Chapter 23.

November 2018

Welcome to the November 2018 edition of DataStax Developer's Notebook (DDN). This month we answer the following question(s);

> My application needs to store a dynamic number of latitude/longitude pairs per single database row, along with a tag for these values like; home, work, mobile, etcetera. We need to perform distance (proximity) queries for any of the latitude/longitude pair values, as well as queries on specific tags; just home, just work, other. Can you help ?
>
> *Excellent question ! All of these application requirements are easily served with DataStax Enterprise Server (DSE). While we've covered DSE Search queries including spatial/geo-spatial in past editions of this document, the specific requirement you have for (a dynamic count of attributes), is something we have not covered in this series of documents previously.*
>
> *Using this same technique, DataStax Enterprise can also serve a polymorphic schema ability, similar to MongoDB.*
>
> *This edition of this document will address this application requirement with no prerequisites, although; you might well be served to visit the past editions of this document detailing DSE Search and DSE (spatial/geo-spatial Search), to gain a deep understanding.*

Software versions

The primary DataStax software component used in this edition of DDN is DataStax Enterprise (DSE), currently release 6.7. 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 version 18.0, 64 bit.

23.1 Terms and core concepts

As stated above, ultimately the end goal is to geo-spatial query a dynamic number of latitude/longitude (lat/lon) pairs per single database record. A couple of points to recall:

- DataStax Enterprise (DSE) has four primary functional areas; DSE Core, DSE Search, DSE Analytics, and DSE Graph.
- While you can perform a sequential scan of a table using DSE Search or DSE Analytics, generally you should expect that every query predicate must be served via in index.

  Huh ? Every column referred to in the WHERE clause of a SELECT statement should be supported via an index.
- DSE Core acts as the storage tier to the three other functional areas of DSE, and provides, effectively, hash indexes. DSE Search provides indexes other than hash, and query processing for text analytics, spatial/geo-spatial, times series, and more.
- Any dynamic list of data within a single row with DSE is served via a collection. DSE has three collection types; set, list, and map. Set and list are effectively scalar arrays; an array of just integers, just strings, other. Set and list differ in the manner that they allows duplicate values or not, maintain sort order, or not.

  Maps support a key/value pair. Using user defined types (UDTs), the key or the value (or both), may be multi-columned.

For the solution to the problem statement listed above, we will use a set of just lat/lon pairs. This column will support the query; does this row have a given valued lat/lon pair, regardless of any tag (work, mobile, other) associated with any single lat/lon pair. Then, we will use a map of a tag (work, mobile, other), and a lat/lon pair, to support the query; does this row have a (work, mobile, other) record for a given lat/lon pair.

Indexing will be a DSE Search index using RPT, the DSE Search (Apache Solr) SpatialRecursivePrefixTreeFieldType column type. Proximity queries will be served using the geofilt query parser.

Note: All of the above topics are detailed in the April/2018 edition of this same document, which was dedicated to DSE Search spatial/geo-spatial.

23.2 Complete the following

At this point in this document we are ready to start coding. A couple of disclaimers:

- Before putting any of these techniques into production, run a production sized test. The code we display should work for you, but you do need to size your DSE server correctly when running production sized loads.

  DSE Search indexes use Apache Solr under the covers, and Solr can consume a bit of resource quickly and without advance notice.

- Using the techniques in the code below, you might be tempted to use more schemaless tables and columns.

  Huh ? You can index and store columns not defined in the DSE table proper. MongoDB has this capability (polymorphic schemas), and MongoDB is a good general purpose database.

  DataStax Enterprise (DSE) is less concerned with being a general purpose database, and instead DSE aims to be a Web scale database. To achieve Web scale, you should plan to avoid wide use of capabilities like schemaless, scatter gather queries, etcetera.

Example 23-1lists the first block of code necessary to implement the application requirements we are detailing. A code review follows.

