idx
   ON my_mapdata
   (
   geo_hash7
   )
   USING 'StorageAttachedIndex'
   WITH OPTIONS = { 'case_sensitive': true, 'normalize': false };

CREATE CUSTOM INDEX name3_idx
   ON my_mapdata
   (
   name3
   )
   USING 'StorageAttachedIndex'
   WITH OPTIONS = { 'case_sensitive': false, 'normalize': true };
      //
CREATE CUSTOM INDEX name5_idx
   ON my_mapdata
   (
   name5
   )
   USING 'StorageAttachedIndex'
   WITH OPTIONS = { 'case_sensitive': false, 'normalize': true };
      //
CREATE CUSTOM INDEX name7_idx
   ON my_mapdata
   (
   name7
   )
   USING 'StorageAttachedIndex'
   WITH OPTIONS = { 'case_sensitive': false, 'normalize': true };




```

Relative to Example 46-1, the following is offered:

- In this and all of the examples that follow, disregard the line numbers that precede (each line).
- The single table in play is titled, “my_mapdata”.
- For ease of use, we use the CQLSL command shell command titled, “COPY”. If we are using the free tier of DataStax Astra, we limits the ingest rate to 100 rows per second.
- Several indexes, all TEXT;

We achieve the geo-spatial function of this Web application be encoding (storing) data that is geo-encoded, then performing indexed lookups on equalities.

Example 46-2 details how to run the DDL file from above. A code review follows.

Example 46-2 Running/installing our schema file.

```




echo ""
echo "247887 rows, 100/rows/sec, .. .. equals 45 minutes or so"
echo ""
echo ""

cqlsh -u my_user -p my_password -b secure-connect-my-database.zip -f 31*


```

Relative to Example 46-2, the following is offered:

- We use the CQLSH command interface to file our previously listed CQL DDL file.
- "-u" and "-p" are for user name and password.
- We also add use of the "secure connection bundle", which is downloadable from DataStax Astra.

Example 46-3 displays a stand alone, Python, DataStax Astra query client, using GraphQL. A code review follows.

Example 46-3 Stand alone, Python, GraphQL (test) program

```


#  pip install gql
```

```



###############################################################


import requests
import json
import time
   #
from gql import gql, Client
from gql.transport.requests import RequestsHTTPTransport

import libgeohash as gh

import urllib3
#  This line suppresses the Https warnings-
#
urllib3.disable_warnings(urllib3.exceptions.InsecureRequestWarning)



###############################################################


ASTRA_CLUSTER_ID     = "275553d2-XXXXXXXXXXXXXXXXXXX11378eac"
ASTRA_CLUSTER_REGION = "us-east1"
ASTRA_DB_USERNAME    = "my_user"
ASTRA_DB_PASSWORD    = "my_passwordXXX"

ASTRA_KEYSPACE       = "my_keyspace"
```

```
ASTRA_TABLE          = "my_mapdata"

ASTRA_MAXRETRIES     = 10



##############################################################


#  Get Auth Token

l_url1 = "https://" + ASTRA_CLUSTER_ID + "-" + ASTRA_CLUSTER_REGION +     \
   ".apps.astra.datastax.com/api/rest/v1/auth"
      #
l_data1 = '{"username":"' + ASTRA_DB_USERNAME + '", "password":"' + ASTRA_DB_PASSWORD + '"}'


for _ in range(ASTRA_MAXRETRIES):
   try:
      response = requests.post(l_url1, data=l_data1,headers={"Content-Type": "application/json"})
   except:
      #  Astra free tier, we get the occasional time outs
      #
      time.sleep(0.25)
      print "NOTICE: Bump 1"
      continue
   else:
      break
else:
   print ""
   print ""
   print "ERROR: Failed to connect with Astra instance"
   print ""
   print ""
   exit(3)

my_authToken = response.json()['authToken']



##############################################################


#  Query via GraphQL


url3 = "https://" + ASTRA_CLUSTER_ID + "-" + ASTRA_CLUSTER_REGION +     \
   ".apps.astra.datastax.com/api/graphql"

sample_transport=RequestsHTTPTransport(
    url=url3,
    use_json=True,
    headers={
        "Content-type": "application/json",
        "X-Cassandra-Token": my_authToken,
    },
    verify=False,
    retries=3,
)

client = Client(
    transport=sample_transport,
    fetch_schema_from_transport=True,
)
```

```


##############################################################
##############################################################


#  In my real program, this was a much larger/complex query
#

l_queryString = '''
   query {

   Q1 : myMapdata(value:  { name3: "sta" }, options: {limit: 1} )
      { values { mdName } }

   Q2 : myMapdata(value:  { name3: "nai" }, options: {limit: 1} )
      { values { mdName } }

   }
'''

l_queryString = gql(l_queryString)

for _ in range(ASTRA_MAXRETRIES):
   try:
      l_result = client.execute(l_queryString)
   except:
      #  Astra free tier, we get the occasional time outs
      #
      time.sleep(0.25)
      print "NOTICE: Bump 2"
```

