DATASTAX ACADEMY

DS220

03: Additional Apache Cassandra™ CQL Functionality

DATASTAX ACADEMY

Objectives

Over the next few sections, we will cover:

- Why you should know these other CQL features before you create your data model
- How and why to use collection data types
- How UDTs can be implemented for a more flexible model
- Counter columns and their implications
- How UDFs and UDAs can be used together in your data model

DATASTAX ACADEMY

Introduction

DATASTAX ACADEMY

Other Cool CQL Features

Why do you care? Aren't we talking about data modeling?

- Knowing what other functionality there is can factor into your design
- Apache Cassandra™ has additional data types for flexibility
- These include:
  - Collections
  - Counters
  - UDTs
  - UDFs and UDAs

DATASTAX ACADEMY

Why do I care?

These are tools in your arsenal for modeling data!

- Simplifying table design
- Optimize/streamline table functionality
- Store data more efficiently
- Might change table designs completely

DATASTAX ACADEMY

Collections

DATASTAX ACADEMY

Collection Columns

Group and store data together in a column

- Collection columns are multi-valued columns
- Designed to store a small amount of data
- Retrieved in its entirety
- Cannot nest a collection inside another collection—unless you use FROZEN
  - More on FROZEN to come!

DATASTAX ACADEMY

SET Collection Type

Creating and inserting with SET

- Typed collection of unique values
- Stored unordered, but retrieved in sorted ordered
- SET example:

```
CREATE TABLE users (
   id text PRIMARY KEY,
   fname text,
   lname text,
   emails set<text>
);

INSERT INTO users (id, fname, lname, emails)
   VALUES ('cass123', 'Cassandra', 'Dev', {'cass@dev.com',
'cassd@gmail.net'});
```

LIST Collection Type

Altering a table and updating using LIST

- Like SET—collection of values in same cell
- Do not need to be unique and can be duplicated
- Stored in a particular order
- LIST example:

```
ALTER TABLE users ADD freq_dest list<text>;

UPDATE users SET freq_dest= ['Berlin', 'London',
'Paris']
    WHERE id = 'cass123';
```

Collection Column Syntax, Cont.

Creating and inserting with MAP

- Typed collection of key-value pairs—name and pair of typed values
- Ordered by unique keys
- MAP example:

```
ALTER TABLE users ADD todo map<timestamp, text>;
```

```
UPDATE users SET todo = { '2018-1-1' : 'create database', '2018-1-2' :
'load data and test' : '2018-2-1’ : 'move to production '}
   WHERE id = 'cass123';
```

Using Frozen in a Collection

- If you want to nest datatypes, you have to use FROZEN
- Using FROZEN in a collection will serialize multiple components into a single value
- Values in a FROZEN collection are treated like blobs
- Non-frozen types allow updates to individual fields

DATASTAX ACADEMY

UDTs

DATASTAX ACADEMY

User Defined Types (UDTs)

Attach multiple data fields to a column

- User-defined types group related fields of information
- User-defined types (UDTs) can attach multiple data fields, each named and typed, to a single column
- Can be any datatype including collections and other UDTs
- Allows embedding more complex data within a single column

UDT Example

Creating the Types

```
CREATE TYPE address (
  street text,
  city text,
  zip_code int,
  phones set<text>
);

CREATE TYPE full_name (
  first_name text,
  last_name text
);
```

DATASTAX ACADEMY

Using Created UDTs

Using the newly created types in a table definition

```
CREATE TABLE users (
  id uuid,
 name frozen <full_name>,
 direct_reports set<frozen <full_name>>,
 addresses map<text, frozen <address>>,
 PRIMARY KEY ((id))
);
```

DATASTAX ACADEMY

Exercise 3.1: User Defined Types (UDTs)

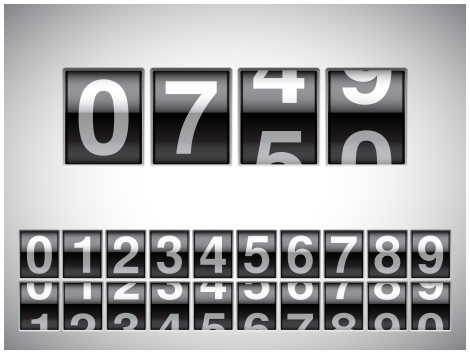
DATASTAX ACADEMY

Counters

Counter Datatype

What is a counter?