Example 23-1 Create table and related.

```
DROP KEYSPACE ks_23;

CREATE KEYSPACE ks_23
   WITH REPLICATION = {
      'class' : 'SimpleStrategy',
      'replication_factor' : 1
      };

USE ks_23;

CREATE TABLE t1
   (
   col1            INT PRIMARY KEY,
   col2            TEXT,
   m1_             MAP<TEXT, 'PointType'>,
   col4            TEXT,
   m2_             MAP<TEXT, TEXT>,
   col6            TEXT,
   col7            SET<'PointType'>,
   col8            TEXT
   );
```



```
//
//  Do not make the initial index including m1_
//
//  The index you want on m1_ does not auto-generate
//  until version 6.7 of DSE.
//
CREATE SEARCH INDEX ON ks_23.t1
   WITH COLUMNS col1, * { excluded : true };

ALTER SEARCH INDEX SCHEMA ON t1
   ADD types.fieldtype[@class='solr.SpatialRecursivePrefixTreeFieldType',
   @name='my_geo_point', @distanceUnits='kilometers'];

ALTER SEARCH INDEX SCHEMA ON t1
   ADD fields.dynamicField[@name='m1_*',
   @type='my_geo_point', @indexed='true'];

RELOAD  SEARCH INDEX ON ks_23.t1;
REBUILD SEARCH INDEX ON ks_23.t1;

DESCRIBE PENDING SEARCH INDEX SCHEMA ON t1 ;
DESCRIBE ACTIVE  SEARCH INDEX SCHEMA ON t1 ;
```

Relative to Example 23-1, the following is offered:

- We execute a standard CREATE KEYSPACE, and USE (keyspace).
- The CREATE TABLE has (n) columns of interest:
  - m1_ is defined of type map.

    Maps are by definition, key/value pairs. The key or the value or both could each be a single column (as displayed), or multi-columned. Multi-column is achieved using a user defined type (UDT). Our use case today does not require multi-column, so none are displayed.

    The key to our map is of type TEXT, and is expected to keep values similar to; home, work, mobile, other.

    The value to our map is of type, PointType. DSE Search will see PointType and create a DSE Search field type of, SpatialRecursivePrefixTreeFieldType, commonly referred to as, RPT.

    RPT columns generally store a latitude/longitude pair, and support geo-spatial queries like distance from point, other.
  - m2_ is used much later in this document, to support a use case not directly related to spatial/geo-spatial.

    - col7 is not a map, and is instead of type, set. Sets are by definition, a list (array) of a single type of column; integer, text, or as in use in this case, RPT.
  - The CREATE SEARCH INDEX statement calls to make the primary key of this table known to DSE Search.

Note: With version 6.7 of DSE, we could auto-generate more of the code and conditions we explicitly create below.

Huh ? We could write less code in this example using version 6.7.

The example we present is more verbose, and will run on version 6.0.4 of DSE.

  - The two ALTER SEARCH INDEX statements create a field type, and then field, the standard pattern when defining DSE Search indexes. Comments:
    - The field type is of type, RPT, with units kilometers. (The default unit of measure is degrees, which helps no one.)
    - The field is defined to be a, dynamicField. The two use cases within DSE for dynamicField(s) are older version DSE geo-spatial (not our case today), and maps (a DSE collection type).

      In effect, any key value placed in this table, in the column titled "m1_", and prefixed with a "m1_" key value will be indexed in this manner; an RPT field type.

Note: We see more on this topic below, when we detail, col7.

  - A standard RELOAD, REBUILD, calling to deploy our DSE Search index, and then diagnostic commands in the form of DESCRIBE.

Example 23-2 displays the next piece of our example. A code review follows.

Example 23-2 Inserting data

```
//
//                       Lat      Long
//                       =======  ========
//  Denver, CO           39.7392  104.9903
//  Colorado Springs, CO 38.8339  104.8214
//  Madison, WI          43.0731   89.4012
//


//
```



```
//  Strangely, values input at Long/Lat
//
INSERT INTO t1 (col1, m1_)
   VALUES (1, { 'm1_home' : 'POINT(104.9903 39.7392)', 'm1_work' :
'POINT(104.8214 38.8339)' } );
INSERT INTO t1 (col1, m1_)
  VALUES (2, { 'm1_home' : 'POINT(104.9903 39.7392)'
} );
INSERT INTO t1 (col1, m1_)
  VALUES (3, { 'm1_auto' : 'POINT(89.4012 43.0731)'
} );
INSERT INTO t1 (col1, m1_)
   VALUES (4, { 'm1_auto' : 'POINT(104.9903 39.7392)', 'm1_home' :
'POINT(104.8214 38.8339)' } );
INSERT INTO t1 (col1, m1_)
  VALUES (5, { 'm1_home' : 'POINT(89.4012 43.0731)'
} );
```