```
125        continue
126     else:
127        break
128  else:
129     print ""
130     print ""
131     print "ERROR: Failed to connect with Astra instance"
132     print ""
133     print ""
134     exit(4)
135
136  print l_result
137
138
```

Relative to Example 46-3, the following is offered:

- This is a stand alone program; connecting to DataStax Astra, querying using GraphQL.
- This program serves geo-spatial data. We use the standard Python package titled, “libgeohash”, for encoding and decoding.
- This program queries DataStax Astra uses GraphQL. We use the “gql” Python library for this.
- Because DataStax Astra communicates uses Https, and we didn't wish to spend time to set this up, line 21 calls to disable the numerous Https/Http exception warnings we’d normally receive.
- Lines 41 through 68 get an “Authorization Token” from DataStax Astra, using Http REST. We are using the free tier of DataStax Astra, which presents the occasional service request time out error. As such, we loop, and execute the call to receive a token inside a try/except block.

  If we receive a timeout here, we print “Bump 1” to the terminal window.
- Lines 74 through 95 effectively declare variables used when querying. Nothing here can really fail; these are just variable sets/declarations.
- Lines 100 through 136 run our query proper. As before, we call DataStax Astra using a loop with a try/except block. If we time out, we print “Bump 2” to the terminal window.

– Lines 103 begins definition of our GraphQL query string.

Example 46-4 presents the entire Python Web application server program. A code review follows.

Example 46-4 Server program, thin Web client, Python, GraphQL

```
1
  2
  3  #  Single page Web application written in Python. Displays
geo-
  4  #  spatial and GraphQL using Astra/C*.
  5  #
  6  #    .  Web page will serve at localhost:8082
  7  #
  8  #       There are instructions on this Web page, How this
program
  9  #       functions.
 10  #
 11  #    .  Data comes from Astra/C*. the 'secure connect bundle'
 12  #       needs to be in the current working directory.
 13  #
 14  #       Also, a number of usernames, other, are hard coded
 15  #       into this file.
 16
 17
 18
############################################################
 19  ## Imports
#################################################
 20
 21
 22  #  Flask is our Python based Web server.
 23  #
 24  from flask import Flask, render_template, request, jsonify
 25
```

```
#  This import allows us to use a directory other than the
#  default for Flask CSS files and related.
#
import os

#  Geohash library
#
import libgeohash as gh

#  Ability to execute queries using GraphQL
#     pip install gql
#
import requests
import json
import urllib3
import time
   #
from gql import gql, Client
from gql.transport.requests import RequestsHTTPTransport

urllib3.disable_warnings(urllib3.exceptions.InsecureRequestWarning)


#############################################################
#############################################################


#  Constants used to connect with database server; Astra/C*
#

```

```
ASTRA_CLUSTER_ID     = "275553d2-46XXXXXXXXXXXXXXXXXXXXX8eac"
ASTRA_CLUSTER_REGION = "us-east1"
ASTRA_DB_USERNAME    = "my_user"
ASTRA_DB_PASSWORD    = "my_passwordXXXXX"

ASTRA_KEYSPACE       = "my_keyspace"
ASTRA_TABLE          = "my_mapdata"

ASTRA_MAXRETRIES     = 10



#############################################################
#############################################################


#  Get Authorization Token to be able to speak to Astra/C*
#

l_url1 = "https://" + ASTRA_CLUSTER_ID + "-" + ASTRA_CLUSTER_REGION + \
   ".apps.astra.datastax.com/api/rest/v1/auth"
      #
l_data1 = '{"username":"' + ASTRA_DB_USERNAME + '", \
   "password":"' + ASTRA_DB_PASSWORD + '"}'

for _ in range(ASTRA_MAXRETRIES):
   try:
      response = requests.post(l_url1, data=l_data1,headers=
         {"Content-Type": "application/json"})
   except:
      #  Astra free tier, we get the occasional time outs
```

```
        #
        time.sleep(0.25)
        print "NOTICE: Bump 1"
        continue
    else:
        break
else:
    print ""
    print ""
    print "ERROR (7442): Failed to connect with Astra instance."
    print ""
    print ""
    exit(3)

my_authToken = response.json()['authToken']
    #
print "INFO: Got Authorization Token, " + my_authToken



#############################################################
#############################################################


#  Connection handle, Query below via GraphQL
#

l_url3 = "https://" + ASTRA_CLUSTER_ID + "-" + ASTRA_CLUSTER_REGION + \
    ".apps.astra.datastax.com/api/graphql"

l_transport=RequestsHTTPTransport(
    url=l_url3,
    use_json=True,
    headers={
        "Content-type": "application/json",
        "X-Cassandra-Token": my_authToken,
    },
    verify=False,
    retries=3,
)

m_client = Client(
    transport=l_transport,
    fetch_schema_from_transport=True,
)



############################################################
## Inits, Opens, and Sets
##################################


#  Instantiate Flask object
#
m_app = Flask(__name__)


#  Set flask defaults for locating files
#
m_templateDir = os.path.abspath("45_views" )
m_staticDir   = os.path.abspath("44_static")
   #
m_app.template_folder = m_templateDir
m_app.static_folder   = m_staticDir
```

```



##############################################################
## Our Web pages (page handlers) ############################


#
#  This is our main page.
#
#  This ia a single page Web app; after this page loads,
#  everything else is just data/AJAX.
#
@m_app.route('/')
def do_servePage():
   return render_template("60_Index.html")


      ###############################################


#  This is our query response (page)
#

@m_app.route('/_do_query')
def do_query():

   l_lat        = request.args.get('h_lat'        )
   l_lng        = request.args.get('h_lng'        )
   l_textFilter = request.args.get('h_textFilter' )
      #
   l_latLng = gh.encode(float(l_lat), float(l_lng), precision=5)
```