- Column used to store a 64-bit signed integer
- Changed incrementally—incremented or decremented
- Values are changed using UPDATE
- Need specially dedicated tables—can only have primary key and counter columns
  - Can have more than one counter column



DATASTAX ACADEMY

Counter Example

Allow queries for number of videos there are per tag

```
CREATE TABLE moo_counts (
  cow_name text,
  moo_count counter,
  PRIMARY KEY((cow_name))
);

UPDATE moo_counts
SET moo_count = moo_count + 8
WHERE cow_name = 'Betsy';
```

DATASTAX ACADEMY

Counter Considerations

Some things to be aware of

- Distributed system can cause consistency issues with counters in some cases
- Cannot INSERT or assign values—default value is “0”
- Must be only non-primary key column(s)
- Not idempotent
- Must use UPDATE command—DataStax Enterprise rejects USING TIMESTAMP or USING TTL to update counter columns
- Counter columns cannot be indexed or deleted

DATASTAX ACADEMY

Exercise 3.2: Using Counters in CQL

DATASTAX ACADEMY

UDFs and UDAs

User Defined Functions (UDFs)

Creating UDFs

- Write custom functions using Java and JavaScript
- Use in SELECT, INSERT, and UPDATE statements
- Function are only available within the keyspace where it is defined

Creating UDFs Syntax

- Enabled by changing the following settings in the cassandra.yaml file
  - Java: Set enable_user_defined_functions to true
  - Javascript and other custom languages:
    Set enable_scripted_user_defined_functions to true

User Defined Aggregates (UDAs)

- DataStax Enterprise allows users to define aggregate functions
- Functions are applied to data stored in a table as part of a query result
- The aggregate function must be created prior to its use in a SELECT statement
- Query must only include the aggregate function itself—no additional columns

DATASTAX ACADEMY

Creating UDF Example

```
CREATE OR REPLACE
    FUNCTION avgState ( state tuple<int,float>, val float )
    CALLED ON NULL INPUT
    RETURNS tuple<int,float>
    LANGUAGE java
    AS 'if (val !=null) {
      state.setInt(0, state.getInt(0)+1);
      state.setFloat(1, state.getFloat(1)+val.floatValue());
    }
    return state;';

CREATE OR REPLACE
    FUNCTION avgFinal ( state tuple<int,float> )
    CALLED ON NULL INPUT
    RETURNS float
    LANGUAGE java
    AS 'float r = 0;
    if (state.getInt(0) == 0) return null;
      r = state.getFloat(1);
      r/= state.getInt(0);
    return Float.valueOf(r);';
```

Create UDA Example

```
CREATE AGGREGATE
    IF NOT EXISTS average ( float )
    SFUNC avgState
    STYPE tuple<int,float>
    FINALFUNC avgFinal
    INITCOND (0,0);
```

Querying with a UDF and a UDA

- The state function is called once for each row
- The value returned by the state function becomes the new state
- After all rows are processed, the optional final function is executed with the last state value as its argument
- Aggregation is performed by the coordinator

```
select average(avg_rating) from videos where release_year = 2002 ALLOW FILTERING;
select average(avg_rating) from videos where title = 'Planet of the Apes' ALLOW FILTERING;
select average(avg_rating) from videos where mpaa_rating = 'G' ALLOW FILTERING;
select average(avg_rating) from videos where genres contains 'Romance' ALLOW FILTERING;
```