Relative to Example 23-2, the following is offered:

- Four insert statements, the primary key, and then column, m1_.
- We are using lat/lon data for Denver, CO, Colorado Springs, CO (COS), and Madison, WI. Denver and COS are about 50-70 miles apart.
- Some of the data is tagged; home, work, auto, other. If the tag value is prefixed, "m1_", then the value pair will be indexed, otherwise, no.

Example 23-3 displays a number of sample (test) queries. A code review follows.

Example 23-3 SAmple/test queries.

```
//
//  All rows
//
SELECT *
FROM t1
WHERE solr_query = '{ "q" : "*:*"}';

//
//  Has an attribute 'auto'  (indexed as 'm1_auto')
//
SELECT *
FROM t1
WHERE solr_query = '{ "q" : "m1_auto:*"}';

//
//  Rectangle: Long then Lat, left side always less than right side
```



```
//
SELECT *
FROM t1 where solr_query =
   'm1_home:["104.00 39.00" TO "105.00 40.00"]';

//
//  Syntax here is Lat then Long (sorry; open source, older library)
//
SELECT *
FROM t1
WHERE solr_query = '{ "q" : "m1_auto:[39.00,104.00 TO 40.00,105.00]"}';


//
//  Distance from point (circle, units KM)
//
SELECT *
FROM t1
WHERE solr_query =
   '{ "q" : "{!geofilt pt=39.7300,104.9900 sfield=m1_home d=10    }" }';

SELECT *
FROM t1
WHERE solr_query =
   '{ "q" : "{!geofilt pt=39.7300,104.9900 sfield=m1_home d=200   }" }';

SELECT *
FROM t1
WHERE solr_query =
   '{ "q" : "{!geofilt pt=39.7300,104.9900 sfield=m1_home d=4000 }" }';


//
//  Bounding box
//
SELECT *
FROM t1
WHERE solr_query =
   '{ "q" : "{!bbox pt=39.7300,104.9900 sfield=m1_home d=10    }" }';
```

Relative to Example 23-3, the following is offered:

- The first SELECT returns all rows.
- The second SELECT returns all rows with an "auto" ("m1_auto") tag, regardless of value.

- The third SELECT returns all rows with a tag equal to "home" ("m1_home") in the rectangle defined by the values shown.

  Using this query syntax, values are entered first longitude, then latitude, and smaller value ranging to larger value.

  From our sample data, rows 1 and 2 are returned.
- The fourth SELECT returns just row 4, and is meant to show just an alternate form of the rectangle syntax. In this form, data is entered latitude then longitude.
- Queries five through seven all query distance from a point in kilometers.
- And query eight perform a bounding box query. (Similar to point from a circle, but using a box; boxes requiring less geometry and thus executing faster and with less resource.)

Apache Solr, Dynamic Schemas

The example above works using the Apache Solr feature generally titled, dynamic schemas. We don't have to use a geo-spatial valued field when using dynamic schemas.

Example 23-4 displays this use case. A code review follows.