```
      #
   print ""
   print "INFO: Query using (geohash5), " + l_latLng
   print ""

   l_markers = query_function(l_latLng, l_textFilter)
      #
   return jsonify(l_markers)


      ###############################################


#  sample output from gh.neighbors(),
#
#    {'e': '9xj3v', 'sw': '9xj3e', 'ne': '9xj6j', 'n': '9xj6h',
#       's': '9xj3s', 'w': '9xj3g', 'se': '9xj3t', 'nw': '9xj65'}
#

def query_function(i_latLng, i_textFilter):
   global m_client


   #  'C0' is our center point, where we query from
   #
   #  This is also the data set displayed when walking
   #
   l_loca_C0 = i_latLng

   #  'neighbors1' are the first set of points just past our
   #  center, the first ring, if you will
   #
```

```
#  This is also the data set displayed when driving slow
#
l_neighbors1 = gh.neighbors(l_loca_C0)

#  And anything (2) are our second ring of points, just
#  past our first ring. Generally displayed when driving
#  fast.
#
l_N2  = gh.neighbors(l_neighbors1['n' ])['n' ]
l_E2  = gh.neighbors(l_neighbors1['e' ])['e' ]
l_S2  = gh.neighbors(l_neighbors1['s' ])['s' ]
l_W2  = gh.neighbors(l_neighbors1['w' ])['w' ]
   #
l_NE2 = gh.neighbors(l_neighbors1['ne'])['ne']
l_SE2 = gh.neighbors(l_neighbors1['se'])['se']
l_SW2 = gh.neighbors(l_neighbors1['sw'])['sw']
l_NW2 = gh.neighbors(l_neighbors1['nw'])['nw']

#  Building our query string when a name is specified for
#  a business.
#
if (len(i_textFilter) >= 7):
   l_textFilter = ', name7: "' + i_textFilter[:7] + '"'
elif (len(i_textFilter) >= 5):
   l_textFilter = ', name5: "' + i_textFilter[:5] + '"'
elif (len(i_textFilter) >= 3):
   l_textFilter = ', name3: "' + i_textFilter[:3] + '"'
else:
   l_textFilter = ' '
      #
print len(l_textFilter)
print "INFO: Text Filter, " + l_textFilter

#  The column list we return from query
```

```
    #
    l_columnList = " mdLat mdLng mdName mdAddress mdCity mdProvince mdPhone mdSubcategory "


    #  Building the final query string; GraphQL query strings can get long
    #
    l_queryString = '''
      query {{

        C0 : myMapdata(value:  {{ geoHash5: "{0}"  {17} }} ) {{ values {{ {18} }} }}

        N1 : myMapdata(value:  {{ geoHash5: "{1}"  {17} }} ) {{ values {{ {18} }} }}
        E1 : myMapdata(value:  {{ geoHash5: "{2}"  {17} }} ) {{ values {{ {18} }} }}
        S1 : myMapdata(value:  {{ geoHash5: "{3}"  {17} }} ) {{ values {{ {18} }} }}
        W1 : myMapdata(value:  {{ geoHash5: "{4}"  {17} }} ) {{ values {{ {18} }} }}

        NE1 : myMapdata(value: {{ geoHash5: "{5}"  {17} }} ) {{ values {{ {18} }} }}
        SE1 : myMapdata(value: {{ geoHash5: "{6}"  {17} }} ) {{ values {{ {18} }} }}
        SW1 : myMapdata(value: {{ geoHash5: "{7}"  {17} }} ) {{ values {{ {18} }} }}
        NW1 : myMapdata(value: {{ geoHash5: "{8}"  {17} }} ) {{ values {{ {18} }} }}

        N2 : myMapdata(value:  {{ geoHash5: "{9}"  {17} }} ) {{ values {{ {18} }} }}
```

```
       E2 : myMapdata(value:  {{ geoHash5: "{10}" {17} }} ) {{ values {{ {18} }} }}
       S2 : myMapdata(value:  {{ geoHash5: "{11}" {17} }} ) {{ values {{ {18} }} }}
       W2 : myMapdata(value:  {{ geoHash5: "{12}" {17} }} ) {{ values {{ {18} }} }}

       NE2 : myMapdata(value: {{ geoHash5: "{13}" {17} }} ) {{ values {{ {18} }} }}
       SE2 : myMapdata(value: {{ geoHash5: "{14}" {17} }} ) {{ values {{ {18} }} }}
       SW2 : myMapdata(value: {{ geoHash5: "{15}" {17} }} ) {{ values {{ {18} }} }}
       NW2 : myMapdata(value: {{ geoHash5: "{16}" {17} }} ) {{ values {{ {18} }} }}

      }}
   '''
      #
   l_queryString = gql(l_queryString.format(l_loca_CO, l_neighbors1['n'],
      l_neighbors1['e'], l_neighbors1['s'], l_neighbors1['w'],
      l_neighbors1['ne'], l_neighbors1['se'], l_neighbors1['sw'],
      l_neighbors1['nw'], l_N2, l_E2, l_S2, l_W2, l_NE2, l_SE2,
      l_SW2, l_NW2, l_textFilter, l_columnList ))

