CS 548—Fall 2018
Enterprise Software Architecture and Design
Assignment Eleven—CQL

Design the CQL data model for a medical information system. You should provide the schema definition, as a CQL script that defines a data model in a single keyspace, as well as instance data that demonstrates the use of the data model in a Cassandra system. You should provide video(s) that demonstrate creating and populating tables in Cassandra, and successfully querying those tables using `cqlsh` (the Cassandra shell).

A clinic database consists of a collection of patient records, one for each patient. Each patient has:
1. a record identifier (a UUID),
2. a patient identifier (that may be assigned for example by a national government),
3. a patient name, and
4. a date of birth.

Every healthcare provider has a record that includes their record identifier (UUID), their provider identifier (NPI), their name, and an indication of their specialization (surgery, radiology, oncology, etc).

Furthermore each patient and each provider is related by a one-to-many relationship to a collection of treatment records. Every treatment record includes a UUID, the date at which treatment started, the year in which the treatment started (see below) and a diagnosis of the condition for which the treatment is prescribed (e.g. throat cancer, HIV/AIDS, hepatitis, etc). Currently there are three specific forms of treatment records:
1. A drug treatment record includes a drug and a dosage.
2. A surgery treatment record includes the form of surgery.
3. A radiology treatment record includes a list of dates of radiology treatments.

Every treatment record should also identify the patient receiving the treatment (by their patient identifier) and the provider administering that treatment (by their NPI).

For each treatment type, you should define a user-defined type. Since CQL does not support subtyping, you will have to define these as three standalone user-defined types, one for each treatment type. A treatment record should contain columns for common treatment information (treatment UUID, diagnosis, patient identifier and provider NPI), and columns with user-defined types for treatment-specific information. A given treatment record will contain only one non-null column with treatment-specific information.

There are four forms of queries that are intended for this database:
1. List all patients.
2. List all providers.
3. List treatments by patients receiving those treatments.
4. List treatments by providers providing that treatment.

You will therefore need to define four tables, one for each of these queries. For the one-to-many relationships from patients to treatments, and from providers to treatments, you should denormalize the treatment data. However it would not be appropriate to use CQL collection types. Instead represent these relationships using composite columns in the treatment tables. You can use CQL collection types for the list of dates for radiology treatments.

A key design choice in CQL is the choice of the partition key, on the basis of which data is distributed across nodes in the database. If the partition key is too coarse, there may be too much data on a single node, leading to hot spots and potentially exhausting the maximum number of columns allowed on a node. For example, for the table listing treatments by patient, it would be sufficient to use the patient key as a partition key: A single patient will not have that many treatments. However, for the table listing treatments by provider, there is a danger that there will be too many treatments for a provider to store in a single node. This is why treatment information includes the year of treatment (in addition to date of treatment): Use a partition key that includes the provider key and the year of treatment for this table. For both of these tables, use the date of treatment as a clustering key, to order the results within a partition.

Provide your solution as a single CQL script that defines this data model, creates sample data for this data model and does some test queries against this data. The data model script should also define a keyspace (call it cs548) for the data model. You may want to use the Datastax Studio IDE to ensure that your CQL scripts are syntactically valid. This is a Web-based development environment for creating and querying tables in Cassandra (Datastax Enterprise). You should provide a demo video that demonstrates the use of the cqlsh CQL shell to enter data for your data model and perform simple queries in Cassandra e.g.,

```
dsc-cassandra-3.0.0<53> bin/cqlsh
Connected to Test Cluster at 127.0.0.1:9042.
[cqlsh 5.0.1 | Cassandra 3.0.0 | CQL spec 3.3.1 | Native protocol v4]
Use HELP for help.
cqlsh> create keyspace dev
   ... with replication = {'class':'SimpleStrategy','replication_factor':1};
cqlsh> use dev;
cqlsh:dev> create table emp (empid int primary key, emp_first varchar,
emp_last varchar, emp_dept varchar);
cqlsh:dev> insert into emp (empid, emp_first, emp_last, emp_dept) values
(1,'fred','smith','eng');
cqlsh:dev> select * from emp;

 empid | emp_dept | emp_first | emp_last
-------+----------+-----------+----------
     1 |      eng |      fred |    smith
```

Download Datastax Enterprise Edition and Datastax Studio here. There is also a Docker version of Datastax Enterprise (Docker images, Datastax Docker Guide). See also using CQL in Datastax Studio.

Your solution should be uploaded via the Canvas classroom, as a zip file. This zip file should have the same name as your Canvas userid. It should unzip to a folder with this same name, which should contain the files and subfolders with your submission.

It is important that you provide a document that documents your submission, included demonstration videos, a CQL script and a completed rubric for the assignment.