Example 23-4 Apache Solr dynamic fields in general

```
//
//  Odd edge case; we need this next line because we never
//  indexed a 'StrField' in this table before; that rarely
//  happens.
//
ALTER SEARCH INDEX SCHEMA ON t1
   SET types.fieldtype[@class='org.apache.solr.schema.StrField']
   @name='StrField';

ALTER SEARCH INDEX SCHEMA ON t1
   ADD fields.dynamicField[@name='m2_*',
   @type='StrField', @indexed='true'];

RELOAD  SEARCH INDEX ON ks_23.t1;
REBUILD SEARCH INDEX ON ks_23.t1;


//
//  'political party' is stored, but not indexed
//
INSERT INTO t1 (col1, m2_)
   VALUES (10, { 'm2_fname' : 'Bob', 'm2_lname' : 'Smith' } );
INSERT INTO t1 (col1, m2_)
```



```
   VALUES (11, { 'm2_fname' : 'Bob', 'm2_lname' : 'Roberts', 'm2_weight' :
'160' } );
INSERT INTO t1 (col1, m2_)
   VALUES (12, { 'm2_fname' : 'Dave', 'm2_lname' : 'Smith', 'm2_weight' :
'170', 'm2_gender' : 'M' } );
INSERT INTO t1 (col1, m2_)
   VALUES (13, { 'm2_fname' : 'Angie', 'm2_gender' : 'F', 'political_party' :
'Democrat' } );


SELECT *
FROM t1
WHERE solr_query = '{ "q" : "(m2_fname:Bob OR m2_gender:F)" }' ;
```

Relative to Example 23-4, the following is offered:

- First we add the field type, StrField; an odd edge case, since StrField is used by default for all TEXT data. (Normally this field type is added for us automatically.)
- Then we add the field titled, "m2_", which is defined as a map, TEXT, TEXT.

  Any keys (and values) added to the field "m2_" will be indexed as TEXT.
- We add four rows as shown.
- The SELECT is OR-topped, and returns rows, 10, 11 and 13.

Adding a field neutral query capability

An application requirement detailed above was the ability to query across tags assigned to any lat/lon pair. Example 23-5 details this technique. A code review follows.

Example 23-5 Dynamic schema capability

```
ALTER SEARCH INDEX SCHEMA ON t1
   ADD fields.field[@name='col7',
   @type='my_geo_point', @indexed='true',
   @multiValued='true'];

RELOAD  SEARCH INDEX ON ks_23.t1;
REBUILD SEARCH INDEX ON ks_23.t1;


INSERT INTO t1 (col1, m1_, col7)
   VALUES (1, { 'm1_home' : 'POINT(104.9903 39.7392)', 'm1_work' :
'POINT(104.8214 38.8339)' },
      { 'POINT(104.9903 39.7392)', 'POINT(104.8214 38.8339)' } );
```

```
INSERT INTO t1 (col1, m1_, col7)
  VALUES (2, { 'm1_home' : 'POINT(104.9903 39.7392)'
},
      { 'POINT(104.9903 39.7392)'                          } );
INSERT INTO t1 (col1, m1_, col7)
  VALUES (3, { 'm1_auto' : 'POINT(89.4012 43.0731)'
},
      { 'POINT(89.4012 43.0731)'                           } );
INSERT INTO t1 (col1, m1_, col7)
   VALUES (4, { 'm1_auto' : 'POINT(104.9903 39.7392)', 'm1_home' :
'POINT(104.8214 38.8339)' },
      { 'POINT(104.9903 39.7392)', 'POINT(104.8214 38.8339)' } );
INSERT INTO t1 (col1, m1_, col7)
  VALUES (5, { 'm1_home' : 'POINT(89.4012 43.0731)'
},
      { 'POINT(89.4012 43.0731)'                           } );

SELECT *
FROM t1
WHERE solr_query =
   '{ "q" : "{!geofilt pt=39.7300,104.9900 sfield=col7 d=10    }" }';
```

Relative to Example 23-5, the following is offered:

- The ALTER SEARCH INDEX statement specifies multiValued=true, used rarely, and almost exclusively when using set or list collection types.
- RELOAD, and REBUILD.
- The SELECT returns rows 1, 2 and 4.

23.3 In this document, we reviewed or created:

This month and in this document we detailed the following:

- We detailed indexing a geo-spatial data containing (array), with tags on each value; home, work, other.
- And we detailed querying this same (array) without reference to this tag value.
- Using these same techniques, we provided a polymorphic schema capability similar to MongoDB.

Persons who help this month.

Kiyu Gabriel, Jim Hatcher, Alex Ott, and Caleb Rackliffe.

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
```