   #  Retry fetch loop
   #
   for _ in range(ASTRA_MAXRETRIES):
      try:
         l_result = m_client.execute(l_queryString)
      except:
         #  Astra free tier, we get the occasional time outs
         #
         time.sleep(0.25)
         print "NOTICE: Bump 2"
         continue
      else:
         break
   else:
      print ""
      print ""
      print "ERROR (7443): Failed to connect with Astra instance."
      print ""
      print ""
      exit(3)

   return l_result


#############################################################
#############################################################


#
#  And then running our Web site proper.
#
if __name__=='__main__':

   m_app.run(host = "localhost", port = int("8082"), debug=True)

```

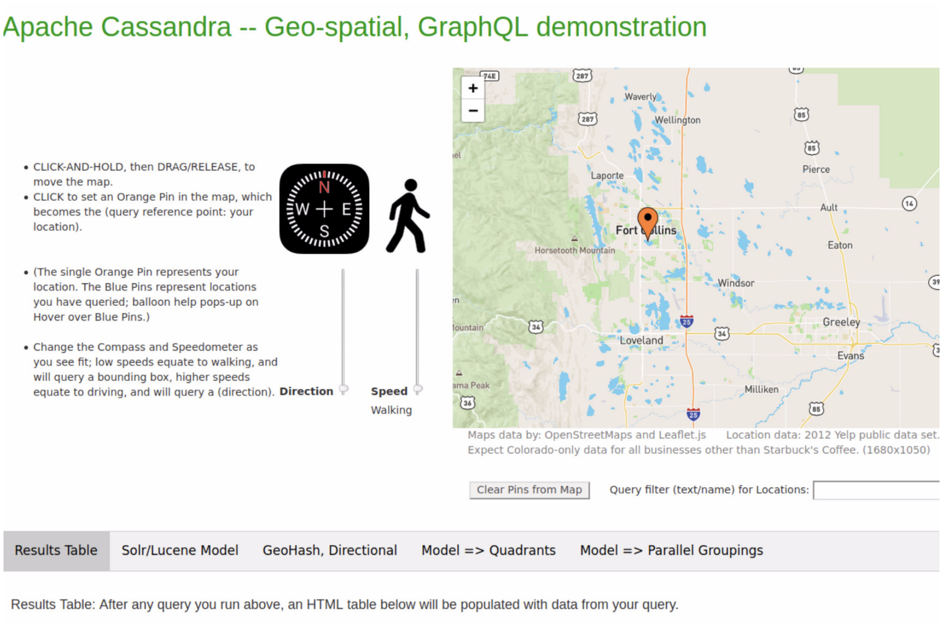
```
320
321
322
323
#############################################################
324
#############################################################
325
326
```

Relative to Example 46-4, the following is offered:

- The ‘guts’ of this program do not differ much from the (stand alone) program detailed above. Any new code is really to operate the single page Web application.
- Nothing prior to line 130 should present itself as new.
- Lines 136 through 149 set values specific to Python/Flask, our Web server; the location of HTML, CSS, and JavaScript files.
- Lines 160 through 162 serve our (index.html) page.
- Lines 168 through 308 form our query listener, and most of this is data.
  - Lines 174 through 176, we retrieve our query parameters.
  - h_textFilter is an optional (business name) to query, add as a query predicate.
  - query_function() acts as our DAO.
  - Sample output from the GraphQL query is listed on line 194, so you have a sense of what we'll send to the client; what we need to parse there.
  - So, we query a radius to a central point; our location on the map, which we label C0. And we query neighbors; the 8 points on a compass, which we label, E, W, N, and so on. And, we query neighbors to those 8 points, which we label, E2, W2, N2, and so on.
  - Line 232 and thereabouts is checking to see is we did in fact receive a business name to also query predicate on.
  - Line 251 and beyond build our GraphQL query string; effectively, querying compass points and their neighbors, is similar to running a 20 way SQL UNION query.

    - And line 287 begins out fetch loop; same as before.
- Line 316 launches our Web application listener.

Figure 46-1displays an image of the Web application we are detailing. After this image, we offer the HTML source listing. A code review follows.



Apache Cassandra -- Geo-spatial, GraphQL demonstration

- CLICK-AND-HOLD, then DRAG/RELEASE, to move the map.
- CLICK to set an Orange Pin in the map, which becomes the (query reference point: your location).

- (The single Orange Pin represents your location. The Blue Pins represent locations you have queried; balloon help pops-up on Hover over Blue Pins.)

- Change the Compass and Speedometer as you see fit; low speeds equate to walking, and will query a bounding box, higher speeds equate to driving, and will query a (direction).

Maps data by: OpenStreetMaps and Leaflet.js Location data: 2012 Yelp public data set. Expect Colorado-only data for all businesses other than Starbuck's Coffee. (1680x1050)

Query filter (text/name) for Locations:

Results Table: After any query you run above, an HTML table below will be populated with data from your query.

Figure 46-1 Image of the Web application

Example 46-5 lists the HTML to our Web application. A code review follows.

Example 46-5 HTML Listing for our Web application

```



<!--
-----------------------------------------------------------

-----------------------------------------------------------------



     DataStax Astra; Geo-hash. GraphQL demonstration program.

```

```

----------------------------------------------------------------
---- --------------------------------------------------------- -->


<!DOCTYPE html>
<html>


<!-- --------------------------------------------------------- -->
<!-- --------------------------------------------------------- -->
<!-- --------------------------------------------------------- -->
<!-- --------------------------------------------------------- -->


<head>

   <meta charset="utf-8" />
   <meta name="viewport" content="width=device-width, initial-scale=1.0">

   <!--
   This block required for jQuery, which gives us Ajax support.
   -->
   <script src="{{ url_for('static', filename='10_jquery.min.js'  ) }}">
   </script>
```

```
<link rel="stylesheet" type="text/css"
   href="{{ url_for('static', filename='11_bootstrap.min.css') }}">

<!--
This block required for the TABbed DIVs.
-->
<link rel="stylesheet" type="text/css"
   href="{{ url_for('static', filename='20_TABbedMenu.css') }}">
<script src="{{ url_for('static', filename='21_TABbedMenu.js') }}">
</script>

<!--
This block required for Leaflet, which give us our maps.
-->
<link rel="stylesheet" type="text/css"
   href="{{ url_for('static', filename='24_leaflet.css') }}">
<script src="{{ url_for('static', filename='25_leaflet.js') }}">
</script>

<!--
Used for the vertical sliders.
-->
<style>
input.vertical {
   -webkit-appearance: slider-vertical;
   writing-mode: bt-lr;
   }
</style>

```

```
</head>


<!-- -------------------------------------------------------- -->
<!-- -------------------------------------------------------- -->
<!-- -------------------------------------------------------- -->
<!-- -------------------------------------------------------- -->


<body>

   <br>
   <h1>
      <span style="color:#009900">
      Apache Cassandra -- Geo-spatial, GraphQL demonstration
      </span>
   </h1>
   <br>

   <table border=0>
      <tr>
         <td style="width:600px">
            <table border=0>

               <td>
                  <!-- ---------------------------------------------------------
                  Instructions; How to use this form.

                  ---------------------------------------------------------- -->
                  <ul>
                     <li>
                     CLICK-AND-HOLD, then DRAG/RELEASE, to move the map.
                     </li>
                     <li>
                     CLICK to set an Orange Pin in the map, which becomes the
                     (query reference point: your location).
                     </li>
                     <br>
                     <br>
                     <li>
                     (The single Orange Pin represents your location. The Blue
                     Pins represent locations you have queried; balloon help
                     pops-up on Hover over Blue Pins.)
                     </li>
                     <br>
                     <li>
                     Change the Compass and Speedometer as you see fit; low
                    speeds equate to walking, and will query a bounding box,
                   higher speeds equate to driving, and will query a (direction).
                     </li>
                  </ul>
               </td>

               <td>
                  <!--
   --------------------------------------------------------
```

```
      Visual control for direction

--------------------------------------------------------- -->
      <img id="img_compass" src="{{ url_for('static', filename='./images/compass_0.png') }}"
         width="120" height="120">
      <br>
      <br>
      <label for="slider_compass">Direction</label>
      <input type="range" min="0" max="360" value="0" step="45" id="slider_compass"
         class="vertical" orient="vertical" oninput="f_updateCompass(value)"
         list="slider_compass_settings" onchange="f_onChange1()">
      <datalist id="slider_compass_settings">
         <option>0</option>
         <option>45</option>
         <option>90</option>
         <option>135</option>
         <option>180</option>
         <option>225</option>
         <option>270</option>
         <option>315</option>
         <option>360</option>
      </datalist>

      <script>
         function f_updateCompass(heading) {
            if (heading == 0 || heading ==360) {
               document.querySelector('#img_compass').src =
                  "{{ url_for('static',
filename='./images/compass_0.png') }}";
                             } else if (heading == 45) {
document.querySelector('#img_compass').src =
                                   "{{ url_for('static',
filename='./images/compass_45.png') }}";
                             } else if (heading == 90) {
document.querySelector('#img_compass').src =
                                   "{{ url_for('static',
filename='./images/compass_90.png') }}";
                             } else if (heading == 135) {
document.querySelector('#img_compass').src =
                                   "{{ url_for('static',
filename='./images/compass_135.png') }}";
                             } else if (heading == 180) {
document.querySelector('#img_compass').src =
                                   "{{ url_for('static',
filename='./images/compass_180.png') }}";
                             } else if (heading == 225) {
document.querySelector('#img_compass').src =
                                   "{{ url_for('static',
filename='./images/compass_225.png') }}";
                             } else if (heading == 270) {
document.querySelector('#img_compass').src =
                                   "{{ url_for('static',
filename='./images/compass_270.png') }}";
                             } else {
document.querySelector('#img_compass').src =
                                   "{{ url_for('static', filename='./images/compass_315.png') }}";
                                }
                             }
                       </script>
                    </td>

                    <td>
                       <!-- --------------------------------------------------------
                       Visual control for speed

---------------------------------------------------------- -->
                       <br>
                       <br>
                       <img id="img_speed" src="{{ url_for('static', filename=
                          './images/person.png') }}" width="100" height="100">
                       <br>
                       <br>
                       <label for="slider_speed">Speed</label>
                       <input type="range" min="0" max="80" value="0" id="slider_speed"
                        step="10" oninput="f_updateSpeed(value)" onchange="f_onChange1()"
                          class="vertical" orient="vertical" list="slider_speed_settings">
                       <output for="slider_speed" id="output_speed_gauge">Walking</output>
                       <datalist id="slider_speed_settings">
                          <option>0</option>
                          <option>10</option>
                          <option>20</option>
```

```
   <option>30</option>
   <option>40</option>
   <option>50</option>
   <option>60</option>
   <option>70</option>
   <option>80</option>
</datalist>

<script>
   function f_updateSpeed(speed) {
      if (speed < 10) {
            document.querySelector('#output_speed_gauge').value = "Walking";
            //
            document.querySelector('#img_speed').src =
            "{{ url_for('static', filename='./images/person.png') }}";
      } else if (speed < 40) {
            document.querySelector('#output_speed_gauge').value = "Driving Slow - " + speed;
            //
            document.querySelector('#img_speed').src =
            "{{ url_for('static', filename='./images/jeep.png') }}";
      } else {
            document.querySelector('#output_speed_gauge').value = "Driving Fast - " + speed;
            //
            document.querySelector('#img_speed').src =
                             "{{ url_for('static', filename='./images/jeep.png') }}";
                          }
                       }
                 </script>
              </td>

           </table>
        </td>
        <td>
           <!-- --------------------------------------------------------
           div for the map proper
           ---------------------------------------------------------- -->
           <div id="div_map" style="width: 720px; height: 480px;"></div>
           </div>
           <p style="color:gray">
           Maps data by: OpenStreetMaps and Leaflet.js
           &nbsp&nbsp&nbsp&nbsp
           Location data: 2012 Yelp public data set.
           <br>
           Expect Colorado-only data for all businesses other than Starbuck's Coffee. (1680x1050)
           </p>
          <br>
           &nbsp&nbsp&nbsp&nbsp
           <input onclick="f_clearDataPins();" type=button value="Clear Pins from Map">
           &nbsp&nbsp&nbsp&nbsp
           Query filter (text/name) for Locations:
           <input type="text" id="it_textFilter" size="32"
              onchange="f_onChange2()">
```

```
            </td>
        </tr>
    </table>


    <!--------------------------------------------------------
    Script that runs the map
   ---------------------------------------------------------->

    <script>

        var l_mymap = L.map('div_map').setView([40.5259, -104.9263], 10);

        //  l_pinsRefArr[] keeps the blue pins we return and render from
       //  queries; effecitvely, the (stores) you are looking for.
        //
      //  l_locaPin is our reference point/location; where we are standing
        //  or where our car currently currently sits.
        //
        //  l_response is whatever answer we got from the server.
        //
        var l_pinsRefArr = [];
        var l_locaPin    = null;
        var l_response;

            // ////////////////////////////////////////////

```

```
 L.tileLayer('https://api.mapbox.com/styles/v1/{id}/tiles/{z}/{x}/{y}?access_token={access_token}', {
    maxZoom: 18,
    id: 'mapbox/streets-v11',
    access_token: 'pk.eyJ1IjoibWFwYm94IiwXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXJcFIG214AriISLbB6B5aw',
    tileSize: 512,
    zoomOffset: -1
    }).addTo(l_mymap);

 var PinIcon = L.Icon.extend({
    options: {
       iconAnchor:   [0, 0],
       iconSize:     [30, 50]
       }
    });
       //
 var bluePin  = new PinIcon({iconUrl: "{{ url_for('static', filename= './images/blue_pin.png') }}" }),
    orangePin = new PinIcon({iconUrl: "{{ url_for('static', filename= './images/orange_pin.png') }}" });

 //  Passing an array, obviously.
 //
//  This invocation sets the 'current location' pin, as we start the program.
 //
 f_setLocaPin([40.585258, -105.084419]);

    // ///////////////////////////////////////////

 //  'e' is an event object, with properties for
 //     e.latlng,       of type LatLng
 //     e.latlng.lat    and
 //     e.latlng.lng    of type float
 //
//    marker() below will overload, but we always send an array for debugging
 //
 l_mymap.on('click', function(e){
    f_setLocaPin([e.latlng.lat, e.latlng.lng]);
    f_runQuery(e.latlng.lat, e.latlng.lng,
       document.querySelector('#it_textFilter').value);
    });
 //  Sets our 'current location' pin
 //
 function f_setLocaPin(e) {
    if (l_locaPin !== null) {
       l_locaPin.remove();
       }
    var txt = ("<b>This is your current location.</b><br>" +
       "CLICK anywhere else to change your current location.");
    l_locaPin =  L.marker(e, {icon: orangePin})
       .bindPopup(txt)
       .addTo(l_mymap);
    }

    // //////////////////////////////////////////

//  Sets all other pins, effectively; our data pins for the businesses
 //  we return from query
 //
 function f_setDataPin(e, txt) {
```

```
           // var strx = e.latlng;
          // var l_pin = L.marker([strx.lat, strx.lng], {icon:
bluePin}).addTo(l_mymap);
           var l_pin = L.marker(e, {icon:
bluePin}).addTo(l_mymap);
              //
           var l_pin_popup;
              //
           l_pin.on('mouseover', function(e) {
              l_pin_popup = L.popup({ offset: L.point(0,0)});
                 l_pin_popup.setContent(txt);
                 l_pin_popup.setLatLng(e.target.getLatLng());
                 l_pin_popup.openOn(l_mymap);
                 });
           l_pin.on('mouseout', function(e) {
              l_mymap.closePopup(l_pin_popup);
              });
                 //
           l_pinsRefArr.push(l_pin);
           }

        //  Erase all of the data pins from the map
        //
        function f_clearDataPins() {
           for (var i = 0; i < l_pinsRefArr.length; i++) {
              l_pinsRefArr[i].remove();
              }
           l_pinsRefArr  = [];
           }

           // ///////////////////////////////////////////

        //  AJAX function, calls to server to get businesses
close to our
```

```
//  current location
//
function f_runQuery(i_lat, i_lng, i_textFilter) {
   $.getJSON(
      "/_do_query",
      {
      h_lat        : i_lat        ,
      h_lng        : i_lng        ,
      h_textFilter : i_textFilter
      },
      function(r_response) {
         // To just see what is returned,
         ///
         // var l_txt1 = JSON.stringify(r_response);
         // alert(l_txt1);

         l_response = r_response;
            //
         f_onChange1();
         }
      );
   };

   // ///////////////////////////////////////////////

//  Because we change what is displayed based on multiple events,
//  put this in a separate function.
//
function f_onChange1() {

   f_clearDataPins();

   l_currentSpeed   =
   document.querySelector('#slider_speed'  ).value
   l_currentCompass = document.querySelector('#slider_compass').value
      //
   if (l_currentSpeed < 10) {
      l_idx1 = "C0";
   } else if (l_currentSpeed < 40) {
     if (l_currentCompass == 0 || l_currentCompass == 360) {
         l_idx1 = "N1";
      } else if (l_currentCompass == 45 ) {
         l_idx1 = "NE1";
      } else if (l_currentCompass == 90 ) {
         l_idx1 = "E1";
      } else if (l_currentCompass == 135) {
         l_idx1 = "SE1";
      } else if (l_currentCompass == 180) {
         l_idx1 = "S1";
      } else if (l_currentCompass == 225) {
         l_idx1 = "SW1";
      } else if (l_currentCompass == 270) {
         l_idx1 = "W1";
      } else {
         l_idx1 = "NW1";
      }
   } else {
     if (l_currentCompass == 0 || l_currentCompass == 360) {
         l_idx1 = "N2";
      } else if (l_currentCompass == 45 ) {
         l_idx1 = "NE2";
      } else if (l_currentCompass == 90 ) {
         l_idx1 = "E2";
      } else if (l_currentCompass == 135) {
```

```
               l_idx1 = "SE2";
            } else if (l_currentCompass == 180) {
               l_idx1 = "S2";
            } else if (l_currentCompass == 225) {
               l_idx1 = "SW2";
            } else if (l_currentCompass == 270) {
               l_idx1 = "W2";
            } else {
               l_idx1 = "NW2";
            }
         }

         //  Format what we send to the HTML table builder differently
         //
         var l_sendToHtmlTable = []

        //  Parse thru our query results, build balloon help text
         //
        for (i = 0; i < l_response[l_idx1]["values"].length; i++) {
            l_latLng = [ parseFloat(l_response[l_idx1]["values"][i]["mdLat"]),
parseFloat(l_response[l_idx1]["values"][i]["mdLng"]) ];
                  //
            l_name     = l_response[l_idx1]["values"][i]["mdName"       ];
            l_subCat   = l_response[l_idx1]["values"][i]["mdSubcategory"];
            l_addr     = l_response[l_idx1]["values"][i]["mdAddress"    ];
            l_city     =
            l_response[l_idx1]["values"][i]["mdCity"      ];
            l_province = l_response[l_idx1]["values"][i]["mdProvince"   ];
            l_phone    = l_response[l_idx1]["values"][i]["mdPhone"      ];
               //
            l_ballTxt  = "<b>" + l_name + "</b><br>" +
               l_subCat + "<br>" + l_addr + ", " + l_city + ", " +
               l_province + "<br><br>" + l_phone;
                  //
            f_setDataPin(l_latLng, l_ballTxt);
                  //
            l_sendToHtmlTable.push( {h_name: l_name, h_latLng: l_latLng, h_ballTxt: l_ballTxt} );
            }

         //  Call to render HTML table
         //
         f_buildHtmlTable(l_sendToHtmlTable);

         };

         // ////////////////////////////////////////////

         //  Called only from the text entry field (filter)
         //
         function f_onChange2() {

            f_runQuery(l_locaPin.getLatLng().lat, l_locaPin.getLatLng().lng,
               document.querySelector('#it_textFilter').value);

```

```
          }

          // ///////////////////////////////////////////

    </script>


    <!-- --------------------------------------------------------
    Start the TABbed divs.
   ---------------------------------------------------------- -->

    <br>
    <br>

    <ul class="tab">
       <li><a href="javascript:void(0)" class="tablinks"
          onclick="openDiv(event, 'div_1')"
         id="li_tab1"     >Results Table            </a></li>
       <li><a href="javascript:void(0)" class="tablinks"
          onclick="openDiv(event, 'div_2' )"
         id="li_tab2"    >Solr/Lucene Model         </a></li>
       <li><a href="javascript:void(0)" class="tablinks"
          onclick="openDiv(event, 'div_3' )"
         id="li_tab3"    >GeoHash, Directional      </a></li>
       <li><a href="javascript:void(0)" class="tablinks"
          onclick="openDiv(event, 'div_4' )"
         id="li_tab4"    >Model => Quadrants        </a></li>
       <li><a href="javascript:void(0)" class="tablinks"
          onclick="openDiv(event, 'div_5' )"
          id="li_tab5"    >Model  => Parallel Groupings</a></li>
    </ul>
```

```


<!-- -- DIV 1 ---------------------------------------- -->


<div id="div_1" class="tabcontent">
   <br>
   <h4>
      Results Table: After any query you run above, an HTML table
      below will be populated with data from your query.
   </h4>
   <br>
   <br>
   <table id="t_queryData" class="tab_table" cellspacing="0"
      width="80%" align="center">
   </table>
   <br>
   <br>
   <br>
   <br>
   <br>
   <br>
   <br>
   <br>
</div>

<script>

   function f_buildHtmlTable(i_tableData) {
         //

document.getElementById("t_queryData").deleteTHead() ;
              document.getElementById("t_queryData").innerHTML =
"";
                 //
             var l_tabl = document.getElementById("t_queryData");
              var l_head = l_tabl.createTHead();
              var l_hrow = l_head.insertRow(0);
                 //
              var l_cell = l_hrow.insertCell(0);
              l_cell.innerHTML = "<b>" + "Name"         + "</b>";
              var l_cell = l_hrow.insertCell(1);
              l_cell.innerHTML = "<b>" + "Location"     + "</b>";
              var l_cell = l_hrow.insertCell(2);
              l_cell.innerHTML = "<b>" + "Description"  + "</b>";
                 //
              for (i = 0; i < i_tableData.length; i++) {
                    //
                 var l_brow = l_tabl.insertRow(i+1);
                 var l_data = i_tableData[i];
                    //
                 var l_cell = l_brow.insertCell(0);
                 var l_col1 = l_data["h_name"];
                 l_cell.innerHTML = l_col1;
                    //
                 var l_cell = l_brow.insertCell(1);
                 var l_col1 = l_data["h_latLng"];
                 l_cell.innerHTML = l_col1;
                    //
                 var l_cell = l_brow.insertCell(2);
                 var l_col2 = l_data["h_ballTxt"];
                 l_cell.innerHTML = l_col2;
                 }
              };

```

```
</script>


<!-- -- DIV 2 ----------------------------------------- -->


<div id="div_2" class="tabcontent">
   <br>
   <br>
   <img src="{{ url_for('static', filename='./images/geohash0.png') }}"
      width="1093" height="498">
   <br>
   <br>
   <br>
   <br>
</div>


<!-- -- DIV 3 ----------------------------------------- -->


<div id="div_3" class="tabcontent">
   <br>
   <br>
   <img src="{{ url_for('static', filename='./images/geohash3.png') }}"
      width="1093" height="498">
   <br>
   <br>
   <br>
   <br>
```

```
</div>


<!-- -- DIV 4 ----------------------------------------- -->


<div id="div_4" class="tabcontent">
   <br>
   <br>
   <img src="{{ url_for('static', filename='./images/geohash1.png') }}"
      width="1093" height="498">
   <br>
   <br>
   <br>
   <br>
</div>


<!-- -- DIV 5 ----------------------------------------- -->


<div id="div_5" class="tabcontent">
   <br>
   <br>
   <img src="{{ url_for('static', filename='./images/geohash2.png') }}"
      width="1093" height="498">
   <br>
   <br>
   <br>
   <br>
   </div>


   <!-- -------------------------------------------------- -->


   <!--
   This code is for the TABbed divs; makes TAB 1 appear on page load.
   -->
   <script type="text/javascript">
      document.getElementById("li_tab1" ).click();
   </script>


   <!-- -------------------------------------------------- -->


</body>


</html>




```

Relative to Example 46-5, the following is offered:

- We use 4JavaScript libraries;
  - The first is for JQuery, and support for AJAX; asynchronous (query) calls to the server program.

  - The second is to support TABbed divs, shown on the bottom of our screen.
  - The third is for our mapping library; LeafLet.js, which uses the OpenStreetMap view library.
  - And lastly, a library for the vertical sliders we use for compass heading and speed.
- Lines 72 through 116 are end user instructions, printed on the HTML form.
- Line 117 begins the code for our compass; which direction are you walking or driving.
- Line 174 begins the code for our speed; are you walking, driving slow, or driving fast.

  We use these 2 visual controls to impact our queries; do we show points close to ourself (we are standing and hungry), or do we show points near to ourselves and on a vector (we are driving North, slowly), or do we show points on our vector but much farther away (we are driving fast, give me ‘think’ time to choose a destination).
- Line 225 begins the HTML for our map.
- Line 252 starts the map code proper; JavaScript, calling to have a map, first, then setting and clearing pins later.
- Line 358 calls to get data from the server program, from the Cassandra database. Most of this code is reading the HTMP compass and speed indicator, and sending the correct/expected query predicate values.
- Line 480 starts our TABbed divs, which are mostly informational to the end user.

46.2 Complete the following

At this point in this document we have done a code review of all assets.

Experiment with running either the stand alone Python/GraphQL client, or running the Web application proper.

46.3 In this document, we reviewed or created:

This month and in this document we detailed the following:

- How to run GraphQL queries against DataStax Astra, using Python.
- How to use LeafLet.js, and the OpenStreetMaps, open source mapping library.

Persons who help this month.

Kiyu Gabriel, Dave Bechberger, and Jim Hatcher.

Additional resources:

Free DataStax Enterprise training courses,

`https://academy.datastax.com/courses/`

Take any class, any time, for free. If you complete every class on DataStax Academy, you will actually have achieved a pretty good mastery of DataStax Enterprise, Apache Spark, Apache Solr, Apache TinkerPop, and even some programming.

This document is located here,

`https://github.com/farrell0/DataStax-Developers-Notebook`

`https://tinyurl.com/ddn3000`